

UNIVERSITY OF SOUTHAMPTON

Runtime Energy Management of Multi-core Processors

by

Charles Leech

Thesis submitted for the degree of
Doctor of Philosophy

in the
Faculty of Physical Sciences and Engineering
Electronics and Computer Science

May 2018

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

RUNTIME ENERGY MANAGEMENT OF MULTI-CORE PROCESSORS

by Charles Leech

Performance requirements of emerging applications and tighter power consumption constraints of mobile and embedded platforms mean that runtime management software is required to control these systems efficiently. In order for embedded systems to maintain their optimality, especially in dynamic environments, runtime software must be capable of learning and adaptability. This thesis investigates and develops runtime modelling methods, including their experimental validation, to reduce energy consumption in homogenous and heterogeneous multi-core processors. A multiple linear regression model is established to predict power and performance and drive runtime adaptations of an application and platform to maximise energy efficiency whilst meeting performance targets.

The proposed method is further validated with a parallel stereo matching application, which is developed to investigate the use of core scaling and analyse trade-offs between power and performance through runtime adaptation for energy saving. Experimental results obtained from a 61-core Intel Xeon-Phi platform show that the same performance can be achieved with an average reduction in power consumption of 27.8% and increased energy efficiency by 30.0%, in comparison to baseline dynamic power management techniques.

To make energy management independent of the application and platform, this thesis presents a holistic approach to runtime management in the form of a runtime framework that is both application- and platform-agnostic. The framework unites the hardware and software layers by embedding the runtime management layer at the centre and enables cross-layer interactions through an API and dynamic *knobs* and *monitors*. The framework is demonstrated experimentally across multiple applications and two heterogeneous platforms, the Odroid-XU3 and Cyclone V. Two state-of-the-art runtime management approaches are validated that reduce the energy consumption of application execution by 18.2% and 17.2%. Trade-offs between power, performance and accuracy are presented in three application-platform scenarios.

Contents

List of Figures	9
List of Tables	13
List of Algorithms	15
List of Acronyms	17
Declaration of Authorship	19
Acknowledgements	23
1 Introduction	25
1.1 Research Challenges	27
1.2 Research Questions	28
1.3 Research Contributions	28
1.3.1 Software Contributions	29
1.3.2 Publications	30
1.4 Thesis Structure	31
2 Literature Review	33
2.1 Multiprocessor Systems	33
2.1.1 Symmetric Multiprocessors	34
2.1.2 Circuit-level Technologies	34
2.1.2.1 Dynamic Voltage and Frequency Scaling	35
2.1.2.2 Power Domains	35
2.1.2.3 Caches and Interconnects	36
2.1.3 Heterogeneous Multiprocessor Architectures	37
2.1.3.1 ARM big.LITTLE	38
2.1.3.2 Composite Cores	39
2.1.3.3 Dynamic Core Morphing	40
2.2 Software for Heterogeneous Multiprocessor Systems	41
2.2.1 Dynamic Power Management Software	41
2.2.2 Task Assignment and Allocation	42
2.3 Application Characterisation for HMP Systems	43
2.3.1 Intel RMS Applications	44
2.3.2 Berkeley Motifs	45
2.3.3 The Stereo Matching Application	45
2.4 Runtime Management	47
2.4.1 Runtime Management Algorithms	47
2.4.2 Machine Learning for Runtime Management	49
2.4.3 Model-based Learning Approaches	49
2.4.3.1 Generalised Linear Models	50

2.4.3.2	Support Vector Machines	50
2.4.3.3	Naive Bayes	51
2.4.3.4	Neural Networks	51
2.4.4	Model-based learning for Runtime Management	52
2.4.4.1	Task Mapping and Parallelism	52
2.4.4.2	DVFS in Runtime Management	54
2.4.4.3	DPM in Runtime Management	55
2.4.5	Comparison of Model-based learning Approaches	55
2.5	Runtime Management Frameworks	56
2.6	Experimental Platforms	58
2.6.1	Intel Xeon Phi 7120P	58
2.6.2	Odroid-XU3	60
2.6.3	Cyclone V SoC Development Board	60
2.7	Summary	61
3	Linear regression modelling for runtime power and performance optimisation	63
3.1	Linear Modelling	64
3.2	Runtime Power and Performance Model	65
3.3	Runtime Model Validation	69
3.4	Gradient Decent for Runtime Optimisation	75
3.5	Runtime Manager Overheads	78
3.6	Summary	79
4	Runtime Management of a Parallel Stereo Matching Algorithm	81
4.1	Geometric Model of a Stereo Vision System	82
4.1.1	Pinhole Camera Model	82
4.1.2	Distortion	84
4.1.3	Epipolar Geometry	84
4.1.4	Stereo Geometry	85
4.2	Disparity Estimation	86
4.2.1	Algorithm	87
4.2.1.1	Pre-processing	87
4.2.1.2	Cost Volume Construction	88
4.2.1.3	Cost Volume Filtering	89
4.2.1.4	Disparity Selection	91
4.2.1.5	Post Processing	92
4.2.2	Quantitative Evaluation of Accuracy	92
4.2.3	Multi-threaded Software Implementation	94
4.3	Experimental Characterisation of Performance and Power Consumption	96
4.4	Runtime Management of Stereo Matching	101
4.5	Experimental Results of Online Adaptation	102
4.6	Summary	104
5	A Framework for Application- and Platform-agnostic Runtime Management	107
5.1	Fundamental Concepts of the Cross-layer Framework	109
5.1.1	System Layers	109
5.1.2	Cross-layer Connections	111
5.1.3	The Optimisation of Targets and Constraints	112
5.1.4	Monitor and Application Weighting	113
5.1.5	Knob and Monitor Types	113
5.1.6	Runtime Adaptability	114
5.2	Framework Specification	114

5.2.1	Knobs and Monitors	115
5.2.2	Cross-layer APIs	115
5.2.2.1	Application-RTM Interaction	116
5.2.2.2	RTM-Device Interaction	119
5.3	Framework Software Implementation	120
5.3.1	Namespaces	121
5.3.2	API Interfaces	121
5.3.3	Interprocess Communication	122
5.3.4	Component Classes	123
5.4	Benchmarks	124
5.4.1	Applications	124
5.4.1.1	Jacobi Iterative Method	124
5.4.1.2	Video Decoder	125
5.4.1.3	Whetstone Benchmark	125
5.4.2	Runtime Managers	126
5.4.2.1	Profiler Runtime Manager	126
5.4.2.2	Q-Learning Runtime Manager	126
5.4.2.3	PMC-based Runtime Manager	128
5.4.3	Devices	128
5.4.3.1	Odroid XU3	128
5.4.3.2	Cyclone V	130
5.5	Framework Experimentation Tools	130
5.5.1	User Interface	131
5.5.2	Logger	132
5.6	Experimental Evaluation and Results	133
5.6.1	System Profiling and Trade-off Analysis	133
5.6.2	Runtime Manager Validation	135
5.6.2.1	Q-Learning RTM	137
5.6.2.2	PMC-based RTM	138
5.7	Summary	139
6	Conclusion and Future Directions	141
6.1	Conclusions	141
6.2	Research Questions	143
6.3	Future Investigations	144
	Appendix A RTM Modelling and Analysis Scripts	147
	Appendix B RTM Experimental Data	157
	Appendix C Stereo Matching Algorithm Code	159
	Appendix D Stereo Matching Algorithm Data Analysis Scripts	169
	Appendix E Framework Code	175
	Appendix F Framework Data Analysis Scripts	181
	Appendix G Papers Published and Submitted	189
	Bibliography	281

List of Figures

2.1	System block diagram of a theoretical homogeneous/symmetric multiprocessor, reprinted from [1].	34
2.2	The ARM Cortex-A15 features multiple power domains for the core and surrounding logic, reprinted from [2].	36
2.3	Per-core power domains can reduce power consumption and provide higher performance levels, reprinted from [3].	36
2.4	System block diagram of a theoretical Heterogeneous Multiprocessor (HMP), reprinted from [1].	37
2.5	ARM microarchitectural pipelines of the big.LITTLE HMP, reprinted from [4]. .	38
2.6	Microarchitecture of a Composite Core featuring two backend modules called μ Engines, reprinted from [5].	39
2.7	The process of dynamic core morphing, reprinted from [6].	40
2.8	Workload Convergence from applications (red) to computational kernels (green) to algorithms (yellow) and finally to mathematical primitives (purple), reprinted from [7]. Arrows show how applications can be derived from a common set of computational kernels.	43
2.9	Illustration of how an application can be composed of RMS stages, reprinted from [8].	44
2.10	The Berkeley Motifs and their association with particular applications and application domains, reprinted from [9]. The three levels of need (green=low, yellow=medium, red=high) relate to the rate of occurrence of the motif within the application.	45
2.11	Graphical examples of model-based learning approaches for (A) linear regression and (B) support vector machine classification.	49
2.12	Comparison of energy consumption for benchmarks with and without thread-packing on a homogeneous CMP, reprinted from [10].	56
2.13	Conceptual overviews of frameworks proposed in literature, reprinted from [11, 12, 13, 14].	57
2.14	Intel Xeon Phi diagrams, adapted from [15] and reprinted from [16].	58
2.15	Odroid-XU3 Board Image, reprinted from [17]	59
2.16	Odroid-XU3 system block diagram, adapted from [17]	60
2.17	Altera Cyclone V System on Chip (SoC) Development Board.	61
3.1	Flowchart of the runtime power and performance model generation process. . . .	66
3.2	Plots of measured power (a), performance (b) and energy (c), for the stereo matching application executing on the Xeon Phi, across a range of frequencies and number of cores. Each point is an average of four measurements. Data is listed in Appendix B	67
3.3	Runtime models for power (a) and performance (b) generated from training data.	68
3.4	Effect of the number of training samples on the mean absolute error (A) and the rate of reduction of the error (B) of the power and performance models. Error is the difference between predictions using the models and the remaining experimental characterisation data.	70

3.5	Increasing the number of training samples for both core count and frequency reduces the power and performance modelling error. Mean error is show in red, boxes extend from the first to third quartile with the median line marked.	71
3.6	Analysis of Variance for the latency model.	73
3.7	Analysis of Variance for the current model.	74
3.8	Backtracking line search for the step size of gradient descent, reprinted from [18].	76
3.9	Runtime optimisation examples (a) without performance constraint and (b) with performance constraint. Power consumption is shown using an overlaid colour map and the performance level is shown with contour lines.	77
3.10	Step size, normalised power and normalised performance over the gradient descent iterations.	78
3.11	Timing diagram of the optimisation and measurement overheads of the runtime management processes during application execution.	79
4.1	Pinhole camera model illustrating the projection of a Three Dimensional (3D) point in space to the image plane, reprinted from [19].	83
4.2	The two modes of radial distortion, reprinted from [19].	84
4.3	Epipolar geometry of a stereo imaging system, reprinted from [20].	85
4.4	Coplanar geometry of a calibrated stereo imaging setup reprinted from [20]. . . .	86
4.5	Block diagram of the DE algorithm composed of CVC, CVF, DS and PP stages. The cost volume is created in CVC, with D slides, each slice is filtered in CVF and then combined in DS. The shaded regions are the enhanced parallel stages which offer opportunity for runtime tuning of the threading level.	87
4.6	Left input image, greyscale and x-gradient images produced from pre-processing.	88
4.7	(A) Left and right input images with illustration of the matching process, (B - D) slices through the cost volume after construction with Equation 4.14 at $d = 19$, 33 and 46.	89
4.8	Cost volume slices before (A - B) and after (D - E) filtering with Equation 4.15 at disparities $d = 19$ and 33. The disparity map built from an unfiltered (C) and filtered (F) cost volume.	91
4.9	The affect of post processing illustrated with a comparison between the disparity map (A) after the selection stage, (B) after post processing and (C) the ground truth map from the dataset source.	92
4.10	Masks used to evaluate the accuracy of the disparity map in reference to the ground truth. These are; non-occluded regions (nonocc), all known regions (all) and regions near depth discontinuities (disc)	93
4.11	Comparison of the depth map output from our algorithm and the ground truth depth map provided with the dataset.	94
4.12	The dimensions of parallelism introduced to each stage of this implementation of the Disparity Estimation (DE) algorithm.	95
4.13	Power and performance operating points for the range of cores and frequencies of the Xeon Phi when executing the DE algorithm. The power and performance of each point is an average of four measurements, collected by cycling the algorithm with the image pair from Section 4.2.	97
4.14	Optimisation and scaling actions made in the DE algorithm's operating space on the Xeon Phi platform.	98
4.15	Energy and performance trade-offs for 2^x , ($2 \leq x \leq 5$) and 60 cores over the range of operating frequencies when executing the DE algorithm on the Xeon Phi. . . .	99
4.16	Speed-up in performance of the DE algorithm as a result of core scaling. Speed-up is in relation to the performance at four cores.	100
4.17	Block diagram of the stereo matching run-time optimisation approach	101
4.18	Time series analysis of the Runtime Manager (RTM) performing online adaptations of core number and frequency to optimise power and energy whilst meeting a target performance.	103

5.1	Cross-layer framework and APIs enabling communication between the application, runtime management and device layers using knobs and monitors. The operating system spans all three layers and hosts the execution of each.	110
5.2	Cross-platform and application-agnostic knob and monitor examples used within the framework.	111
5.3	Implementation of the framework from a software-component perspective, including the location of knobs and monitors, interfaces and sockets in each layer. . . .	121
5.4	Namespaces in this implementation of the framework to provided code separation.	121
5.5	Example structure of the runtime management layer with the act-observe-adjust phases highlighted.	127
5.6	Block digram of the tools (UI and Logger) integrated into the framework to aid experimentation.	130
5.7	Main menu of the UI.	131
5.8	Demo mode of the UI.	132
5.9	Auto mode of the UI.	132
5.10	Pareto-optimal states for the error and performance trade-off of the Jacobi application on the Odroid-XU3 and Cyclone V. Label numbers link each operating point to the rows of Table 5.7, which show the knob and monitor values that each point encodes.	134
5.11	Runtime management layer updating application and device knob settings in response to step increases in the performance monitor minimum bound of the Jacobi application running on the Odroid platform. Application monitors and knobs are shown in graphs 1-2 and 3-4, respectively, with device monitors and knobs shown in 5-6 and 7, respectively. The RTM uses a look-up table with a control loop that filters it based on the monitor bounds.	136
5.12	Pareto-optimal states identified by the RTM for the power and performance trade-off of the video decoder application on the Odroid-XU3.	137
5.13	Pareto-optimal states identified by the RTM for the temperature and performance trade-off of the Whetstone application on the Odroid-XU3.	137
5.14	Comparison between Q-learning RTM and Ondemand governor for the power consumption of the video decoder executing on the Odroid.	138
5.15	Top: Comparison between PMC-based RTM and Ondemand governor for the power consumption of the video decoder executing on the Odroid. Middle and lower: the MRPI metric and the frequency choices of the PMC-based RTM and Ondemand governor.	139

List of Tables

2.1	Governors available in the Linux kernel since version 2.6 [21].	41
2.2	Summary of runtime management algorithms published in the literature.	48
2.3	Grouping of existing model-based runtime learning techniques for power and energy management	52
2.4	Voltage and frequency scaling steps of the Xeon Phi processors.	59
3.1	MLR modelling hypotheses	65
3.2	Summary of the statistical properties of the latency model.	72
3.3	Summary of the statistical properties of the current model.	72
3.4	Data for each step in the gradient descent search for performance optimisation of the stereo matching application executing on the Xeon Phi platform.	77
3.5	Gradient descent search data for optimisation of the stereo matching application with a performance constraint, executing on the Xeon Phi platform.	78
4.1	Comparison of the accuracy of related stereo matching algorithms using standard image pairs from the Middlebury database [22].	94
4.2	Normalised power and performance operating points. Traversing between these performs optimisation or scaling actions.	98
4.3	Optimised operating points, sampled from Figure 4.18, that minimised power consumption under each performance target.	104
5.1	Structure of application and device knobs in the cross-layer framework.	114
5.2	Structure of application and device monitors in the cross-layer framework.	115
5.3	Complete set of framework API functions.	117
5.4	Application-level knobs and monitors present in Jacobi (Section 5.4.1.1), video decoder (5.4.1.2) and Whetstone (5.4.1.3).	125
5.5	Available device-level knob and monitor types.	129
5.6	Device-level knobs and monitors for Odroid-XU3 (Section 5.4.3.1) and Cyclone V (5.4.3.2) platforms.	129
5.7	Knob and monitor values for Pareto-optimal points of the Odroid platform in Figure 5.10.	134
B.1	Characterisation data for a multi-core platform and multi-threaded application, including the power consumption and performance, in FPS, resulting from a particular frequency and number of active cores.	157
B.2	Characterisation data continued from Table B.1.	158

List of Algorithms

3.1	R code to generate the power and performance models of Figure 3.3a and 3.3b. .	69
3.2	R code to test the number of training samples required for error convergence. The full script can be found in Listing A.2 of Appendix A.	70
4.1	Block threading approach to throttle parallelism at the disparity level.	95
5.1	API functions required to register an application (presented in Section 5.4.1.1) and create application-level knobs and monitors.	116
5.2	API functions required to get updated values of application knobs and set updated monitor values through the framework.	118
5.3	API functions required to deregister an application and its knobs and monitors from the framework.	119
5.4	API functions for the RTM to register the device and its knobs and monitors. . .	119
5.5	The RTM pulling monitor values from the device and pushing device knob settings through the API.	120
5.6	Example JSON code for an application knob registration message.	122
5.7	Discrete knob registration case in the message handler function of the app interface of the RTM layer.	123
5.8	Extract from the logger output of an experiment using the framework.	132
A.1	R code for Multiple Linear Regression (MLR) analysis and the plotting of Figure 3.3.147	
A.2	R code for MLR error convergence analysis and the plotting of Figure 3.4.	148
A.3	Complete R code for gradient decent analysis and the plotting of Figure 3.9 and 3.10.	150
A.4	Python script to plot experimental characterisation data for Figure 3.2, 4.15 and 4.16.	153
C.1	Top-level code for the disparity estimation algorithm.	159
C.2	Power measurement and energy recording code for the Xeon Phi platform. . . .	162
C.3	RTM code for modelling and optimisation of the stereo matching application and Xeon Phi platform.	164
D.1	Python script to generate power, performance and energy operating points from experimental data for Figure 4.13 and 4.14.	169
D.2	Python script to plot experimental data for the RTM optimisation in response to changing performance targets in Figure 4.18.	172
E.1	API header for common types in the PRiME Framework.	175
E.2	Common application types API header.	175
E.3	Common device types API header.	176
E.4	Application API header, containing the <code>rtm_interface</code>	177
F.1	Time series plotting and processing script for standard logger output. Used to plot Figure 5.11.	181
F.2	Python class <code>mon_ops</code> used to store and calculate monitor operating points. . . .	185

List of Acronyms

2D	Two Dimensional.
3D	Three Dimensional.
ADSW	Adaptive Support Weight.
ANOVA	Analysis of Variance.
API	Application Programming Interface.
CFS	Completely Fair Scheduler.
CMP	Chip Multiprocessor.
CPU	Central Processing Unit.
CVC	Cost Volume Construction.
CVF	Cost Volume Filtering.
DCM	Dynamic Core Morphing.
DCT	Dynamic Concurrency Throttling.
DE	Disparity Estimation.
DPM	Dynamic Power Management.
DS	Disparity Selection.
DSP	Digital Signal Processor.
DVFS	Dynamic Voltage and Frequency Scaling.
EAS	Energy Aware Scheduler.
FPGA	Field-Programmable Gate Array.
FPS	Frames Per Second.
GIF	Guided Image Filtering.
GLM	Generalized Linear Models.

GPU	Graphics Processing Unit.
HD	High Definition.
HMP	Heterogeneous Multiprocessor.
HPC	High Performance Computing.
IO	Input/Output.
IPC	Instructions per Cycle.
ISA	Instruction Set Architecture.
JSON	JavaScript Object Notation.
MIC	Many Integrated Core.
ML	Machine Learning.
MLR	Multiple Linear Regression.
MRPI	Memory Reads Per Instruction.
NN	Neural Network.
OLS	Ordinary Least Squares.
OpenCL	Open Compute Language.
OS	Operating System.
PID	Process Identifier.
PMC	Performance-monitoring Counter.
PP	Post Processing.
pthread	POSIX threads.
QoE	Quality of Experience.
QoS	Quality of Service.
RMS	Recognition, Mining and Synthesis.
RTM	Runtime Manager.
SBC	Single Board Computer.
SGM	Semi Global Matching.
SMP	Symmetric Multiprocessor.
SoC	System on Chip.
SVC	Support Vector Classification.

SVM	Support Vector Machines.
SVR	Support Vector Regression.
TDP	Thermal Design Power.
UDS	Unix Domain Socket.
UI	User Interface.
VR	Virtual Reality.

Declaration of Authorship

I, Charles Leech, declare that this thesis entitled Runtime Energy Management of Multi-core Processors and the work presented in it are my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work that have been published are listed under the research contributions.

Signed:

Date:

Acknowledgements

I would firstly like to extend my thanks to my supervisors; principally Professor Bashir M. Al-Hashimi for his guidance and direction throughout my PhD, Dr Geoff V. Merrett for his supervision and advice at many times and Dr Tom J. Kazmierski for providing me with the opportunity to undertake this research. My thanks also go to the Engineering and Physical Sciences Research Council (EPSRC) for funding my work over the course of the PhD.

A great amount of time has been spent in the presence of a unique group of colleagues within and around the PRiME Project, both past and present. Some notable individuals include; Dr Graeme Bragg, Dr Domenico Balsamo, Dr Monica Glanc-Gostkiewicz, Matt Walker, Dr James Davis, Dr Sheng Yang, Dr Josh Levine and many more. It has been fantastic to work with such a charismatic group of people. In addition, I am thankful to all the members of the wider PRiME Project for creating a stimulating environment for ideas and discussion, as well as to all those who contributed to multiple productive collaborations.

Thank you to all my friends who have been around for entertainment, relaxation and plenty of special life events; Alex and Rachel Clayton for being great housemates, good cooks and the best friends; Jon New and Kate Rawlings for the impromptu chats, films and welcome distractions; Matthew Posner and Becky Bellworthy for getting me into cycling, pioneering the best cycling club in town and having great taste in coffee; and Rhys Thomas and Laurie Pestana for their passion for cycling, their warm friendship and all the cafe stops. A few more special mentions to some influential figures in my brief cycling career go to Tj Key and Trevor Allen, for bringing out the competitive side in me and James Robards and Chris Brooks for both having impeccable style. Thank you all for the miles and smiles along the way.

I would like to say a big thank you to my parents; Diana and Roger and my brother James, for shaping my upbringing, making me into who I am today, and providing many welcome escapes.

Finally, the greatest thanks are reserved for my wife, Rachel who has always believed in me and given me her endless love, support and understanding through our life-long journey together.

Chapter 1

Introduction

Processors have evolved over the decades due to a complex combination of influences and driving forces. From advancements in fabrication technology and improvements in software capabilities, to the emergence of new computing platforms, next-generation applications and changes in the demands of users.

The development of processor architectures can be characterised into three distinct eras. From the 1970s, a single-threaded era focused on the primary target of improving performance of a single processor, achieved through transistor scaling, improvements in fabrication and increased processor clock speeds. For many years, these techniques maintained the pursuit of Moore’s Law and satisfied the demands of applications at the time. Dubbed “the free lunch”, these applications benefited from uncompromising improvements in performance and processing speed with each generation of processors [23].

However, with technological advancements pushing to the boundaries of physics, a lack of clock scaling stalled the previously freely-available performance improvements. Subsequently, the second multi-core era exploited advances in fabrication techniques to allow the integration of multiple processors within a single chip. From then on developments in processor technology focused on core scaling as a priority. In reaction, the advent of parallel processing brought about changes throughout the entire software stack, with new programming models and paradigms designed to extract parallelism from applications.

This trend stalled due to an exponential increase in the power density of chip multiprocessors. High power consumption from fast clock rates combined with increasing transistor densities pushed thermal dissipation rates in excess of operating levels. As a result, only portions of a chip could function at any one time, in order to manage thermal levels and prevent damage to circuitry. This phenomenon has been nicknamed *dark silicon*, in a similar manner to the inactivity of dark energy, because the inactive parts of the chip exist but cannot operate due to thermal implications [24].

In the current era of processor development, the electronics industry looks to new innovations to continue delivering improved performance with each generation of devices. The focus now is on

achieving the best balance between energy and performance, a target that is being most successfully achieved through heterogeneous architectures containing asymmetric processors. Heterogeneous multiprocessor architectures have the potential to deliver both speed and energy efficiency through adaptability and targeted execution. With performance asymmetry, a heterogeneous multiprocessor implements a different trade-off between energy efficiency and performance for each processing resource or core. Smaller cores are used to process simple tasks in an energy-efficient way while larger cores provide higher-performance processing for more compute-intensive tasks. The introduction of functional asymmetry further expands the range of compute modes to highly-parallel and custom-accelerator processors. The mix of compute resources now supports a diverse collection of programming models, code structures and application types.

Furthermore, advances in system-on-chip integration and transistor scaling have successfully driven the reduction in size of electronic systems, leading to the emergence of entirely new platforms and environments in mobile and embedded computing. Devices operating in these domains are fundamentally governed by their energy efficiency, which is now considered as a design priority rather than an afterthought or side-affect. Moreover, the performance capability of processors has increased far in excess of the developments in battery capacity. This has led to a large reduction in the usage time for mobile devices from the order of days to hours. In addition, consumers demand an ever-improving user experience, with higher-performance and more graphically-intensive mobile applications, necessitating the use of high frequency multi-core processors to meet tighter performance deadlines. Now the prominent design challenge for these devices has become the development of effective methods for minimising power consumption as the energy efficiency of processors and embedded systems must be maximised in order to maintain battery life.

The increase in performance capability of mobile devices, due to heterogeneous multiprocessor architectures, has opened up this sector to applications that require the variety of processing resources available to compute tasks. The array of technological features and sensors now embedded on these devices allows them to deliver a rich and immersive experience that is interactive and connected to the user. A notable example is Virtual Reality (VR) where the user experiences a simulated environment different from that of their current surroundings. Primarily used as an entertainment experience, for example, to complete puzzles or play games. The experience requires a user to wear a headset projecting 3D visuals and headphones providing surround-sound. Accelerometer-based motion sensors in the headset track movement of the head in all directions and update the display and sound perspective to match. For this application, highly-effective real-time processing is required to deliver an adequate Quality of Experience (QoE) to the user. Otherwise, any lag in the display or sound will render the entire experience invalid as the conflicting sensual environment is unpleasant for the user. Moreover, as a mobile device, that must be ergonomic and lightweight, a constraint is placed on the power consumption and form factor of the embedded electronics. These factors combine to require a system that is highly-electronically advanced, involving high-speed sensing, complex interpretation and High Definition (HD) video streaming, but must also be lightweight, comfortable and energy efficient, presenting a significant engineering challenge.

With the development of heterogeneous architectures comes the need to identify and characterise the properties of applications, in order to most effectively execute them across the multitude of specialised resources. Intel's Recognition, Mining and Synthesis (RMS) and the Berkeley Motifs

are two prominent taxonomies that have emerged to meet this need, with differing approaches that are explored in more detail in Section 2.3. The taxonomies provide a framework for characterisation so that the constituent elements of applications such as computer vision and data analytics can be identified and consolidated.

To ensure that characterised applications execute on the necessary processing resources, given the asymmetry in heterogeneous architectures, runtime management software is an essential component in the system stack. The Runtime Manager (RTM) is responsible for ensuring that the performance requirements of all the applications executing on the platform are met, whilst minimising physical quantities of the platform such as energy consumption or temperature. The degree to which the RTM achieves the balance between these orthogonal objectives is known as the Quality of Service (QoS) delivered.

Runtime management can be achieved by any process that controls and adapts the behaviour of a platform and/or application during execution. Approaches can be classified using the mathematical techniques that they use, from closed-loop control to machine learning, or the power management technologies that they control, including frequency scaling and task allocation. RTM approaches that have been implemented in the literature are presented and evaluated in Section 2.4, with a focus on machine learning methods, which show greatest suitability to this task due to their high accuracy and adaptability.

Balancing accuracy with adaptability is an important consideration when developing a runtime management approach, due to the wide range of heterogeneous platforms and applications. For any approach to be suitable for the mobile and embedded domains it must be able to adapt to changes in application requirements as well as operate with unknown applications and platforms. Learning at runtime is therefore essential under these circumstances.

A cross-layer framework is proposed in this thesis to support runtime learning and promote properties such as adaptability, application-agnosticism and ensure that runtime software is cross-platform. Platform and application hooks that control tunable parameters are captured as generic constructs called *knobs* and important performance metrics or physical quantities are captured as *monitors*. Monitors allow observation and learning of system behaviour, to ensure that performance requirements are met and energy is optimised. Knobs enable the RTM to be adaptive and control both the platform and applications in an agnostic manner. An Application Programming Interface (API) is defined as a specification of the concepts encompassed by the framework. This can be used by application, RTM and platform developers so that their components may be used in any circumstance.

The challenges associated with developing a runtime energy management approach, optimising complex applications and formulating a framework for runtime management are discussed in the following section.

1.1 Research Challenges

The energy consumption optimisation of multi-core systems at runtime is an established area of research, with Dynamic Power Management (DPM) technologies being implemented in many

mobile and embedded devices. These approaches can respond to fluctuations in the load of the system, however they are limited in that they possess no predictive capabilities. This is a requirement in order to manage energy and performance trade-offs due to the fact that knowledge of the potential operating space is necessary, *i.e.* what energy savings exist and how to achieve them. Modelling energy-performance trade-offs in multi-core systems at runtime is therefore necessary in order to identify optimisation opportunities. Moreover, the continuous optimisation and prediction of energy consumption becomes increasingly difficult in the presence of dynamic application requirements, behaviour that is common in mobile and embedded workloads.

Furthermore, the majority of runtime management approaches are specific to a single, group or domain of applications and/or devices, which limits their applicability in the general case. In order to generalise these approaches, an environment must be established where the mechanisms for runtime management are exposed in a standardised way. However, this environment must still meet the prerequisite that the runtime management approach remains able to operate as effectively as it did in the specific case.

1.2 Research Questions

To overcome the challenges outlined, the need for runtime energy management in multi-core systems is evident and motivates this research to address the following questions:

1. How can a multi-core system be modelled at runtime to optimise energy efficiency and performance requirements?
2. What are the potential energy and performance trade-offs when applying runtime energy management to next-generation mobile and embedded applications?
3. How can the runtime management of energy and performance be made more cross-platform and application-agnostic? How can this be extended to the management of other properties?

1.3 Research Contributions

In seeking to address the research questions posed, this thesis makes several contributions to the research in this area.

1. The development and validation of a modelling approach, based on MLR, that can be trained at runtime using a minimum number of samples to predict the power and performance of a multi-core system and multi-threaded application. This is coupled with a runtime optimisation method based on gradient descent to optimise platform and application settings under a performance target. This contribution has been published in ACM TECS as Leech *et al.* [25]

2. Deployment of the runtime management approach to optimise the execution of a multi-threaded implementation of the disparity estimation algorithm for stereo matching, including characterisation of the operating space of the application on a multi-core platform with scaling and trade-off analysis. Moreover, runtime modelling and optimisation of the application is carried out in response to specified performance targets. This contribution has also been published in ACM TECS as Leech *et al.* [25]
3. The creation of a cross-layer framework for application- and platform-agnostic runtime management. Specification of an API for cross-layer communication of information through knobs and monitors exposed by applications and platforms. The realisation of a software implementation of the framework including experimentation tools and data analysis methods. The validation of the framework with three applications, two RTMs and two heterogeneous multi-core platforms.

The work presented in this thesis is aligned with the objectives of the PRiME Project (Power-efficient, Reliable, Many-core Embedded systems), with which the author is a member. The PRiME Project is an EPSRC funded research programme developing the theory and practices for future high-performance embedded systems utilising many-core processors. PRiME's objective is to enable processor core scaling with sustainable energy consumption and reliability.

This thesis has addressed aspects of these objectives through the runtime energy management approach that is present in Chapter 3. The application presented in Chapter 4 and the platforms in Chapter 5 are examples of next-generation applications and high-performance embedded systems, for which new optimisation practices have been developed. Due to the collaborative nature of the project, some of the contributions stated were made as part of team projects. The development of the cross-layer framework was a collaborative effort with researchers from within the project at both the University of Southampton and Imperial College London. As a result, the author was at least significantly involved in the work for any contributions stated. In addition, it is possible that the code shown in some listing was not written wholly by the author, as this is the nature of collaborative software development, however it is required in order to ensure the clarity of the listing. A detailed breakdown of the contributions made by the author to the development of the framework is given in the introduction to Chapter 5.

1.3.1 Software Contributions

Two significant software contributions have been made during the course of the PhD and are described here.

PRiMEStereoMatch: The stereo matching application, containing the disparity estimation algorithm, was developed to perform the experimentation presented in Chapter 4. This was extended during the course of the project into a significant software program. To enable others to use this software and provide a contribution to the research community, the application has been made publicly available under an open-source license through a GitHub repository. This can be accessed at the address: <https://github.com/PRiME-project/PRiMEStereoMatch>

PRiME_Framework: The software implementation of the cross-layer framework, which was used to perform the experimentation in Chapter 5, has also been made publicly available under

an open-source license in the same manner. One of the fundamental purposes of the framework is to encourage research into runtime management to be conducted in an application- and platform-agnostic environment that enables the comparison and evaluation of the effectiveness of different approaches. The framework also represents a research enablement tool, allowing researchers to focus on solving runtime management challenges by separating the development process into layers, connected using the API. The project is released as the PRiME Framework as is it a collaborative project within the PRiME Project. The framework can be accessed at the address: https://github.com/PRiME-project/PRiME_Framework

1.3.2 Publications

The contributions of the research presented in this thesis have also been published as follows:

- Leech, Charles, Vala, Charan Kumar, Acharyya, Amit, Yang, Sheng, Merrett, Geoff V. and Al-Hashimi, Bashir (2017) Run-time performance and power optimization of parallel disparity estimation on many-core platforms *ACM Transactions on Embedded Computing Systems* [25]
- Tenentes, Vasileios, Leech, Charles, Bragg, Graeme, Merrett, Geoff V., Al-Hashimi, Bashir, Amrouch, Hussam, Henkel, Jörg and Das, Shidhartha (2017) Hardware and software innovations in energy-efficient system-reliability monitoring In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*. IEEE. 5 pp. [26]
- Singh, Amit, Leech, Charles, Basireddy, Karunakar Reddy, Al-Hashimi, Bashir and Merrett, Geoff V. (2017) Learning-based run-time power and energy management of multi/many-core systems: current and future trends *Journal of Low Power Electronics* [27]
- Vala, Charan Kumar, Immadisetty, Koushik, Acharyya, Amit, Leech, Charles, Balagopal, Vibishna, Merrett, Geoff V. and Al-Hashimi, Bashir (2017) High-speed low-complexity guided image filtering-based disparity estimation *IEEE Transactions on Circuits and Systems I: Regular Papers*. [28]
- Leech, Charles and Kazmierski, T J (2014) Energy Efficient Multi-Core Processing *ELECTRONICS*, 18, (1), pp. 3-10. [29]
- Leech, Charles, Bragg, Graeme, Davis, James, Constantinides, George, Merrett, Geoff V. and Al-Hashimi, Bashir (2018) The PRiME Framework: Application- and Platform-agnostic Runtime Management *Transactions on Computer-Aided Design of Integrated Circuits and Systems* (Under Review)

The following publications were also made during the course of the PhD, but are not directly reported as contributions in this thesis:

- Leech, Charles, Raykov, Yordan P., Ozer, Emre and Merrett, Geoff V. (2017) Real-time room occupancy estimation with Bayesian machine learning using a single PIR sensor and microcontroller At *IEEE Sensors Applications Symposium (SAS)*, Glassboro, NJ, 2017, pp. [30]

1.4 Thesis Structure

The thesis is structured as follows. Chapter 2 presents a review of research into hardware and software innovations around homogeneous and heterogeneous multiprocessor systems. Furthermore, it explores emerging applications and methods to support their implementation on these platforms. The chapter presents model-based machine learning approaches and their applicability to the runtime energy management of multi-core systems. In addition, the chapter looks beyond these to runtime management frameworks, which aim to provide a common approach to runtime management. The chapter concludes with information on the platforms used for experimentation in chapters 3 to 5.

Chapter 3 presents an MLR model-based runtime management approach applicable to multi-core systems. The effectiveness of the model is validated empirically and a gradient-descent optimisation approach is proposed to utilise the model at runtime and select optimal operating settings.

Leading on from this, Chapter 4 documents the development of a stereo matching application, based on the disparity estimation algorithm, for depth calculation using a stereoscopic camera. The objective is to develop a scalable application that is amenable to runtime management on multi-core platforms, to deliver both energy-efficient and high-performance operation. This chapter covers exploration of the application's operating space on a homogeneous many-core platform, which motivates the need for runtime management. Runtime modelling and adaptation is demonstrated using the approach developed in the previous chapter.

In order to facilitate application- and platform-independent runtime learning and adaptation, Chapter 5 presents a cross-layer framework for system-level control and monitoring. The framework exposes parameters from both the application and device as knobs and monitors. The operation of the framework and its accompanying API library are demonstrated through proof-of-concept examples. Results show how controlling the exposed parameters allows trade-offs to be made between power, accuracy and performance.

Finally, Chapter 6 concludes the thesis with reflections on the contributions made in the preceding chapters to runtime energy management, the optimisation of next-generation applications and the development of a standardised framework to support application and platform-agnostic runtime management. Potential future directions for the work and the wider research field are discussed.

Chapter 2

Literature Review

In order to overcome the research challenges described, and address the research questions posed, a deeper understanding of the established knowledge and current research in the relevant areas is required. Therefore, this chapter presents a review of modern platforms; next-generation applications and state-of-the-art runtime energy management techniques.

The drive to increase the energy efficiency of mobile and embedded systems has lead to many advancements in the design of hardware platforms. Section 2.1 presents a review of the circuit- and architectural-level technologies implemented in modern multi-core/multiprocessor systems to improve energy efficiency. In addition, software that is required to control or compliment these technologies is presented, including DPM methods, task allocation schema and programming models.

Innovations in hardware and software are driven by the requirements of next-generation applications. The capabilities of homogeneous and heterogeneous multi-core embedded systems have enabled many applications to move into this space. Section 2.3 presents an investigation into these applications and benchmarks in order to understand their properties and establish characterisation schemes to improve their implementation on heterogeneous multi-core systems.

Linking hardware and software innovations together, Section 2.4 presents the literature surrounding runtime management, including runtime algorithms for the optimisation of next-generation applications on multi-core platforms. There is a focus on model-based machine learning methods, which have the potential to enable model-building at runtime for the online prediction and optimisation of power, energy and performance. Runtime model-building can help to address the challenge of adapting to dynamic changes in application requirements and platform constraints. Lastly, the platforms that are used for experimentation in this thesis are presented in Section 2.6, to provide technical information about their architecture and specification.

2.1 Multiprocessor Systems

This section presents the research relating to energy-efficient computing using Symmetric Multiprocessor (SMP) and HMP systems. Through the evaluation of current research and technology,

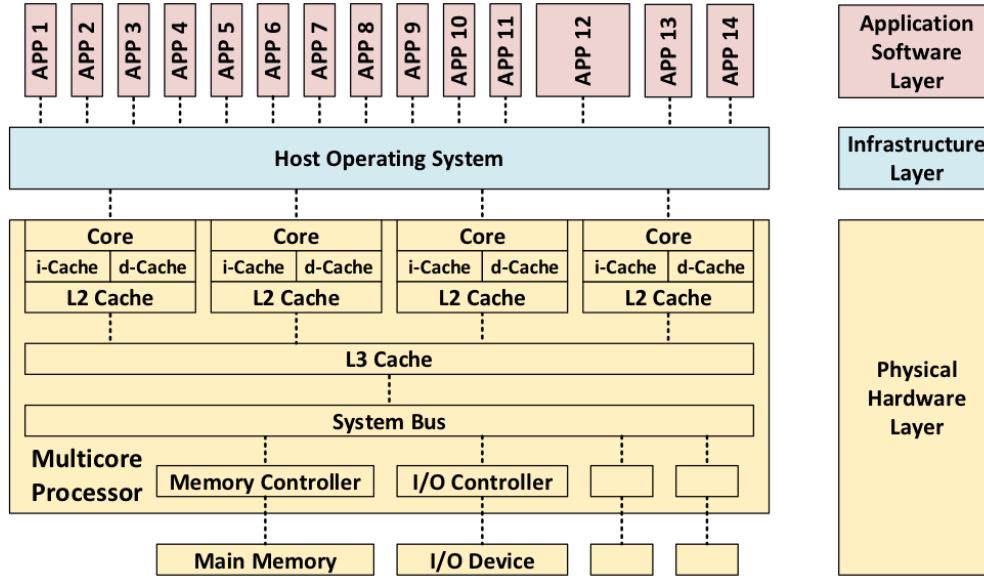


FIGURE 2.1: System block diagram of a theoretical homogeneous/symmetric multiprocessor, reprinted from [1].

this chapter aims to understand where novel contributions can be made and motivate the experiments conducted in the following chapters.

2.1.1 Symmetric Multiprocessors

In an SMP, each processor has an identical microarchitecture, which can perform the same operations, shares the same memory hierarchy and has the same Input/Output (IO) facilities as all the other processors within the Central Processing Unit (CPU). Figure 2.1 illustrates the typical structure of an SMP architecture and depicts a quad-core processor, each with a private i-cache, d-cache and L2 cache. A shared last-level cache is commonly implemented in commercial SMPs systems, L3 in this scenario. A system bus or other interconnect is used to connect the memory subsystem to IO and external components in the system. An advantage of the uniformity in SMP architectures is that it is relatively simple to implement a single Operating System (OS) image that spans all the processors, this is shown as the infrastructure layer in Figure 2.1 and contains the software and firmware that directly interacts with the hardware layer. Furthermore, a single OS image enables applications to be developed for, and execute in, a common software environment. The OS can schedule threads across all the processors uniformly, therefore they are transparent to the application [31].

2.1.2 Circuit-level Technologies

This section describes a set of technologies that are commonly applied to electronic circuits, including multiprocessor systems, as a means of achieving higher energy efficiency with minimal performance loss. Many technologies, such as clock gating and power domains, can be applied

generically across all of the system's circuitry, to improve both static and dynamic power consumption. Others require operating system control and run-time management to extract greater dynamic efficiency from the processor.

2.1.2.1 Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) is a DPM technique that allows the power consumption of a processor to be optimised through fine-grain adjustment of the supply voltage and/or clock frequency [32, 33, 34, 35]. When there is time slack in the performance demands of an application, lower voltage and frequency levels are used to allow the application to be executed more energy efficiently. Instructions are executed more slowly due to the increased clock period and the delays of critical paths in the hardware are extended due to longer propagation times. Higher DVFS levels are used when performance constraints are tight, but energy efficiency is sacrificed as a result. Multi-core processors can implement per-core DVFS mechanisms in order to maximise energy efficiency through independent regulation of the levels of each core. Significant energy savings are demonstrated by Wonyoung *et al.* who investigate such a system [35]. Homogeneous multi-core architectures can emulate performance heterogeneity with per-core DVFS control because each core will have a different performance and power consumption specification depending on the current DVFS level used. DVFS is controlled by the runtime manager, allowing it to provide the best QoS for the system in running a particular application.

DVFS is the primary mechanism used by the runtime management approach presented in Chapter 3, to control the power consumption and throughput of the platform. For the case study presented in Chapter 4, the runtime manager controls the DVFS subsystem of the Intel Xeon Phi platform (presented in Section 2.6.1) to change the frequency of all the cores at runtime.

2.1.2.2 Power Domains

Power domains are regions of a processor that can operate across several voltage supply levels in order to regulate power consumption. Domains typically encompass functional blocks or logic circuitry but they can scale to entire cores. Sub-domains are used to allow smaller logic sections to be deactivated for finer performance and power variations of the chip.

The ARM Cortex-A15 MPCore processor supports multiple power domains for each region of the processor chip [2]. Figure 2.2 shows the larger domains, labelled Processor and Non-Processor, which allow large parts of the processor to be regulated or deactivated. Example sub-domains are shown by the boxes within each larger domain, such as CK_GCLKCR. Power domains are often coupled with power modes as a means of gradually switching on or off several power domains, to enter low power, idle or shutdown states. The Cortex-A15 features multiple power modes with specific power domain configurations such as Dormant mode, where some Debug and L2 cache logic is powered down but the L2 cache RAMs are powered up to retain their state, or *Powerdown* mode where all power domains are powered down and the processor state is lost [2].

In multi-core architectures, power domains can be used to power down individual cores when idle or during non-parallel tasks in order to manage power consumption. Sinkar *et al.* [3] and Ghasemi *et al.* [36] present low-cost, per-core voltage domain solutions in order to improve

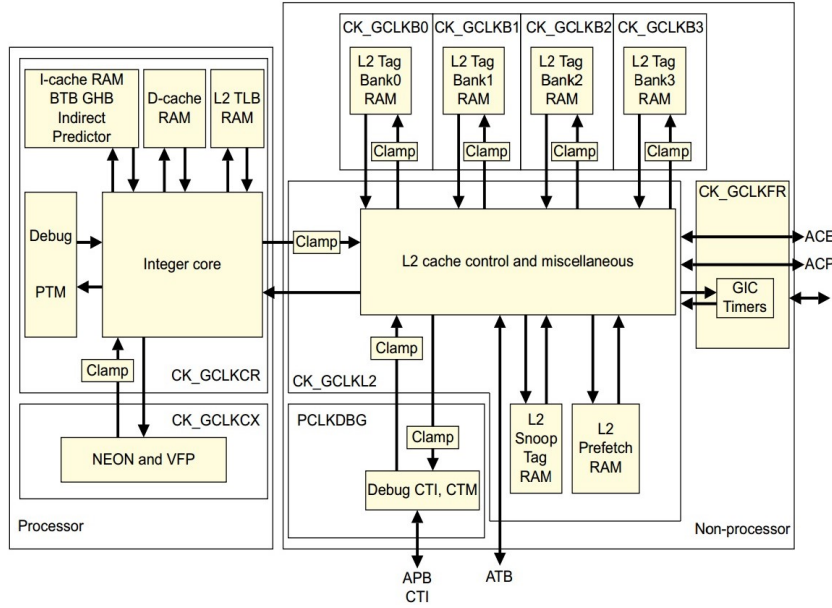


FIGURE 2.2: The ARM Cortex-A15 features multiple power domains for the core and surrounding logic, reprinted from [2].

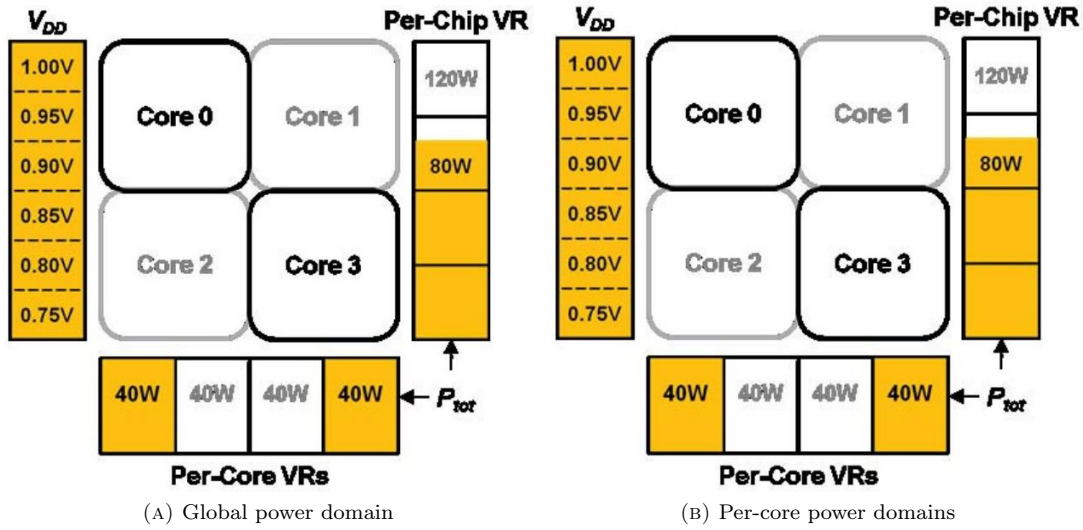


FIGURE 2.3: Per-core power domains can reduce power consumption and provide higher performance levels, reprinted from [3].

performance in power-constrained processors. Figure 2.3 shows how Sinkar's mechanism can provide reduced chip power consumption but maintain per-core performance. In Figure 2.3a, a chip-wide power domain is shown with all cores active at the same level. In contrast, Figure 2.3b shows a per-core power domain that allows unnecessary cores to be powered down, and active cores to provide a higher performance level, while still reducing the overall chip power level.

2.1.2.3 Caches and Interconnects

It is not only the design of the processor's internal circuitry that is important in maintaining energy efficiency. The co-design of the interconnect, caches and the processor cores is required

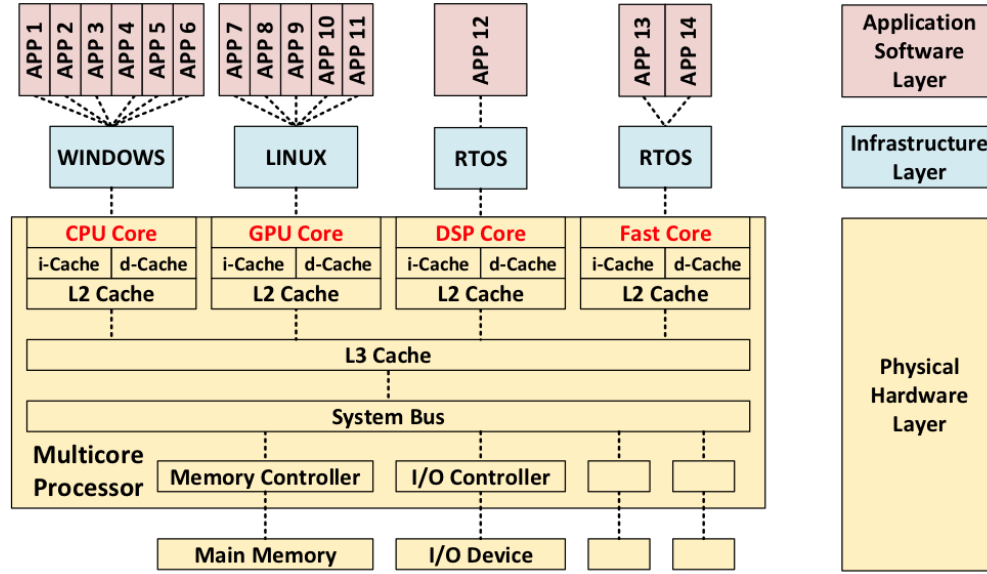


FIGURE 2.4: System block diagram of a theoretical HMP, reprinted from [1].

to achieve high performance and energy efficiency. Kumer *et al.* examine parameters such as the area, power and bandwidth costs required to implement the processor-cache interconnect, showing that large caches sizes can constrain the interconnect's size and large interconnects can be power-hungry and inefficient [37]. Zeng *et al.* also recognise the high level of integration that is inherent in Chip Multiprocessors (CMPs) and attempt to reduce the interconnect power consumption by developing a novel cache coherence protocol [38]. Furthermore, Basmadjian *et al.* develop a methodology for estimating the power consumption of multi-core processors [39], taking into account resource sharing and power saving mechanisms on top of the power consumption of each core.

2.1.3 Heterogeneous Multiprocessor Architectures

A heterogeneous or asymmetric multiprocessor architecture is one composed of cores of varying sizes and complexities which are designed to compliment each other in terms of performance and energy efficiency. A typical system will employ smaller cores to process simple tasks in an energy efficient way and larger cores to provide high-performance processing for when computationally-demanding tasks are presented. A task matching or switching system can also be implemented to intelligently assign tasks to cores; balancing a performance demand against maintaining system energy efficiency. These systems are particularly good at handling a diverse workload where fluctuations of high and low computational demand are common.

A heterogeneous architecture can be created in many different ways and many alternatives have been developed due to the heavy research interest in this area. Modifications to general-purpose processors, such as asymmetric core sizes [40], custom accelerators [41], varied caches sizes [42] and heterogeneity within each core [5, 6], have all been demonstrated to introduce heterogeneous features into a system. The hardware and software architecture of a theoretical HMP is shown in Figure 2.4. Multiple operating systems or kernels may be implemented, running simultaneously to manage each resource.

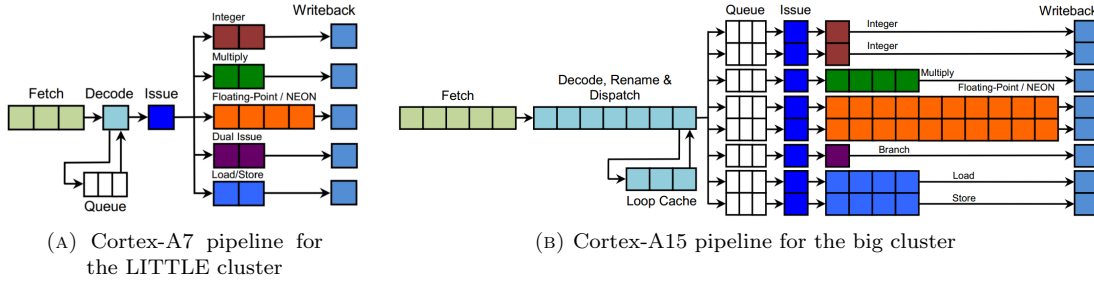


FIGURE 2.5: ARM microarchitectural pipelines of the big.LITTLE HMP, reprinted from [4].

For heterogeneous multi-core systems, it is important to consider what combination of core sizes will lead to the most optimal system. Moreover, this can be defined for a multitude of factors such as energy efficiency, throughput or area efficiency. Kumar *et al.* evaluate a range of combinations of two Alpha cores in terms of power efficiency, where the core combinations represent different points in a power/performance design space [43]. The best compromise between the two factors is chosen, but often their choice is constrained by power or performance requirements.

In HMP architectures, a single Instruction Set Architecture (ISA) can be retained to create a performance asymmetric architecture (although not truly heterogeneous), ensuring core-to-process compatibility and enabling seamless task migration. Kumar *et al.* demonstrate this in their research, where two performance-asymmetric Alpha RISC architectures are combined to be more energy and area efficient than a homogeneous equivalent [44, 43]. They claim significant energy benefits by dynamically allocating application execution to the most appropriate core, indicating a 39% average energy reduction for only a 3% performance drop [44].

2.1.3.1 ARM big.LITTLE

The ARM big.LITTLE architecture is a production example of a HMP system consisting of an area- and energy-efficient “LITTLE” Cortex-A7 quad-core processor cluster coupled with a higher-performance “big” Cortex-A15 quad-core processor cluster [4]. The system is designed with the dynamic usage patterns of modern smart phones in mind where there are typically periods of high-intensity processing followed by longer periods of low-intensity processing [45]. Low-intensity tasks, such as texting and audio, can be handled by the Cortex-A7 processor enabling a mobile device to save battery life. When a period of high intensity occurs, the Cortex-A15 processor can be activated to increase the system’s throughput and meet tighter performance deadlines. A power saving of up to 70% is advertised for a light workload, where the Cortex-A7 processor can handle all of the tasks, and a 50% saving for medium workloads where some tasks will require allocation to the Cortex-A15 processor. Both processors implement the same ARMv7 architecture and only differ in their microarchitectures, where they use different combinations of the same architectural blocks. Figure 2.5a shows that the Cortex-A7 is an in-order, dual-issue processor with a pipeline length of between 8 and 10 stages, sacrificing performance for energy and area efficiency. On the other hand, Figure 2.5b shows that the Cortex-A15 is complimentary to this, implementing an out-of-order, sustained triple-issue processor with a pipeline length of between 15 and 24 stages that can deliver a significantly higher throughput.

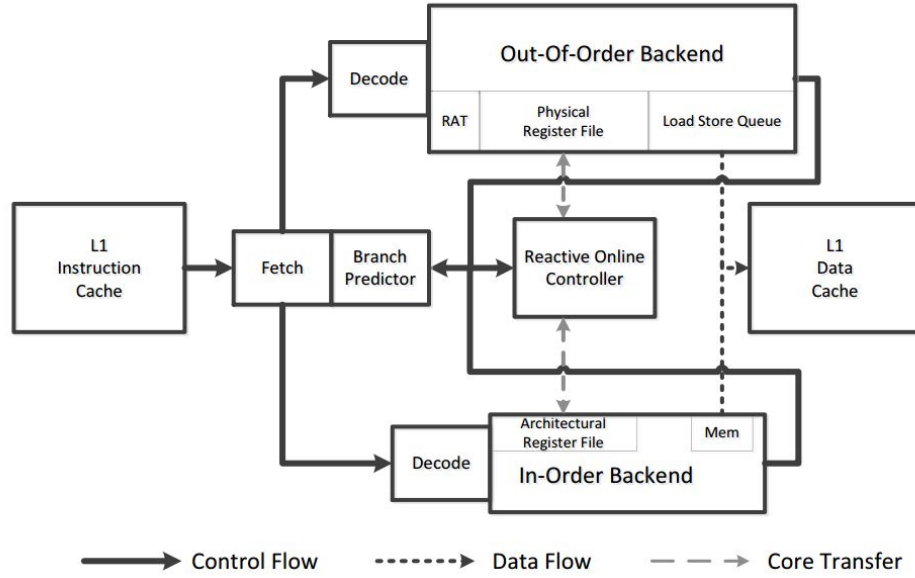


FIGURE 2.6: Microarchitecture of a Composite Core featuring two backend modules called μ Engines, reprinted from [5].

The big.LITTLE architecture is central to the Samsung Exynos 5422 SoC, which is found on the Odroid-XU3 development board presented in Section 2.6.2. This platform is used for the experimentation presented in Chapter 5 to demonstrate runtime energy management of the hardware and to validate the cross-platform capabilities of the framework.

2.1.3.2 Composite Cores

Heterogeneous systems are not limited to varying core sizes. Composite Cores is a microarchitectural design that brings heterogeneity inside each individual core in order to reduce the migration overhead of task switching [5]. The unique feature of this design is that it contains two separate backend modules, called two μ Engines, one of which features a deeper and more complex out-of-order pipeline, tailored for higher performance, while the other features a smaller, compact in-order pipeline designed with energy efficiency in mind. Instructions are dispatched to either backend based on a performance estimation that is matched to the task's requirement and determines whether migration is necessary. Due to the high level of hardware resource sharing and the small μ Engine state, the migration overhead is brought down from the order of 20,000 instructions to 2000 instructions. Figure 2.6 shows the microarchitecture of a composite core, with arrows showing how data and control flow is permitted between modules.

The fine level of granularity by which the system can migrate a task enables it to exploit more rapid fluctuations in execution requirement. For example, a demanding task may be switched to the smaller μ Engine if the task's structure is periodically limiting its throughput such that the large μ Engine is performing it inefficiently. The task can then be returned to the large μ Engine once the structure of the task can utilise the a higher processing capability again. Their results show that the system can achieve an energy saving of 18% using dynamic task migration whilst only suffering a 5% performance loss. This indicates that greater energy efficiency can

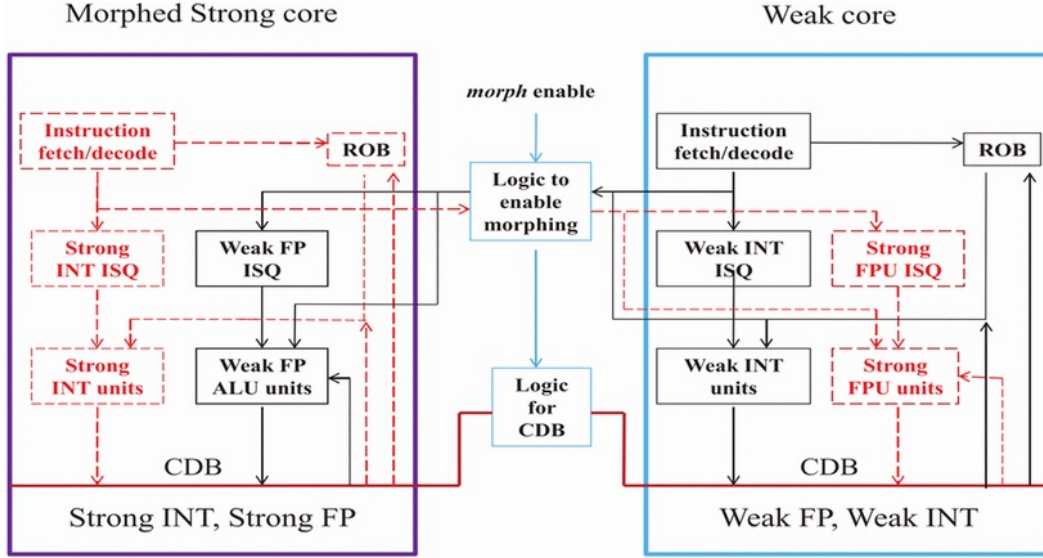


FIGURE 2.7: The process of dynamic core morphing, reprinted from [6].

be extracted from heterogeneous architectures, such as the ARM big.LITTLE, by considering heterogeneity below the core level.

2.1.3.3 Dynamic Core Morphing

Using both a heterogeneous architecture and hardware reconfiguration, a technique called Dynamic Core Morphing (DCM) was developed by Rodrigues *et al.* to allow the shared hardware of a few tightly coupled cores to be morphed at runtime [6]. The cores all feature a baseline configuration, allowing them to perform moderately at all tasks, with the addition of greater strength in a particular area such as integer or floating-point arithmetic. A hardware algorithm monitors the computational demands of the current application and can trigger reconfiguration to re-assign functional units to different cores to speed up execution. A diagram of this process is shown in Figure 2.7 where the reconfiguration network enables the morphed core to become stronger at floating-point operations in addition to its baseline configuration. Alternatively, software monitoring the allocation of threads to cores may choose to swap a thread to a core with computational strengths matching those required for the current program phase. The efficiency of the system can lead to performance-per-watt gains of up to 43% and an average saving of 16% compared to a static or homogeneous architecture. Core morphing is also implemented by Das et al. [46] so that the two cores of a heterogeneous processor can be fused into a single core when the application requires it. Otherwise, the two cores will remain separate to operate more energy efficiently and can run individual threads concurrently.

TABLE 2.1: Governors available in the Linux kernel since version 2.6 [21].

Name	Description & Operation
Performance	Sets the CPU statically to the highest frequency within the borders of <code>scaling_min_freq</code> and <code>scaling_max_freq</code> .
Powersave	Sets the CPU statically to the lowest frequency within the borders of <code>scaling_min_freq</code> and <code>scaling_max_freq</code> .
Userspace	Allows the user, or any userspace program running with UID <code>root</code> , to set the CPU to a specific frequency through the sysfs file <code>scaling_setspeed</code> .
Ondemand	Sets the CPU frequency depending on the current system load. Load estimation is triggered by the scheduler through the <code>update_util_data->func</code> hook; when triggered, <code>cpufreq</code> checks the CPU-usage statistics over the last period and the governor sets the CPU accordingly.
Conservative	Sets the CPU frequency depending on the current system load, but it gracefully increases and decreases the CPU speed rather than jumping to max speed in response to load on the CPU. This behaviour is more suitable in a battery powered environment.

2.2 Software for Heterogeneous Multiprocessor Systems

Advances in multiprocessor architectures must be complimented by developments in software that will execute on and manage these systems. Heterogeneous architectures present a step-change for many established programming principles and has led to new research into the development of software tools, programming models and programming frameworks to increase the utilisation and efficiency of systems based on these architectures. This chapter presents recent advancements in software for HMP systems including software for dynamic power management at the userspace level, such as frequency governors, schedulers and task allocation/assignment approaches.

2.2.1 Dynamic Power Management Software

Frequency Governors: Many multiprocessor SoCs have the ability to scale the frequency of each processing resource dynamically in order to save power, as described in Section 2.1.2. Frequency scaling can be performed automatically in response to changes in system load, in response to particular events, or it can be controlled manually. In the Linux kernel, frequency control is made accessible to userspace software via a kernel module and driver called *cpufreq* with scaling policies known as *governors* that package together parameters determining the policies' behaviour. All of the standard governors supplied with the Linux kernel since version 2.6 are described in Table 2.1. Governors must specify a minimum, maximum and target frequency. Governors that automatically adjust the CPU's frequency can determine the target frequency with additional parameters, as is the case with *Ondemand* and *Conservative*. Further details of these parameters and their function can be found in the *cpufreq* documentation [21].

Process Schedulers: The process scheduler is a component of the operating system that controls where software processes execute on a platform. The decision of where to schedule a

particular task is determined by a scheduling policy or algorithm. Examples policies include; first-come, first served, earliest deadline first, round-robin and random scheduling. The Completely Fair Scheduler (CFS) was introduced to the Linux 2.6.23 kernel and is based on an implementation of the weighted fair queuing algorithm. This is a throughput based policy which aims to ensure that processor time is maximised among the running or waiting tasks in as fair a manner as possible.

The Energy-Aware Scheduling Project, jointly developed by ARM and Linaro [47, 48], is an approach to power management that coordinates the scheduler and cpufreq driver to improve power saving and remove any conflict between the two previously isolated subsystems [49]. The scheduler is aware of the current and near-future utilisation of the CPUs because it controls where tasks will be allocated. However, the frequency governors can only operate reactively, changing the frequency after a change in utilisation. By connecting the two subsystems, an Energy Aware Scheduler (EAS) can achieve greater energy efficiency through combined actions such as task consolidation onto fewer CPUs or frequency scaling during task migration [48]. The EAS also integrates several improvements over the Linux scheduler, including an awareness of performance asymmetric CPUs, such as in the big.LITTLE architecture presented in Section 2.1.3. This changes the way that CPU capacity is understood by the scheduler as utilisation can change differently in response to frequency scaling depending on the workload and the CPU architecture.

2.2.2 Task Assignment and Allocation

The energy efficiency benefits of heterogeneity can only be exploited with the correct assignment of tasks or applications to each core [50, 51, 52, 53, 54]. Tasks must be assigned in an order that maximises energy efficiency whilst ensuring performance deadlines are met. Awan *et al.* perform scheduling in two phases to improve energy efficiency; task allocation to minimise active energy consumption and then an exchange of higher-energy states to lower, more energy-efficient sleep states [50]. Alternatively, Calcado *et al.* propose division of tasks into m-threads to introduce fine-grained parallelism below thread level [55]. Moreover, Saha *et al.* include power and temperature models into an adaptive task partitioning mechanism in order to allocate tasks according to utilisation rather than worst case execution time [54].

Task assignment can also be performed in response to program phases, which naturally occur when the resource demands or execution patterns of an application change. Phase detection is used by Jooya and Analoui to dynamically re-assigning programs for each phase to improve the performance and power dissipation of HMPs [52]. Programs are profiled in dynamic time intervals in order to detect phase changes. Sawalha *et al.* also propose an online scheduling technique that dynamically adjusts the program-to-core assignment as application behaviour changes between phases with an aim to maximise energy efficiency [53]. Simulated evaluation of the scheduler shows energy saving of 16% on average and up to 29% reductions in energy-delay product can be achieved compared to static assignments.

Several high-level management strategies have been implemented on heterogeneous systems. For example, the Invasive Computing approach, devised by Teich *et al.* [56][57], presents a resource-aware programming paradigm that allows the dynamic exploration and exploitation of hardware resource by program agents.

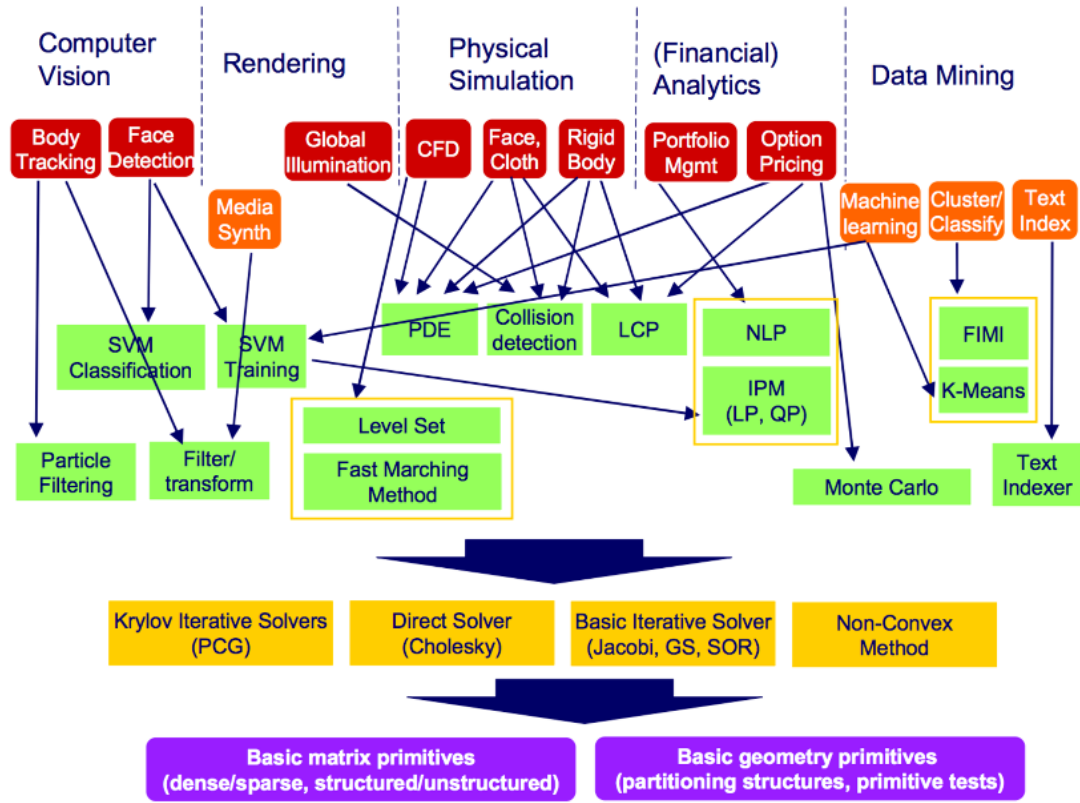


FIGURE 2.8: Workload Convergence from applications (red) to computational kernels (green) to algorithms (yellow) and finally to mathematical primitives (purple), reprinted from [7]. Arrows show how applications can be derived from a common set of computational kernels.

2.3 Application Characterisation for HMP Systems

This section focuses on investigating the characteristics of applications and the underlying intrinsic properties that connect them. Application analysis provides a top-down approach to the development of a system. Applications are important both driving and demonstrating research by providing relevance to real world problems. Moreover, benchmarks are the programmatic realisation of the algorithms that underpin applications. They contain the actual code that will be used to test, evaluate and demonstrate the system and as a result, selection and analysis of suitable benchmarks is equally important in order to ensure identified applications are correctly represented during demonstration.

Research of applications has been conducted for two main reasons. Firstly, to discover the applications and domains that would benefit from execution on HMP systems. This is especially important when the objective is to optimise the application and platform's operation to suit application requirements. Secondly, analysis has been used to identify ways in which cross-layer interactions can be realised; where the application and hardware provide information to each other and additional layers in the system. All the layers must cooperate to maximise the system's energy efficiency at all points in time, therefore communication of information vertically is required.

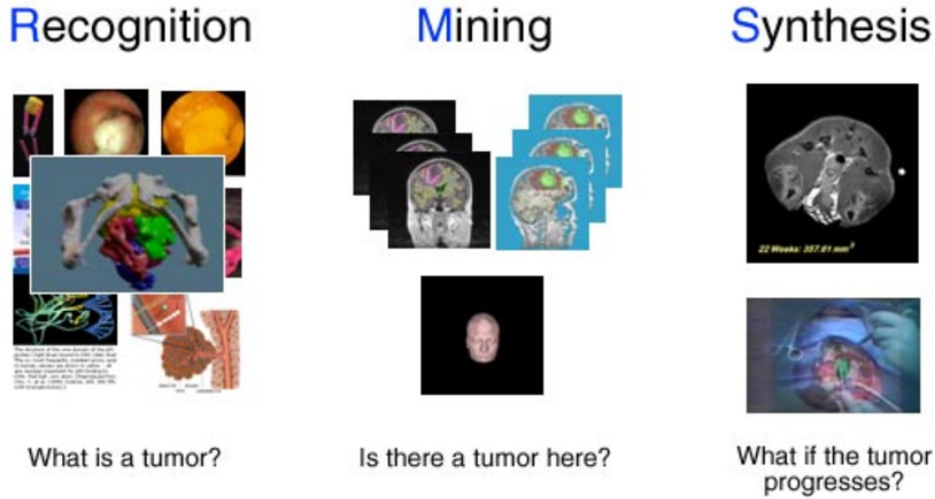


FIGURE 2.9: Illustration of how an application can be composed of RMS stages, reprinted from [8].

Application characterisation is a promising area of exploitation in order to correctly design and manage HMP systems [58, 59, 60, 61]. Characterisation provides a framework for workload converge whereby many applications can be captured by a few commonalities in properties such as structure or content [62, 63]. This convergence is illustrated in Figure 2.8, which shows how workloads and applications share a common sets of kernels, models or algorithms. Characterisation can be utilised to provide QoS requirements to the OS and RTM based on exact properties of the application. Given the structure of the system, this information can enable accurate direction of application tasks to the appropriate cores or processing elements, increasing energy efficiency and performance. Two leading theories into application characterisation are discussed in the following sections.

2.3.1 Intel RMS Applications

RMS applications characterise a wide spectrum of applications into three fundamental processing capabilities; Recognition, Mining and Synthesis [64]. Recognition is derived from the idea that applications are required to have an understanding of their function in order to make informed decisions. This is typically realised through the construction of mathematical models. Mining capabilities are shown in a application's ability to process data and identify information or patterns of significance. The final stage of processing is in the synthesis of data, from which applications derive much of their value. This involves the exploration of theoretical scenarios through the analysis of constructed models. This could be in order to make predictions or take logical actions. These three compute capabilities can be seen explicitly in Figure 2.9 for the understanding, identification and prediction of the development of brain tumours. Currently, the compute requirements for many of these applications, which are often required to operate in real-time, are too great for mobile and embedded systems.

HMPs mark a step towards realising the next generation of compute-intensive applications on mobile and embedded systems. Many applications iterate through the three RMS stages discretely and as such the mathematical models, numerical algorithms and underlying data structures that






	Embed	SPEC	DB	Games	ML	CAD	HPC					
1 Finite State Mach.												
2 Circuits												
3 Graph Algorithms												
4 Structured Grid												
5 Dense Matrix												
6 Sparse Matrix												
7 Spectral (FFT)												
8 Dynamic Prog												
9 Particle Methods												
10 Backtrack/ B&B												
11 Graphical Models												
12 Unstructured Grid												

FIGURE 2.10: The Berkeley Motifs and their association with particular applications and application domains, reprinted from [9]. The three levels of need (green=low, yellow=medium, red=high) relate to the rate of occurrence of the motif within the application.

are used can be extracted to tune execution to match the platform configuration [8]. Properties such as the memory behaviour and program structure of applications can be characterised to determine architectural features and direct task execution [65, 61]. RMS workloads are valuable due to the fact that many of the common algorithms scale, therefore applications can exploit multi-core systems through parallel programming models.

The stereo matching application, presented in Chapter 4, can be used as a case study for application characterisation. It represents a recognition stage in RMS due to the fact that it performs analysis of input images to extract the depth of each pixel. The scalability of the underlying algorithm in the application is also examined in Chapter 4. This increases its adaptability and shows that it is possible for RMS applications to operate in real-time on multi-core systems.

2.3.2 Berkeley Motifs

The Berkeley Motifs represent another example of workload convergence through application characterisation, using similar principles to RMS applications [7, 66]. The Motifs are a development of the RMS concept into 12 “benchmarks” (refined from the original 13 [9]) that are not specific to certain applications but instead represent patterns of computation and communication that are found across a wide range of applications, or map to certain application domains. The heatmap of Figure 2.10 connects the motifs to examples of applications that make use of them, across 3 levels of need, and show how commonly they are found in several applications domains including healthcare and image processing. The RMS application structure ties in closely with the Berkeley motifs, where both represent methods of characterisation for applications.

2.3.3 The Stereo Matching Application

Stereo matching is a process for depth estimation and 3D sensing that has been used across many embedded applications including person counting and tracking [67], autonomous navigation and

obstacle avoidance [68, 69] and mobile robotics [70, 71]. The core algorithm that is implemented to perform the matching process is called Disparity Estimation (DE).

At the highest level, DE can be categorised into global and local algorithms. Global algorithms are formulated as an optimisation across parts of an entire image. They produce precise results, with low average error rates in the calculation of disparity values [22]. However, they typically have complex implementations with high memory and hardware demands, which have the potential to limit scalability to higher resolution images. As a result, investigations have been made to implement dedicated hardware architectures of more precise algorithms, such as Semi Global Matching (SGM) [72, 73] and Adaptive Support Weight (ADSW) filtering [74, 75].

In contrast, local stereo matching algorithms have reduced computational complexity and more localised memory requirements, relying on simpler aggregation strategies [76, 77]. However, these algorithms are prone to disparity errors at depth discontinuity regions due to the use of a fixed local window shape and size [78]. To improve matching accuracy, a few attempts have been made by combining or modifying existing algorithms and transforms [79, 80, 81], the most recent ADSW methods are currently the most accurate [72, 74, 75]. These work by assigning different weights to the pixels in a support window based on their colour or proximity to the central pixel. In this way, they aggregate only those pixels that lie at the same disparity, leading to improved quality at depth borders [78].

Recently, the use of a Guided Image Filtering (GIF) [82] in local ADSW algorithms has been proposed to reduce the complexity of cost aggregation, leading to a high-quality, fast and simple local DE algorithm [83]. Due to the reduced complexity of this type of filter, the algorithm can operate at real-time frame-rates for HD images when implemented in a parallel structure [76]. This has resulted in the migration of software implementations entirely into the hardware domain on FPGA accelerators [72, 73, 76].

However, fixed hardware designs lack the ability to perform adaptations at runtime, which is important for any application operating on an embedded system where power and performance scalability is a key attribute. A local algorithm is chosen for the experimentation in this thesis because it is scalable when implemented in software due to implicit parallelism and low data-dependence properties. ADSW and GIF enhancements ensure a high-quality disparity map in terms of bad pixel errors without sacrificing the algorithm's parallelism. Scalability enables operation across a range of power-performance points, depending on the system's constraints. Furthermore, the memory and computational resource requirements of embedded systems prevent the implementation of Global and SGM algorithms due to their irregular data access patterns and high-complexity algorithms [73].

The approach considered in this thesis can also be categorised as a passive stereo vision method, relying on correspondence between a stereoscopic image pair. Alternative, active stereo vision approaches use a projector-camera setup with a single light source, such as infrared in the case of the Microsoft Kinect [84, 85], or a structured light array [86, 87] to perform depth estimation. Passive and active approaches are not directly comparable as they operate on different data sources.

2.4 Runtime Management

The term runtime management encompasses any process designed to control the operational behaviour of a system (platform and applications) during the execution of software. This is important from a platform energy efficiency perspective, firstly to extend the operational lifetime of mobile devices that are battery powered; secondly, to avoid the unnecessary wastage of energy which can increase running costs; and thirdly to reduce the platform's surface temperature.

Furthermore, applications often have varying performance requirements and execution behaviours depending on program phases, data dependencies and IO interactions. As a result, proactive optimisation of performance and energy is a key challenge, especially on heterogeneous platforms. Runtime management represents an essential paradigm in tackling these challenges by enabling optimisation and the trade-off between computational quality, application throughput, system reliability and energy efficiency during execution.

2.4.1 Runtime Management Algorithms

An increasing number of runtime management algorithms are being employed to control and optimise the execution of applications on heterogeneous embedded systems [90, 96, 100]. These approaches can be broadly classified by criticality (hard or soft real-time), optimisation technology or the employed learning and control method [27]. DPM technologies are widely used in embedded platforms to reduce power consumption when workload requirements change. Table 2.2 presents a summary for a selection of runtime management approaches, published in literature, to demonstrate the variety in the choice of learning method/algorithm, control mechanism and optimisation parameters. The learning method/algorithm encompasses any process that contributes to the runtime management of the platform and/or application. Optimisation parameters represent the target properties of the system, which the approach attempts to directly or indirectly optimise as a result of runtime management. In addition, particular emphasis is made to determine whether training of the approach is necessary and when this occurs. Whether offline and/or online training is required will impact on the adaptability of the approach to new platforms or applications. Offline training requires additional time and prerequisite knowledge of the target platform and/or applications. Online training avoids this drawback but can still require time at the start of execution to training parameters or learn system behaviour, which can result in a period of degraded performance or higher power consumption.

The control mechanism employed by the RTM to manage the platform or application in each approach is identified in Table 2.2, to understand the technologies used. The most common of these is DVFS, which allows an OS to change the processor's voltage and clock frequency during process execution, usually in accordance with manufacturer-determined voltage-frequency ($V-f$) pairs. Runtime algorithms controlling DVFS are commonly based on the utilisation of processor cores, in a similar manner to the Linux *cpufreq* governors. More advanced DVFS control in runtime management algorithms is often based on machine learning. This is designed to either build predictive models of the system [101, 102, 103] or use reinforcement learning to iteratively adapt to changes in workload using historical data [94, 96, 104]. More sophisticated DPM approaches incorporate mapping and scheduling algorithms. In multi-threaded applications, the level of parallelism can be tuned using concurrency throttling [10, 105, 106]. For heterogeneous

TABLE 2.2: Summary of runtime management algorithms published in the literature.

RTM Author	Learning Method/Algorithm	Training	Control Mechanism	Optimisation Parameters
Das <i>et al.</i> [88, 89]	Reinforcement learning	Online	Task allocation + DVFS	Temperature, reliability, energy
Wildermann <i>et al.</i> [90]	Invasive computing [56] + game theory	Offline	Task allocation + Core shutdown	Performance
Bhatti <i>et al.</i> [91]	Machine learning (NN-based)	Offline	DVFS & DPM	Power, energy, performance
Cochran <i>et al.</i> [10]	Multinomial logistic regression classifier	Offline	Thread packing + DVFS	Temperature, power, performance
Curtis-Maury <i>et al.</i> [92]	Multivariate linear regression	Offline	DCT (OpenMP) + DVFS	Power, energy, performance
Tzilis <i>et al.</i> [93]	Simulated annealing + genetic algorithm	Offline	Task allocation	Reliability, performance
Wang <i>et al.</i> [94]	Reinforcement learning	Online	Task allocation + DVFS	Energy
Lama <i>et al.</i> [95]	Model predictive control	Offline	Task allocation + DVFS	Power, performance, energy
Shafik <i>et al.</i> [96]	Reinforcement learning	Online	DVFS	Power, performance, energy
Wang <i>et al.</i> [97]	Reinforcement learning + Bayesian Classification	Offline + Online	DPM	Power, performance
Yang <i>et al.</i> [98]	Linear regression + gradient decent	Online	Task allocation + DVFS	Power, energy, performance
Salinas-Hilburg <i>et al.</i> [99]	Linear regression + Grammatical Evolution	Offline + Online	Task allocation	Power, energy

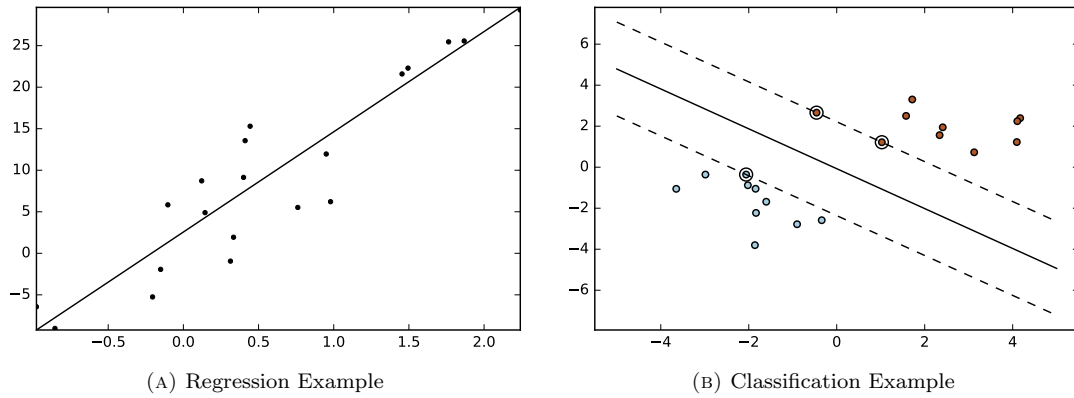


FIGURE 2.11: Graphical examples of model-based learning approaches for (A) linear regression and (B) support vector machine classification.

platforms, applications or their sub-processes may be assigned to specific hardware resources to optimise energy efficiency, leading to runtime resource management [89, 98, 107, 100, 108]. This is advantageous when portions of application code express differing computational behaviour at runtime, for example the level of data parallelism.

2.4.2 Machine Learning for Runtime Management

The aim of model-based learning is to make predictions about future responses based on evidence in the presence of uncertainty. Supervised machine learning is a collective term for a group of algorithms that perform predictive modelling to establish a relationship between a set of predictor (independent) variables and a target (dependent) variable. In this section, each of the algorithms is introduced and discussed in the context of learning-based runtime power or energy management strategies. Following this, the modelling approaches are grouped by the optimisation and control methods that they employ; categorised into mapping, DVFS and/or DPM.

2.4.3 Model-based Learning Approaches

Most supervised machine learning algorithms can be engineered to operate as either classification or regression techniques, where:

- Classification techniques predict discrete responses. Classification models classify input data into categories. Typical applications include medical imaging, speech recognition, and credit scoring.
- Regression techniques predict continuous responses. Typical applications include electricity load forecasting and algorithmic trading.

2.4.3.1 Generalised Linear Models

The most common group of predictive algorithms used for learning-based runtime management are Generalized Linear Models (GLM). This term encompasses the majority of empirically and analytically derived models of performance or power commonly derived for prediction in management systems. Ordinary Least Squares (OLS) is an example of a GLM algorithm and is based on a linear combination of the predictor variables in the form of a hypothesis function that is defined as;

$$\hat{y}(\omega, x) = \omega_0 + \sum_{i=1}^n \omega_i x_i \quad (2.1)$$

The coefficients ω_i are established using a series of j training samples $(x_{i,j}, y_j)_{i=1\dots n, j=1\dots m}$ with i predictor variables. The OLS process minimises the mean-squared prediction error $E(\omega)$ of the model, expressed as:

$$E(\omega) = \sum_{j=1}^m (f_\omega(x_j) - y_j)^2 \quad (2.2)$$

Future target values \hat{y} are predicted given new data x_{n+1} . Figure 2.11a illustrates how a OLS regression function is calculated from the training data points such that the mean-squared error is minimised.

GLM models can be built using many different algorithms besides OLS. Ridge regression addresses some of the problems of OLS by imposing a penalty on the size of coefficients, therefore the ridge coefficients minimise a penalised residual sum of squares. Further alternatives include Lasso, least angle, Bayesian and polynomial regression. Logistic regression is a GLM algorithm used for classification rather than regression and is also known in literature as logit regression or maximum-entropy classification. In this model, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.

2.4.3.2 Support Vector Machines

Support Vector Machines (SVMs) are a set of supervised learning methods that construct a hyperplane or set of hyperplanes in a high-dimensional space, which can be used for classification, regression or other tasks such as outlier detection. In Support Vector Classification (SVC), the model is a representation of the example points in space, mapped so that the categories have maximal separation. This is achieved using a hyperplane that has the largest distance to the nearest training data point of any class (the functional margin). The larger the margin the lower the generalisation error of the classifier. New data points are mapped into the same space and predicted to belong to a category based on which side of the hyperplane they fall into. The model is trained using a dataset of n points in the form $(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)$ where y_i are either 1 or -1 , each indicating the class to which the point \vec{x}_i belongs. Each \vec{x}_i is a p -dimensional vector. The maximum-margin hyperplane, which divides the group of points \vec{x}_i for which $y_i = 1$ from the group of points for which $y_i = -1$, is defined so that the distance between the hyperplane and the nearest point \vec{x}_i from either group is maximised. Any hyperplane can be written as the set of points \vec{x} satisfying $\vec{w} \cdot \vec{x} - b = 0$ where \vec{w} is the normal vector to the hyperplane. Figure 2.11b illustrates the output of the SVC process with the hyperplane (solid line) and the

functional margins (dashed lines) shown. The training data points are shown with the bounding points highlighted.

Support Vector Regression (SVR) uses the same principles as the SVC, but the intention is to develop a function and hyperplane that minimises deviation of y_i for all training data [109]. As with classification, the algorithm takes input vectors \vec{x} and y , but in this case y is expected to have floating point values instead of integer values. The model produced depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

SVMs have several advantages; they are effective in high-dimensional spaces, even in cases where the number of dimensions is greater than the number of samples. They use a subset of training points in the decision function (called support vectors), so are memory efficient. They are versatile because different kernel functions can be specified for the decision function in order to classify data more appropriately. However, if the number of features is much greater than the number of samples, the method is likely to give poor accuracy. Also, SVMs do not directly provide probability estimates for each classification, these must be calculated using additional cross-validation methods.

2.4.3.3 Naive Bayes

Naive Bayes is a simple technique for performing classification and can build models that assign class labels to instances of features, where the labels are identified from some finite set. Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features.

An advantage of naive Bayes is that it only requires a small amount of training data to estimate the parameters necessary for classification. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of predictor variables in the learning problem.

2.4.3.4 Neural Networks

Neural Networks (NNs) build a computational model based on a large collection of connected units called neurons, analogous to axons in a biological brain. Connections between neurons carry an activation signal, which can also be weighted to affect the strength of connections and the probability of activation. Neural networks must be trained from examples, rather than explicitly programmed, and excel in areas where the solution or feature detection is difficult to express as a traditional computer program. With sufficient training, NN can expose complex and hidden relationships that are difficult to characterise using rule-based programming. Typically, neurons are connected in layers, with signals travelling from the input to the output layer, via at least one hidden layer. Back propagation is the use of forward stimulation to modify the connection weights between the neurons in each layer, and is done to train the network using data labelled with known output values. Increasing the number of hidden layers in a NN leads to better representational power and the ability to model more complex functions, but increases the amount of computation, training data and time required to arrive at accurate models.

TABLE 2.3: Grouping of existing model-based runtime learning techniques for power and energy management

Domain	Reference	DPM	DVFS	Mapping
Embedded/Desktop	[110, 111]		✓	
	[106, 100, 112]			✓
	[113, 114]	✓	✓	
	[115]	✓		✓
	[10, 98, 105, 92]		✓	✓
Data Centers/HPC	[92, 116]		✓	✓
	[103]	✓	✓	
	[95, 117, 118]	✓		✓

If trained for too long, NNs can become over-fitted and the model will include characteristics of outliers from the sample data, yielding an approximation with excellent accuracy on training examples, yet poor performance on further data from the same distribution. Over-fitting can be prevented by reserving part of the data as a test set to establish an unbiased estimate of the NN's accuracy. However, the model accuracy can be degraded as a result of holding aside training data for error estimation as it reduces the number of samples which may be required for training. Cross validation is a mechanism to overcome this whereby the dataset is divided into m equal-sized folds, with m NN models built instead of just a single. Each NN is trained on $m - 1$ folds and tested on the remaining fold, therefore the test fold for each NN differs from the other models [110].

2.4.4 Model-based learning for Runtime Management

In the context of runtime learning and management, modelling enables prediction of the current and future states of a system. This can include physical quantities, such as power, temperature and energy, or the specific properties of applications, such as performance, latency and accuracy. When applying specific requirements or constraints to these properties, models can be used to predict the system configuration that will minimise power consumption and maximise performance before execution, such as control over parallelism, DVFS and DPM settings.

The model-based learning approaches that have been used in the literature are discussed, divided into three main control methods; task mapping (including parallelism and multi-threading control), DVFS and DPM. Categorisation of existing literature is shown in Table 2.3. Many model-based approaches use multiple control methods in conjunction to achieve power and performance optimisation, with mapping applied first and then DVFS or DPM refinements made afterwards.

2.4.4.1 Task Mapping and Parallelism

Model-based learning is commonly employed to build power and performance models of applications executing on platforms in order to predict the optimal mapping of an application's tasks

to processors, the configuration of these processors and determine the level of parallelism that should be created.

The first approaches for determining the required level of parallelism and task mapping using GLMs were empirically derived using Amdahl's Law [40]. However, this fails to capture inter-thread communication, data-dependence synchronisations and hardware contentions that lead to sub-optimal scaling. Modelling approaches have been developed to characterise these penalties with feedback-driven threading for homogeneous architectures [119] and *Scale-up/Scale-out* for heterogeneous architectures [115]. In the latter, *Scale-out* refers to thread-level parallelism [120] and *Scale-up* refers to the adaptive thread-core mapping enabled by heterogeneous cores. These processes are characterised by two orthogonal functions, which are derived from a combination of empirical modelling and fitting of additional coefficients via linear regression [115]. Similarly, composite models can be constructed with a combination of an empirically-derived component and a GLMs component. These are designed to characterise the system whilst also mitigate the modelling error that arises from unknown factors. In [106], a holistic and resource-agnostic scalability model is developed in order to determine the degree of parallelism to assign to each task. The model is based on predicting speed-up from Amdahl's Law, with consideration given to the aforementioned parallelism penalties, however in addition it employs linear regression to analyse the speed-up properties of a particular resource in order to set the correct level of parallelism.

Coarse-grain mapping of applications to computational resources can be driven by regression-based learning. One approach uses an OLS algorithm to build a model of the energy/performance trade-offs between using different computing resources in a heterogeneous system for a particular task [98]. The task is mapped on a computing resource at runtime based on the minimum energy consumption for a given application performance requirement. Parallelism within each resource is not considered because of partial support for it across the resources of the particular platform.

Conversely, approaches that target High Performance Computing (HPC) and data centre systems consider task-level parallelism an essential component of their predictive models. These approaches considered modelling the system in order to perform Dynamic Concurrency Throttling (DCT) [92] and thread packing [10], processes which are included as part of mapping. Curtis-Maury et al. consider an IPC-based linear regression solution trained from samples of the power-performance adaptation search space of real workloads. They derive a performance prediction model which dynamically adjusts DCT, DVFS, and thread placement at the granularity of program phases.

In order to more accurately and generically predict performance improvements for changes in mapping, GLMs can leverage Performance-monitoring Counters (PMCs) built into the hardware architecture [100, 110]. This approach is portable across many applications as it only relies on information from the hardware. Furthermore, metrics such as Instructions per Cycle (IPC) and processor utilisation can be used to predict performance and build linear models across many platforms [92]. Pack & Cap is an example of a model-based approach to control mapping which relies on PMC data [10]. Furthermore, it is different to other approaches in that it employs a multinomial logistic regression classifier to make optimal DVFS and thread packing control decisions in order to maximise performance within a power budget. The addition of thread packing to DVFS as a control knob increases the range of feasible power constraints by an

average of 21.0% when compared to DVFS alone and reduces workload energy consumption by an average of 51.6%.

SVC models can also be found in runtime management schemes and are used to classify tasks or programs as suitable for particular functional units in a heterogeneous architecture. Wen et al. [112] develop an OpenCL task scheduling scheme to map kernels from multiple programs on CPU/GPU heterogeneous platforms. At runtime, it determines which kernels are likely to best utilise a device from a performance model that predicts a kernel's speedup based on its static code structure. Naive Bayes has also been used in power management to build power-performance model and perform classification [113]. In this case, the goal is to devise a power management policy for issuing DVFS commands on a CMP system that minimises the total energy dissipation based on the load conditions and workload characteristics [114]. The motivation for utilising a Bayesian classifier is to reduce the overhead of the power management activities, which are performed regularly to determine and assign DVFS settings for each processor core in the system.

The use of NNs for modelling and prediction in runtime management system is not common, given the extensive training time and large volume of data that is required to achieve an accurate model, as discussed in Section 2.4.3.4. However, NNs can have a role to play in the static components of a hierarchical system such as modelling the behaviour of applications. In [110], a resource allocation scheme is created composed of per-application NN performance models and a global resource manager. Shared system resources are periodically redistributed between applications at fixed decision-making intervals. Each application model's its performance as a function of its allocated resources and recent behaviour, using an ensemble of NNs to learn an approximation of this function. Past program behaviour and allocated resource amounts are presented at the input units, and performance predictions are obtained from the output units [110]. The drawback of these models is the training of the NN weights, which can only be done by performing successive passes over training examples.

2.4.4.2 DVFS in Runtime Management

DVFS is used to control the performance/throughput of tasks by adjusting the operating frequency of the processor. Dynamic power dissipation is reduced when decreasing the voltage and frequency and energy can be saved if further voltage scaling occurs. The power and performance relationship between processors and tasks is often modelled to enable prediction of the optimal DVFS settings. A basic model can be built from understanding of the underlying physical characteristics of the static and dynamic power dissipation of components and how these are affected by frequency and voltage, or empirically from experimentation using training samples. The later may be done with the aid of hardware performance statistics such as IPC [92] or PMCs [10, 105], and approaches such as multivariate linear regression to estimate specific coefficients for the hardware event rates of a particular configuration in order to determine the required DCT and DVFS settings. The *Pack & Cap* [10] approach makes use of L_1 -regularisation to select the most relevant PMC metrics automatically and a multinomial logistic regression classifier to determine the DCT and DVFS settings that maximise performance under a power constraint by selecting the output with the highest probability.

Yang et al. use hypotheses about the affect of frequency and voltage on the current and latency for each resource in a heterogeneous platform as the basis for their power/performance model,

which is used to determine the most energy-efficient resource to execute on and the DVFS settings to apply for a given performance requirement [98]. This model is trained using a OLS linear regression technique at the beginning of application execution. Although it can be done a runtime, it does not change over the remainder of the application so it cannot adapt to changes in application behaviour. In a similar way, Juan et al. [111] use constrained-posynomial (positive polynomial) functions to learn the relationship between performance and power and build a model based on frequency and utilisation. Additional energy reduction is achieved through additional DPM techniques including turbo-mode and near-threshold operation in what they call extended-range DVFS.

A Bayesian classification approach to DVFS setting is used by Jung et al. [113, 114] in the prediction of power and performance. The predicted state is used to look up the optimal power management action from a pre-computed policy lookup table. The motivation for using this form of model is the reduced overhead of prediction in the *power manager*, which can provide energy savings for even rapidly and widely varying workloads.

DVFS is the primary mechanism used by the runtime management approach presented in Chapter 3 to control the power consumption and throughput of the platform. Like many of the approaches presented in Table 2.2, in this work DVFS is twinned with the control of thread-level parallelism in the case study presented in Chapter 4 in order to provide greater scaling of power and performance than can be achieved with DVFS alone.

2.4.4.3 DPM in Runtime Management

DPM process are often driven by predictions from models in conjunction with mapping or DVFS actions. DPM can be achieved through migration in HMP to resources with different power/performance operating points [115, 98] or by power gating CPU cores in homogeneous multi/many-core systems [110, 10]. These two processes are captured as Scale-up and Scale-out by Ma et al. [115] who perform DPM and mapping on a heterogeneous architecture based on predictions from GLM performance and power models with additional heuristic scheduling. Cochran et al. propose a similar approach for performance optimisation under a power budget with PARSEC benchmarks on a homogeneous CMP through a combination of thread packing to control parallelism and DVFS setting to reduce power [10].

2.4.5 Comparison of Model-based learning Approaches

Figure 2.12 shows that energy savings can be achieved by combining DVFS and adaptive thread mapping on a homogeneous CMP with the aid of a multinomial logistic regression classifier [10]. The experiments are conducted across a series of PARSEC benchmarks. In the first experiment, the thread packing technique is used to dynamically adjust the number of threads to meet a changing power budget, with DVFS settings applied on top. The second and third experiments use a fixed one and two threads respectively and so must rely on predicted DVFS settings alone. For the majority of cases, thread packing increases the range of achievable power constraints over DVFS alone. Given that DPM techniques have not been applied to shutdown cores, the one-fixed-thread case consumes a lot more energy due to the static power consumption of the

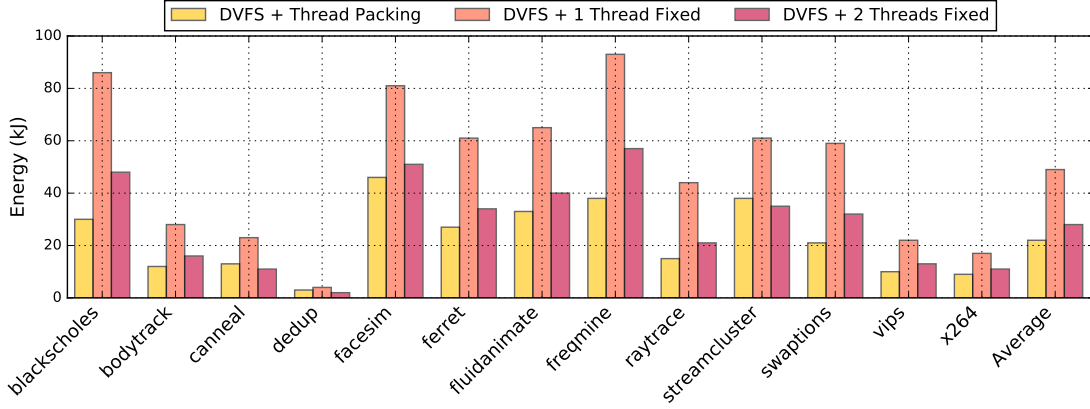


FIGURE 2.12: Comparison of energy consumption for benchmarks with and without thread-packing on a homogeneous CMP, reprinted from [10].

idle cores. The thread-packing experiment is able to utilise all four cores when the power budget allows and the performance gain from this outweighs the increase in dynamic power.

In addition, Yang *et al.* [98] show that learning the power/performance trade-offs for each processing resource of a heterogeneous system enables prediction of the optimal mapping and DVFS settings. A linear regression model is build from training data and used to predict the power and performance of an edge detection filter for particular mapping and DVFS settings on the heterogeneous CPU-DSP-FPGA platform. The runtime management system implemented will switch the mapping of the filter between resources if the performance requirement changes, to ensure optimal energy consumption per frame. The approach presented in Chapter 3 is based on the same modelling and optimisation method as Yang *et al.* but it is extended to operate with multi-core platforms and multi-threaded applications. This new approach models the effect on power and performance of core scaling in addition to DVFS.

2.5 Runtime Management Frameworks

A drawback with existing runtime management algorithms is that they are usually designed only for specific classes of application, such as multimedia [121, 122], web [95] or image processing [98]. In addition, the algorithms may have only been validated with specific applications or benchmarks, such as PARSEC [115, 123], SPLASH-2 [104] or SPEC [103], which can lead to over-optimistic results that are, in general, not transferable to new applications. Similarly, the runtime algorithm may only be designed to manage specific types of architectural configurations, *e.g.* homogeneous multiprocessors [94, 123], data centres [124, 95] or embedded systems [96, 89], or the experimentation has been conducted only on specific hardware platforms [103, 112]. Together, these factors limit the potential of these approaches. The provision of an application- and platform-agnostic framework for runtime management algorithms will alleviate many barriers to the wider development and testing of these runtime algorithms.

The runtime management of applications can be extended by the exposure and adaptation of tunable parameters through a defined framework interface. The concept of enhancing static applications with dynamic knobs has lead to methodologies such as PowerDial [11], ARGO [12],

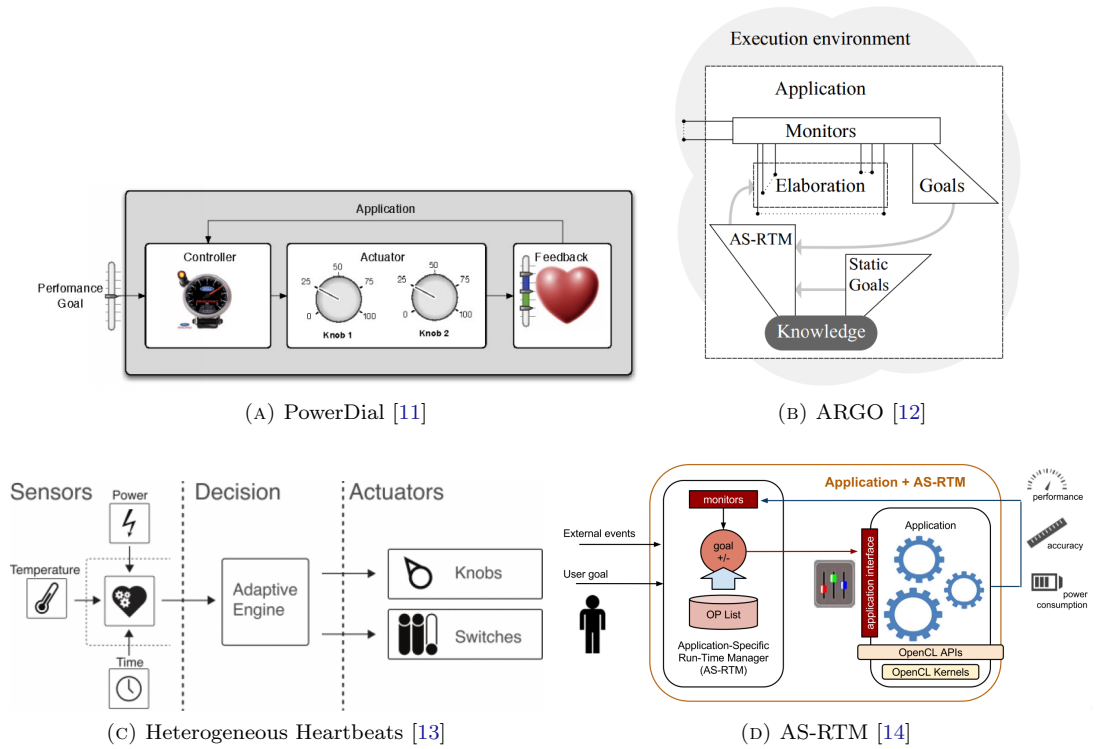


FIGURE 2.13: Conceptual overviews of frameworks proposed in literature, reprinted from [11, 12, 13, 14].

Heterogeneous Heartbeats [13] and AS-RTM [14]. The conceptual overviews for each of these frameworks are shown in Figure 2.13. A commonality between all of these approaches is that they capture and synthesise system properties of interest into monitors or sensors. This data drives some form of decision algorithm or feedback controller, with reference to a target or goal. The last stage, shown in all the approaches, is the actuation of a set of controllable parameters, through a standard interface or API, to modify the external system.

The dynamic adaptation of application knobs has been used for throughput-power trade-offs [125] in addition to precision-throughput trade-offs [126]. Furthermore, exploration of the platform operating space has been used to locate Pareto-optimal points for tunable applications [127]. Formalisation of the monitoring of performance properties can be seen in the Application Heartbeats API [128], which allows information on the behaviour of applications to be conveyed to the RTM. The heartbeat concept has also been used to explore reliability-performance trade-offs [129] and for task management on heterogeneous systems [130]. Additionally, methods of exposing device-level metrics in a standardised way have been demonstrated [131, 132].

These existing framework-based approaches can be broadly placed into three overlapping classifications: those that only abstract runtime algorithm-application [12, 13, 14, 129] or algorithm-device [131, 132] interactions; those that only expose monitors [13, 14, 130, 131, 132]; and those that are tightly coupled to a particular platform, device type or use device-specific runtime management algorithms [11, 13, 14, 130, 132]. Additionally, frameworks based on the concept of heartbeats are limited to applications that express their performance in terms of time and cannot report accuracy or error rates [128, 129, 130].

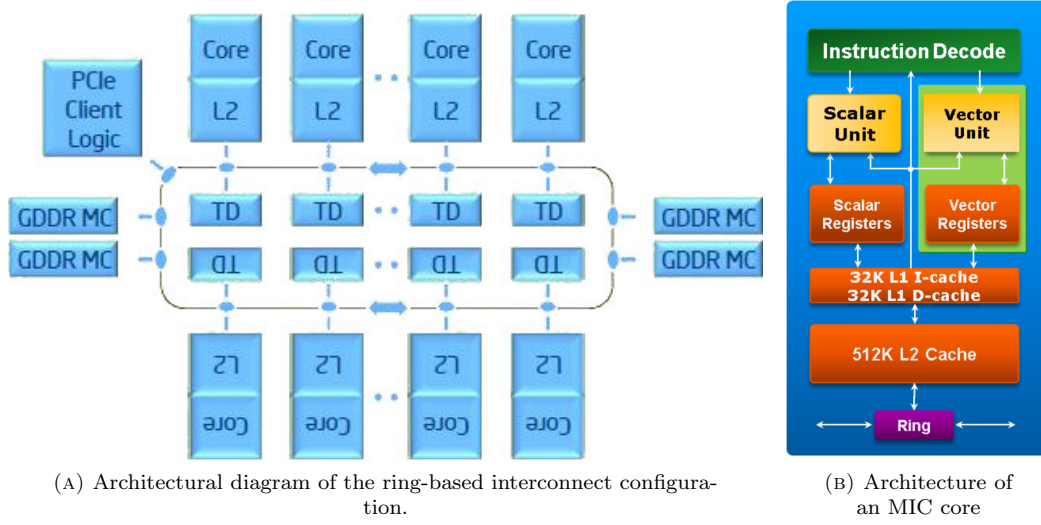


FIGURE 2.14: Intel Xeon Phi diagrams, adapted from [15] and reprinted from [16].

Chapter 5 goes beyond these works to present a novel framework for runtime management which is complete in that it supports both the tuning of application and platform parameters. In addition, it is extensible so that it can support the optimisation of any metric or property through a completely generic interface. Optimisation and control are done in an entirely agnostic manner, increasing the modularity of applications, platforms and runtime management algorithms.

2.6 Experimental Platforms

The hardware platforms used to conduct the experiments described in the following chapters of the thesis are presented in this section. The entire platform is presented, however particular focus is given to the SoC and the processing resources contained within it, as management and control of these parts is the focus for the work. The terms platform and device are used interchangeably throughout the thesis to refer to the SoC and these terms are intended to encapsulate the processing resources and the memory subsystem.

2.6.1 Intel Xeon Phi 7120P

The Intel Xeon Phi™ is an SMP platform based on the Intel Many Integrated Core (MIC) architecture containing up to 61 processor cores. The L2 caches of all the cores are interconnected via a bidirectional ring bus, creating a shared last-level cache of up to 32MB, as illustrated in Figure 2.14a. This figure also shows the interface to the GDDR5 main memory and the global-distributed tag directories which ensure cache coherency. Figure 2.14b shows the microarchitectural blocks of each MIC core, including the separate scalar and vector processing units and the ring bus connection to the L2 cache.

The platform uses the same voltage-frequency island for all the cores, which are scaled together in the 9 steps shown in Table 2.4. The relationship between the supply voltage v and operation

TABLE 2.4: Voltage and frequency scaling steps of the Xeon Phi processors.

Parameter	Step								
	1	2	3	4	5	6	7	8	9
Voltage (V)	0.995	1.01	1.02	1.025	1.035	1.04	1.05	1.055	1.06
Frequency (MHz)	619	666	714	762	857	952	1048	1143	1238

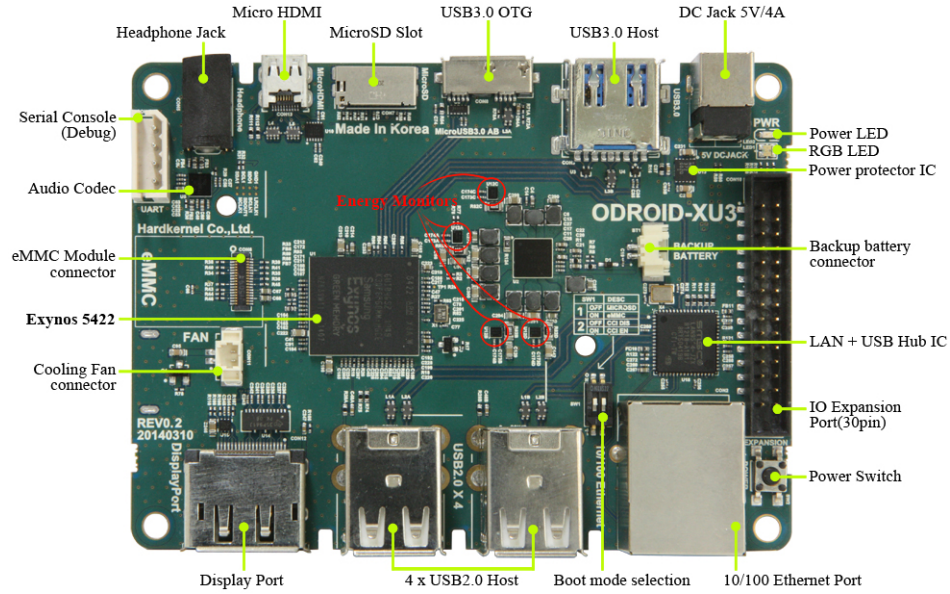


FIGURE 2.15: Odroid-XU3 Board Image, reprinted from [17]

frequency f can be approximated as $v = 0.93 + 0.11f$ where f is in GHz. This equation is used in the modelling presented in Chapter 3 to translate the predicted current into predicted power.

Multiple approaches can be taken to measuring power consumption on the Xeon Phi [133]. To collect the power data used in the experiments presented in Chapter 4, the *micras* filesystem (sysfs) nodes are access directly from within the RTM. This methods provides a breakdown of the power consumption of components within the Xeon Phi SoC and can be performed online to track real-time power consumption. The measurements included are the core, non-core and memory subsystem power readings, in order to capture the consumption of the entire architecture. The code to perform power measurement is shown in Appendix C.

The Xeon Phi platform is used to demonstrate the scaling of power and performance of multi-threaded applications on multi-core platforms by utilising high core numbers. This is in expectation of next-generation commercial mobile and embedded platforms, which trends show may feature architectures with these high core counts in the near future. Chapter 4 illustrates this process in the context of the stereo matching application, which is threaded to take advantage of the high parallelism inherent in the Xeon Phi MIC architecture.

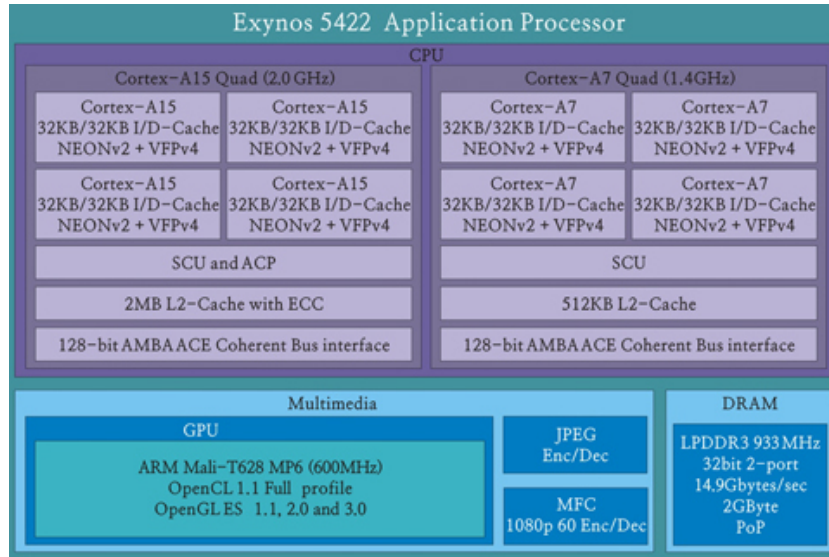


FIGURE 2.16: Odroid-XU3 system block diagram, adapted from [17]

2.6.2 Odroid-XU3

The Odroid-XU3 is an embedded development platform, shown in Figure 2.15, built around the Samsung Exynos 5422 SoC. This SoC contains a HMP architecture with four ARM Cortex-A7 and four Cortex-A15 CPUs arranged in two performance-asymmetric quad-core clusters, as shown in Figure 2.16. In addition, the SoC contains a Mali T624 Graphics Processing Unit (GPU), with six shader cores, and 2GB of LPDDR3 main memory. The presence of multiple heterogeneous processing elements makes this an ideal platform for experimentation with runtime management techniques. Four hardware power measurement sensors are embedded on the board, which measure real-time power consumption for the A7 and A15 clusters, the memory controller and the GPU. In addition, each A15 core and the GPU has a temperature sensor. These are used to assess the power savings achieved by each RTM in the experiments presented in Chapter 5. The processors also support the use of hardware performance counters to measure the activity of various architectural components without interrupting the operation of the system. These are accessed by one of the RTM approaches presented in Chapter 5. Furthermore, the Odroid-XU3 is used in Chapter 5 to perform profiling of applications through the framework and it is used by the runtime management approaches that are validated within the framework.

2.6.3 Cyclone V SoC Development Board

The Altera Cyclone V SoC features two ARM Cortex-A9 CPUs and a Field-Programmable Gate Array (FPGA) fabric with 42k look-up tables, which are tightly coupled within the same chip. The two power regulators present on the development board facilitate the independent control of seven voltage rails, including CPU and FPGA core voltages, along with the monitoring of their real-time power consumption. Open Compute Language (OpenCL) applications can be executed on the FPGA fabric by despatching kernel tasks to pre-compiled accelerators. Applications for the Cyclone V were developed in OpenCL and compiled with the vendor's tool chain. Host

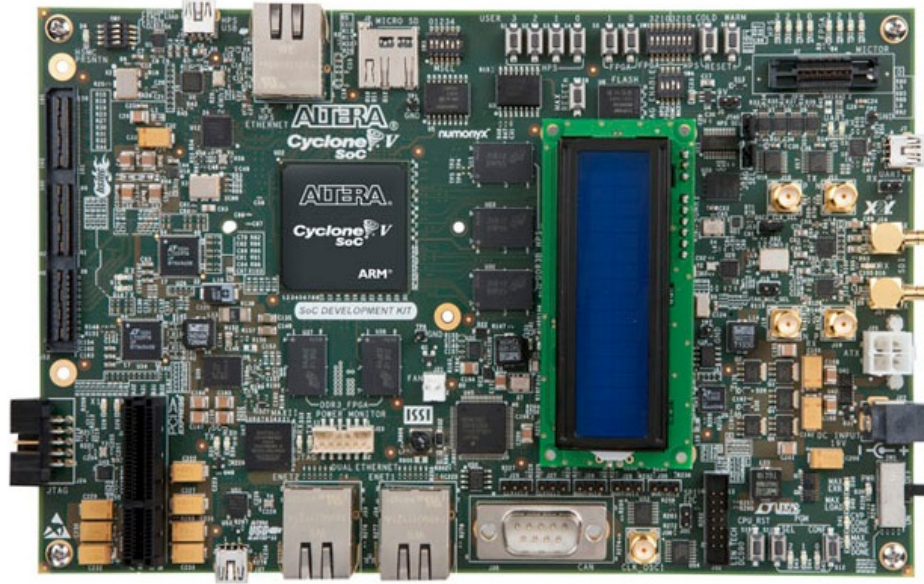


FIGURE 2.17: Altera Cyclone V SoC Development Board.

code was executed on the Cortex-A9 CPUs under Ubuntu 16.04, with kernel tasks dispatched to accelerators residing on the FPGA fabric.

The Altera Cyclone V is used for the experiments described in Chapter 5 for profiling of applications through the framework. The platform is used to demonstrate that the same application code can be executed across multiple platforms when managed through the framework.

2.7 Summary

Developments in processor circuit- and architectural-level technologies help to address the performance demands and power consumption constraints of future embedded devices. HMP architectures and asymmetric processing resources provide a range of operating modes for a system, and allow performance-power trade-offs to suit application demands or power budgets. However, runtime software and operating systems must evolve in order to efficiently manage multi-core and HMP processor architectures. Most critically, control of the application configuration to match platform resources is an important consideration, to avoid conflicts between applications using shared resources that may lead to performance bottlenecks. Moreover, programming frameworks, such as OpenCL, help software developers to more efficiently utilise parallel and heterogeneous processors and increase the portability of code.

Two complimentary taxonomies for application characterisation have been presented, suitable for the next generation of complex and demanding workloads. These characterisation methods cover a diverse range of application domains. Using the characterisation and convergence principles, these taxonomies represent many applications within a small collection of motifs.

Furthermore, this chapter has presented a review of current trends in runtime management algorithms for the optimisation of power and performance across a range of applications, benchmarks and platforms. This is an established area of research, with many approaches specialised to particular domains of application or platforms and leveraging a range of properties across both. Section 2.4 has shown that runtime management represents an essential tool in enabling the optimisation of multi-core and heterogeneous systems, and creating trade-offs between computational quality, application throughput, system reliability and energy efficiency during execution. A range of runtime control techniques have been presented such as DVFS, scheduling and concurrency throttling. Particular focus has been given to model-based learning approaches, in order to establish what research exists and whether these techniques are implemented in runtime energy management solutions.

Looking towards extending the applicability of runtime management approaches, Section 2.5 has shown a review of runtime management frameworks, which aim to enable aspects of application-independence through a standard API or allow runtime algorithms to work across multiple platforms by abstracting device-specific information. A review and classification of published approaches is given to examine where contributions can be made.

Lastly, a summary of the properties and architectures of the hardware platforms used for experimentation in this thesis are presented. To this end, the following chapter presents a runtime energy management approach that aims to optimise the execution of a multi-threaded application on a multi-core platform through predictive modelling and control at runtime. This approach is intended to demonstrate that, by modelling power and performance at runtime, management software can more accurately capture and adjust the system to fluctuations in the behaviour of applications and the constraints of platforms.

Chapter 3

Linear regression modelling for runtime power and performance optimisation

The literature review has shown that both platforms and applications are increasing in their complexity and heterogeneity. Heterogeneous multiprocessor systems incorporate many energy saving technologies, from the circuit level to the architectural level, which are predominantly focussed on reducing leakage power. These technologies are backed by DPM software processes in an attempt to deliver higher energy savings, by directly controlling the hardware, to reduce dynamic power consumption and change the power and throughput characteristics.

Conversely, applications have become more complex, in order to continue delivering higher performance, with very little consideration given to energy efficiency. Instead, applications achieve higher energy efficiency by exploiting multi-threading or heterogeneity in hardware architectures and must now be controlled to ensure this exploitation occurs efficiently. Furthermore, highly dynamic environments are often presented within mobile and embedded systems, both from external stimuli and from the software workload, resulting in fluctuating application demands or platform constraints.

Management software is required, in addition to the OS, in order to ensure that platforms and applications combine and operate in the most efficient manner. This software should have access to parameters in both the hardware and software so that it can control their behaviour. Moreover, energy management at runtime can further increase the efficiency of application execution and extend the duration over which the battery of a mobile or embedded device will last. Model-building at runtime ensures greater adaptability and allows a model to be periodically updated to capture changes in application behaviour or platform properties.

However, many existing RTM approaches based on Machine Learning (ML) are not suitable for runtime modelling due to the fact that they require a large dataset and a long training period, such as neural networks. In addition, many are also not easy to retrain and can only achieve adaptability, or widen their capabilities, by increasing their complexity. To overcome this

constraint, a Multiple Linear Regression (MLR) model-based management approach, applicable to multi-core systems, is presented in this chapter, which can perform power and performance optimisation at runtime. The effectiveness of the model is validated empirically and a gradient-descent optimisation approach is proposed to utilise the model at runtime and select optimal operating settings.

To summarise, the chapter's contributions are:

1. The development of a modelling approach based on MLR that can be trained at runtime to predict the power and performance of a multi-core system and multi-threaded application.
2. Validation of the accuracy and stability of this model through analysis of the number of training samples required and the statistical relevance of the model coefficients.
3. Connection of the models with a runtime optimisation method based on gradient descent to optimise platforms and application settings under a performance target.

Material from this chapter has also been published in ACM TECS as Leech *et al.* [25]. Section 3.1 and 3.2 begin the chapter by presenting a derivation of the linear modelling approach before establishing it in the context of power and performance modelling for a multi-core system. Section 3.3 continues by conducting validation of the models through assessment of the mean absolute error and analysis of the statistical rigour of the modelling coefficients. Section 3.4 presents a runtime optimisation approach utilising the predictive models based on gradient descent. Finally, Section 3.5 gives a breakdown of the overheads before summarising.

3.1 Linear Modelling

MLR is a supervised machine learning approach that is used to establish a relationship between the dependent variables of a system, which can be observed, and a set of associated independent predictor variables, which can be controlled [134]. It is an extension of standard linear regression where only a single predictor variable is used. Linear regression and MLR are used for predicting a quantitative response from a set of inputs. The relationship is defined by a hypothesis function as:

$$h_{\theta}(x) = \alpha + \sum_{i=1}^n \theta_i x_i + \epsilon \quad (3.1)$$

where α is the intercept, x_i are the independent predictor variables, n is the number of predictors, θ_i are the fitting coefficients and ϵ is the error. The values of θ_i are calculated to minimise the mean-squared prediction error ($J(\theta)$) of the hypothesis, which is given by:

$$J(\theta) = \frac{1}{m} \sum_{j=1}^m (h_{\theta}(x^{(j)}) - y^{(j)})^2 = \frac{1}{m} (\Theta^T X - \vec{y})^T (\Theta^T X - \vec{y}) \quad (3.2)$$

where m is number of training samples, $x^{(j)}$ is the j^{th} vector of predictor variables, $y^{(j)}$ is the j^{th} measured value and $h_{\theta}(x^{(j)})$ is the model prediction for the j^{th} vector of predictors. $J(\theta)$ is minimum when its gradient becomes 0. Hence, from Equation 3.2 the gradient of $J(\theta)$ can be defined as:

$$\nabla J(\theta) = \nabla (\Theta^T X - \vec{y})^T (\Theta^T X - \vec{y}) = X^T X \Theta - X^T \vec{y}$$

TABLE 3.1: MLR modelling hypotheses

Model	Hypothesis $h_\theta(x)$
Latency (τ)	$\theta_0 + \theta_1 \frac{1}{f} + \theta_2 \frac{1}{fc} + \theta_3 c$
Current (i)	$\theta_0 + \theta_1 v + \theta_2 v f c$
Performance	$F_{\text{perf}}(f) = 1/\tau$
Power	$F_{\text{pow}}(f) = v i$

From Equation 3.3, the fitting coefficients Θ of the hypothesis in Equation 3.1 can then be computed as:

$$\Theta = \frac{X^T \vec{y}}{X^T X} \quad (3.3)$$

The residual error (ϵ), which gives the difference between the true value and the model prediction, is defined as:

$$\epsilon = \vec{y} - X\Theta \quad (3.4)$$

From Equation 3.3, the computation complexity of the regression-based modelling is $O(n^2 \times m)$, where n is the number of predictors and m is the number of learning samples. Hence, to achieve a fast runtime model both n and m need to be small.

3.2 Runtime Power and Performance Model

A runtime model simulates the properties of the application and hardware platform in a continuous system. Statically-generated models can provide better accuracy, but involve extensive offline profiling of individual applications and platforms. A runtime model enables flexibility in terms of the application-platform configuration, which can be learnt and adapted without the overhead associated with offline profiling. Such a runtime model enables the prediction of power-performance trade-offs under different operating conditions. Through careful design of the runtime models, high accuracy can be achieved with low overhead, therefore a runtime model is a critical component for energy-efficient adaptation. This section outlines the theoretical basis for the power and performance models, demonstrates the model-building process and reports the statistical properties of the model.

In this work, two predictors are used: number of cores (c) and frequency (f), together with the intercept; hence, $n=3$. The model is learnt using runtime measurement of application performance and power sensor data. Performance and power are not linear functions of frequency and the number of cores, therefore models for output current (i) and latency (τ) must first be generated, then performance and power models are built from these. Table 3.1 shows the hypotheses used to generate the models. Column 1 shows the target model and column 2 shows the hypothesis used. These models and their hypotheses are explained further as follows:

- Latency (τ) is expressed as a sum of four terms: the first term is a constant (θ_0) meaning the delay contributed by factors independent of multi-threading and frequency (such as memory contention, I/O setup, etc); the second term ($\theta_1 \frac{1}{f}$) is proportional to the CPU clock period, representing the time spend by the sequential part of the application; the

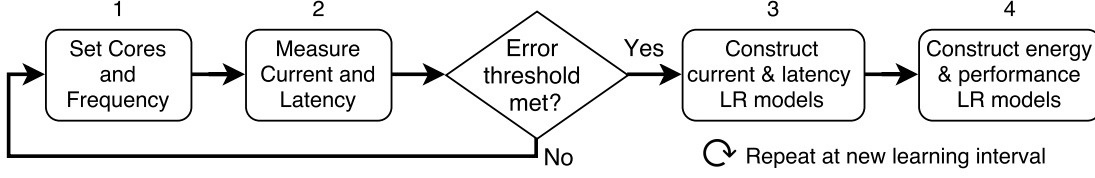


FIGURE 3.1: Flowchart of the runtime power and performance model generation process.

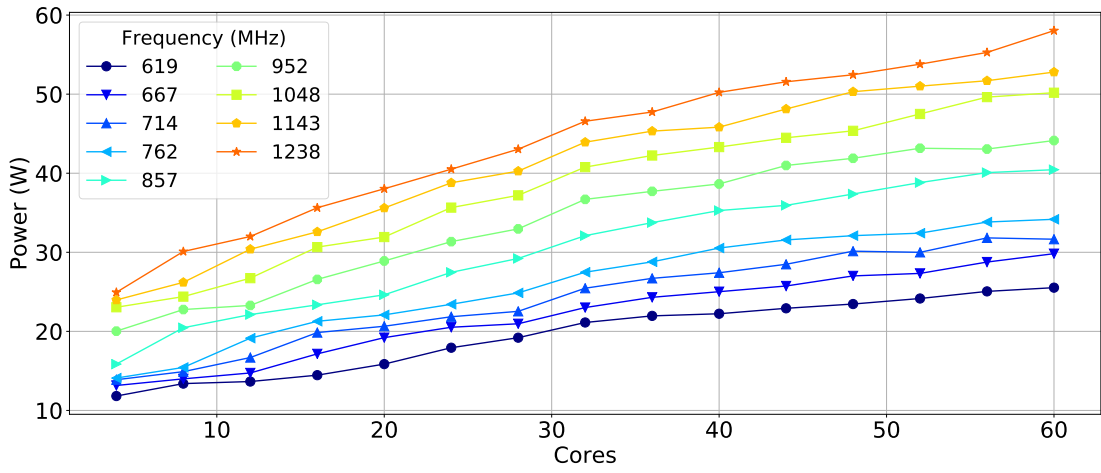
third term ($\theta_2 \frac{1}{f_c}$) is proportional to both the clock period and number of cores, representing the time spent by the parallel part of the application; and the last term ($\theta_3 c$) characterises the latency related to the effects of multi-threading.

- Current (i) is expressed as a sum of three terms: the two terms ($\theta_0 + \theta_1 v$) approximate the leakage current, while the last term ($\theta_2 v f c$) signifies the dynamic current.
- Performance, in units of Frames Per Second (FPS), is expressed as inversely proportional to the latency.
- Power consumption is expressed as a product of the instantaneous current (i) and the supply voltage (v). The supply voltage is derived as a direct function of the operating frequency, which is fixed by the power management controller based on the selected frequency.

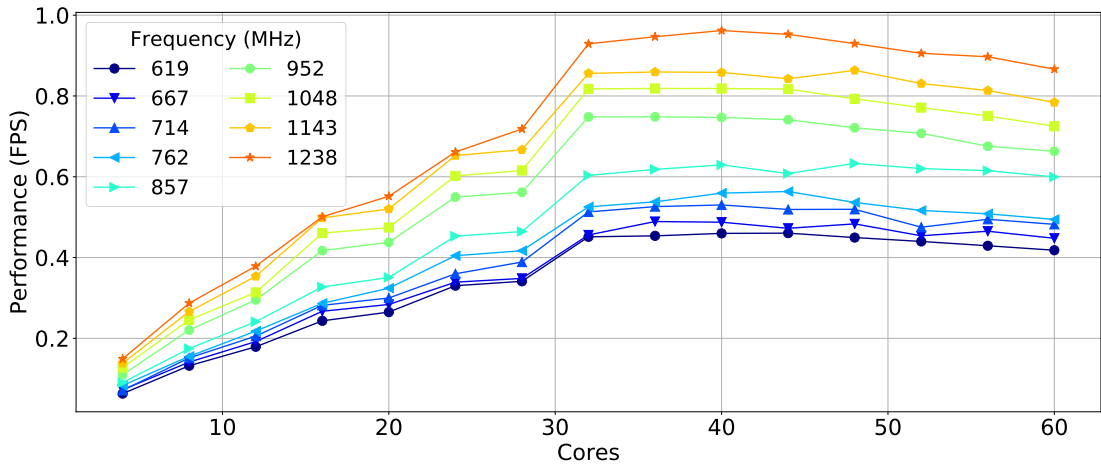
Figure 3.1 shows how the model is learnt in the context of a multi-core platform, using MLR in 4 steps: The modelling starts by varying the operating frequencies and number of active cores (step 1). For every frame, current and latency measurements are captured from the power sensors and the application (step 2). The measurements are used to model the hypotheses until the learning interval is complete. After this interval, current and latency models are generated for the given application running on the platform (step 3). These models are combined to derive the power and performance models (step 4).

Figure 3.2a, 3.2b and 3.2c show scatter plots of the measured power, performance and energy data of the stereo matching application, presented in Chapter 4, executing on the Xeon Phi platform under different operating conditions. For this characterisation, the number of cores used by the application is swept in increments of 4 between 4 and 60 for each frequency level. During runtime, the number of cores used by the application can be specified at the single core granularity. Energy is calculated as the product of the average power and application latency ($1/\text{performance}$). The R script used to generate these plots can be found in Listing A.4 of Appendix A. A significant linear relationship can be observed from both the frequency and the number of cores to the power consumption in Figure 3.2a. For performance, Figure 3.2b shows that while there is a strong linearity with frequency, there is a limit to the linear scaling of performance with the number of cores. Above 32 cores, performance does not increase with increasing core numbers. This non-linearity impacts the accuracy of the model as it the regression process attempts to fit a linear equation to the data.

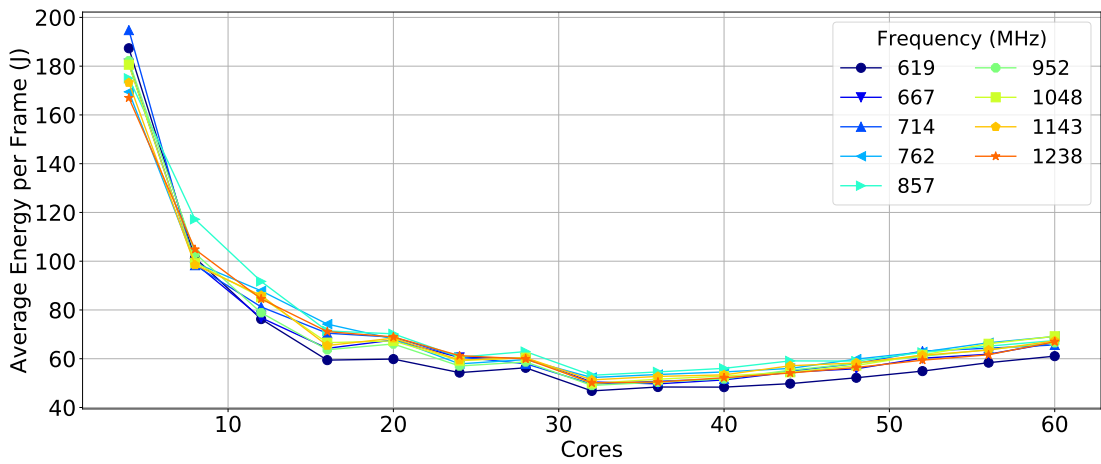
In Figure 3.2c, the energy consumption per frame decreases as more cores are used, because performance scales faster than power consumption, until approximately 32 cores and thereafter there are no energy benefits from core scaling. The energy consumption for each frequency level



(A) Measured power



(B) Measured performance



(c) Measured energy

FIGURE 3.2: Plots of measured power (a), performance (b) and energy (c), for the stereo matching application executing on the Xeon Phi, across a range of frequencies and number of cores. Each point is an average of four measurements. Data is listed in Appendix B

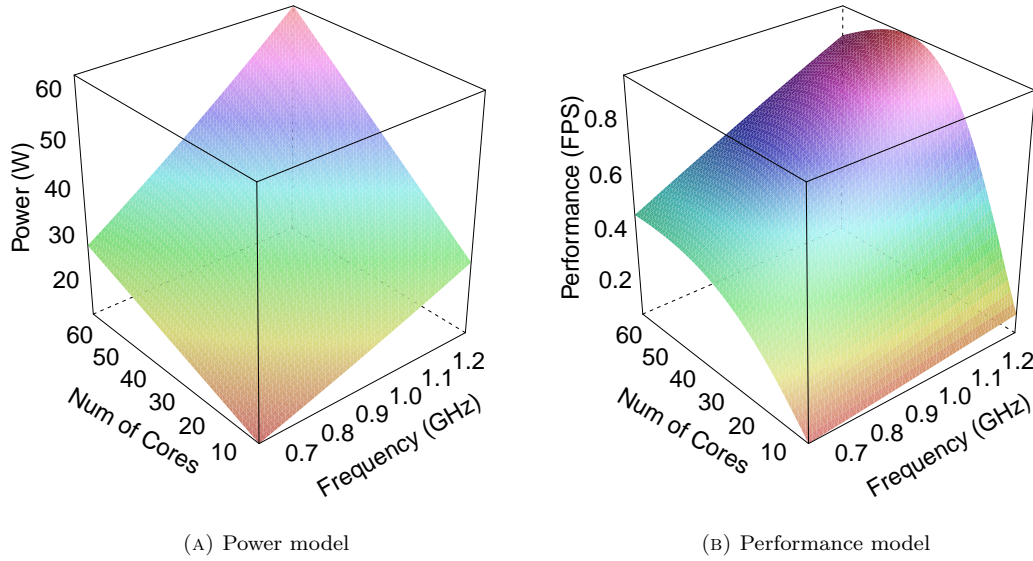


FIGURE 3.3: Runtime models for power (a) and performance (b) generated from training data.

are closely clustered for all the core numbers. This is due to the fact that there is equivalent scaling in power as there is in performance when the frequency is adjusted. For example, at 40 cores and 619 MHz the power consumption is 22.2 W and the performance is 0.460 FPS. Increasing the frequency to 1048 MHz scales power consumption by 1.95x to 43.3 W and scales performance by 1.78x.

A subset of the measured data is used as training samples to generate the power and performance models of Figure 3.3a and 3.3b, following the hypotheses of Table 3.1. In practice, when the RTM and application are running on the platform, as illustrated in Figure 3.1, the only available data is that which is collected at runtime as the application executes. The models predict power and performance across the full range of operating conditions, extrapolating from the training data provided to a continuous function that covers the entire operating surface. During application execution, the runtime models are interrogated every time the application requirements or system constraints change in order to determine predicted power and performance for new optimal operating points. The shape of the 3D models has a high correlation to the measured data, with the power model being highly linear and the performance model showing the same relationship between the number of cores and performance; levelling after approximately 32 cores.

Listing 3.1 shows how the models are generated using the *lm* function from R (lines 22- 24), a statistical computation language [135], which takes the hypothesis function and training dataset as its input, returning a linear model object. The complete R script, including code to generate the plots can be found in Listing A.1 of Appendix A.

Validation of the power and performance models is detailed in the following section, where the impact that the number of training samples has on the model prediction accuracy is demonstrated along with the runtime overheads associated with it.

LISTING 3.1: R code to generate the power and performance models of Figure 3.3a and 3.3b.

```

1 #####
  # Read in data
  #####
  dt = data.table(read.csv('data_lrmtm.txt', header=TRUE, sep=" "))
5 dt = dt[,.(perf=mean(perf), energy=mean(energy)), by=.(freq,cores)] #avg repeat exps

  freq_list = c(619047, 666666, 714285, 761904, 857142, 952380, 1047618, 1142856, 1238094)
  d.fnv = read.table("freq.vdd.csv", header=TRUE, sep=",")
  vdd = d.fnv$vdd[findInterval(dt$freq, freq_list)]
10

  dts = dt[,.(freq, cores, perf, lat=1/perf, power=energy*perf, energy, vdd, vfc=vdd*freq*
    cores, period=1/freq, ppc=1/(freq*cores), current=(energy*perf)/vdd)]

  #get indicies in dts for data to be used for model training
  freq.sel = c(619047,857142,1142856,1238094)
15 cores.sel = c(4,16,32,60)
  train = dts[,.I[cores %in% cores.sel & freq %in% freq.sel]]

  #####
  # Generated models
20 #####
  model.current = lm(current ~ vdd + vfc, data=dts[train], weight=1/current)
  model.lat = lm(lat ~ period + ppc + cores, data=dts[train], weight=1/lat)
  model.fnv = lm(vdd ~ freq, d=d.fnv)

```

3.3 Runtime Model Validation

This section presents validation of the models outlined in the previous section using standard statistical methods and establishes an error convergence bound. Analysis of the training samples required to achieve a sufficiently accurate model is also presented. The accuracy of the generated models depends on a number of factors; the number of samples acquired, the number of predictors, and the underlying relationships between current, latency, performance and power. Additional validation has been conducted to assess the error convergence properties of the power and performance models. Modelling error is calculated using a comparison between the predicted power and performance values and measured data under the same conditions. The validation data set is exclusive of the training samples used to generate the models. This prevents the model being validated by a dataset that is over-representative of the true distribution.

Figure 3.4a shows the effect of the number of training samples on the mean absolute percentage error of the power and performance models. Figure 3.4b provides further insight into how many training samples are required to achieve convergence by plotting the rate of reduction in error ($\text{rate} = \text{derror}/\text{dsamples}$). Convergence in the model error is defined as when the reduction in power and performance modelling error is less than 0.5% between training samples. Figure 3.4b shows that these models require a small convergence interval of only 15-20 training samples.

Listing 3.2 shows the R code used to generate the data for Figure 3.4a. The training samples are randomly selected from the full characterisation set in line 29. The function `model.err` is used to generate the models, test them against the remaining data and calculate the mean absolute error of the power and latency prediction. The other loop means that this process is repeated 40 times for each number of samples to establish an error distribution, from which the mean is calculated. The rate of change in the error is calculated for Figure 3.4b using `abs(diff(res$perf))` and `abs(diff(res$power))`, which finds the difference between the error of consecutive sample numbers.

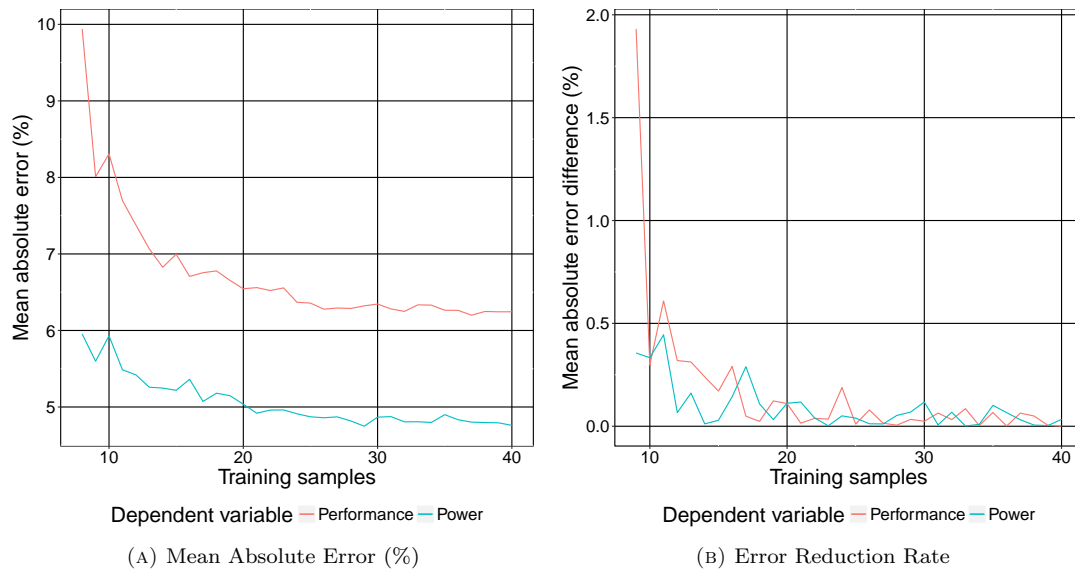


FIGURE 3.4: Effect of the number of training samples on the mean absolute error (A) and the rate of reduction of the error (B) of the power and performance models. Error is the difference between predictions using the models and the remaining experimental characterisation data.

LISTING 3.2: R code to test the number of training samples required for error convergence. The full script can be found in Listing A.2 of Appendix A.

```

1  model.lat.v = lm(lat ~ period + ppc + cores, data=dts[train], weight=1/lat)
   model.current.v = lm(current ~ vdd + vfc, data=dts[train], weight=1/current)
   #model prediction (testing)
   lat.pred = predict(model.lat.v, newdata=dts[test])
5  pwr.pred = predict(model.current.v, newdata=dts[test])*dts[test]$vdd
   eng.pred = predict(model.current.v, newdata=dts[test])*dts[test]$vdd*lat.pred
   #model validation (error calculation)
   lat.err.pc = abs(lat.pred - dts[test]$lat)/dts[test]$lat
   pwr.err.pc = abs(pwr.pred - dts[test]$power)/dts[test]$power
10  eng.err.pc = abs(eng.pred - dts[test]$energy)/dts[test]$energy
   return (c(mean(lat.err.pc), mean(pwr.err.pc)))
}
#####
# Test num samples vs error
#####
15  res = matrix(, nrow=0, ncol=5)
   colnames(res) = c('samples', 'num_freq', 'num_cores', 'perf', 'power')
   num_repeats = 40

20  for(j in c(8:40))
   {
     freq_levels = numeric(num_repeats)
     core_levels = numeric(num_repeats)
     errors = matrix(, nrow=0, ncol=2)
25   colnames(errors) = c('perf', 'power')

     for(i in c(1:num_repeats))
     {
       train_data = dts[,.I[sample(1:nrow(dts),j)]]
       test = -train_data
30       freq_levels[i] = nlevels(factor(dts[train_data]$freq))
       core_levels[i] = nlevels(factor(dts[train_data]$cores))
       errors = rbind(errors, model.err(train_data, test))
     }
35   errors = data.table(errors)
       freq_levels_mean = mean(freq_levels)
       core_levels_mean = mean(core_levels)
       errors_mean = c(mean(errors$perf), mean(errors$power))
       res = rbind(res, c(j, freq_levels_mean, core_levels_mean, errors_mean))
40 }

res = data.table(res)

```

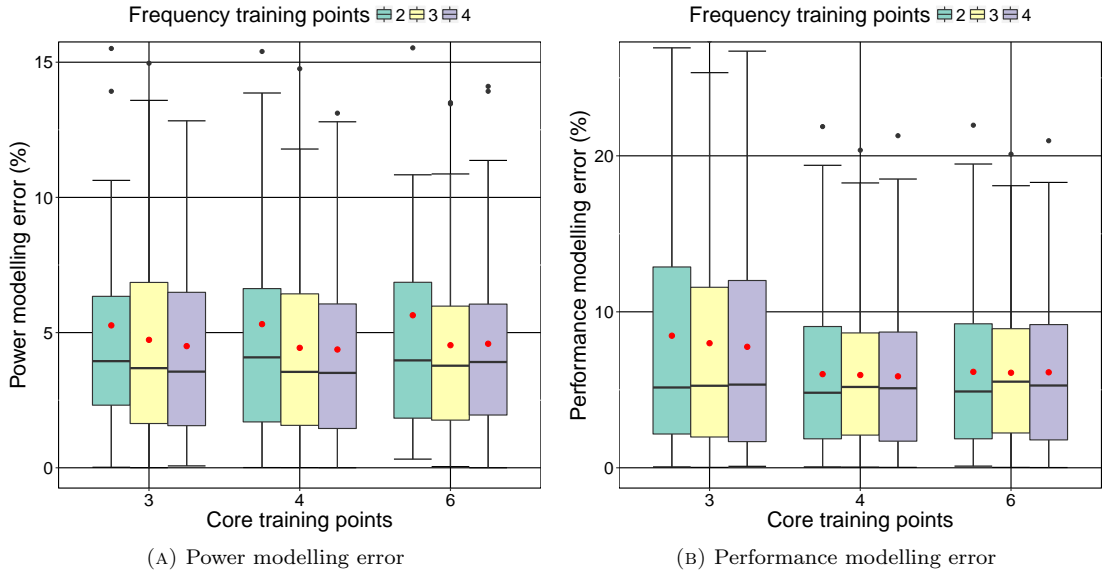


FIGURE 3.5: Increasing the number of training samples for both core count and frequency reduces the power and performance modelling error. Mean error is show in red, boxes extend from the first to third quartile with the median line marked.

An additional dimension worth analysing is the number of different frequency and core points, that are required to ensure convergence in the power and performance modelling errors. The effect of increasing the number of different frequency and core points on the power and performance modelling errors are shown in Figure 3.5a and 3.5b. The x-axis shows the number of core training points and each sub-plot in the group shows the different number of frequency training points. The total number of training points is the product of the core and frequency training points. The data is divided into a training and test dataset, in order to generate the two MLR models and calculate the mean absolute error. This function is called repeatedly with different combinations of training and test data that represent a different number of frequency and core levels. Convergence of the modelling error is significant after four core and four frequency training points (16 samples in total). The mean absolute performance and power model errors at this point are 5.95% and 4.25% respectively. This is because regression with a small number of predictors is rigid and the variance in the model is small [136].

The properties of a model generated in R can be reported from the model object and these are summarised in Table 3.2 and 3.3 for the latency and current models. The fitting coefficients for the model hypothesis are shown in the third block of rows along with their standard errors. These coefficients apply to a model that is valid for the possible range of voltage, frequency and number of cores of the platform. Included here are the p -values for the coefficients in each model, also known as the null hypothesis value. To be statistically significant, the p -value must be lower than 0.05, indicating that there is less than 5% probability that the correlation between the dependent and independent variables occurred by chance [137]. For the latency model, the p -values for θ_0 to θ_3 are $[4.5 \times 10^{-4}, 1.1 \times 10^{-07}, 1.7 \times 10^{-15}, 5.17 \times 10^{-3}]$, therefore they are well below the required threshold. An overall p -value is given for the model as $< 2 \times 10^{-16}$. The same can be said for the statistical significance of the current model, the summary of which is shown in Table 3.3.

TABLE 3.2: Summary of the statistical properties of the latency model.

Hypothesis:	$\text{lat} = \theta_0 + \theta_1 \frac{1}{f} + \theta_2 \frac{1}{f_c} + \theta_3 c$			
Weighted Residuals: (weights = 1/lat)				
Min	1Q	Median	3Q	Max
-0.18548	-0.03557	0.00443	0.05013	0.10722
Coefficient	Estimate	Std. Error	t-value	$\Pr(> t)$
θ_0	-6.54×10^{-1}	1.49×10^{-1}	-4.40	4.5×10^{-4}
θ_1	9.38×10^5	1.04×10^5	9.03	1.1×10^{-7}
θ_2	3.36×10^7	1.12×10^6	29.99	1.7×10^{-15}
θ_3	8.51×10^{-3}	2.63×10^{-3}	3.24	5.17×10^{-3}
Residual standard error:		0.0787	on 16 degrees of freedom	
Multiple R-squared:		0.993		
Adjusted R-squared:		0.992		
p-value:		$< 2 \times 10^{-16}$		

TABLE 3.3: Summary of the statistical properties of the current model.

Hypothesis:	current = $\theta_0 + \theta_1 v + \theta_2 v f c$			
Weighted Residuals: (weights = 1/current)				
Min	1Q	Median	3Q	Max
-0.3995	-0.1430	-0.0209	0.1933	0.3176
Coefficient	Estimate	Std. Error	t-value	Pr(> t)
θ_0	-1.83×10^2	1.41×10^1	-13.0	3.0×10^{-10}
θ_1	1.94×10^2	1.38×10^1	14.0	9.1×10^{-11}
θ_2	4.15×10^{-7}	1.51×10^{-8}	27.4	1.6×10^{-15}
Residual standard error:		0.226	on 17 degrees of freedom	
Multiple R-squared:		0.989		
Adjusted R-squared:		0.988		
p-value:		$< 2 \times 10^{-16}$		

The value of the fitting coefficients for each model is specific to the experimental platform used. For the current model, these are determined by the exact leakage and dynamic currents that occur, which is dependent on the process technology used to manufacture the physical hardware. For the latency model, the coefficients are determined by the support for parallelism in the platform architecture. This includes how well multi-threading is supported and at what point scaling becomes sub-linear. Heterogeneous architectures in particular, with performance-asymmetric processor cores, will lead to performance scaling which varies depending on the latency effect of the next core to be used.

In the next section of each table, the residual standard error measures the average amount that the average value of the latency or current data will vary from the predicted variable (*i.e.* the regression line). The R-squared terms are close to 1, which indicates that the variance in the latency and current are strongly captured by the model. The final stage of validation is to perform an Analysis of Variance (ANOVA) on each model. Figure 3.6 and 3.7 show the results of this analysis in four plots including; residuals versus fitted values, a Q-Q plot of standardised residuals, a scale-location plot (square roots of standardised residuals versus fitted values), and a plot of residuals versus leverage that adds bands corresponding to Cook's distances of 0.5 and 1 [138].

The top left plot in each set shows the residuals of the models plotted against the fitted values. It

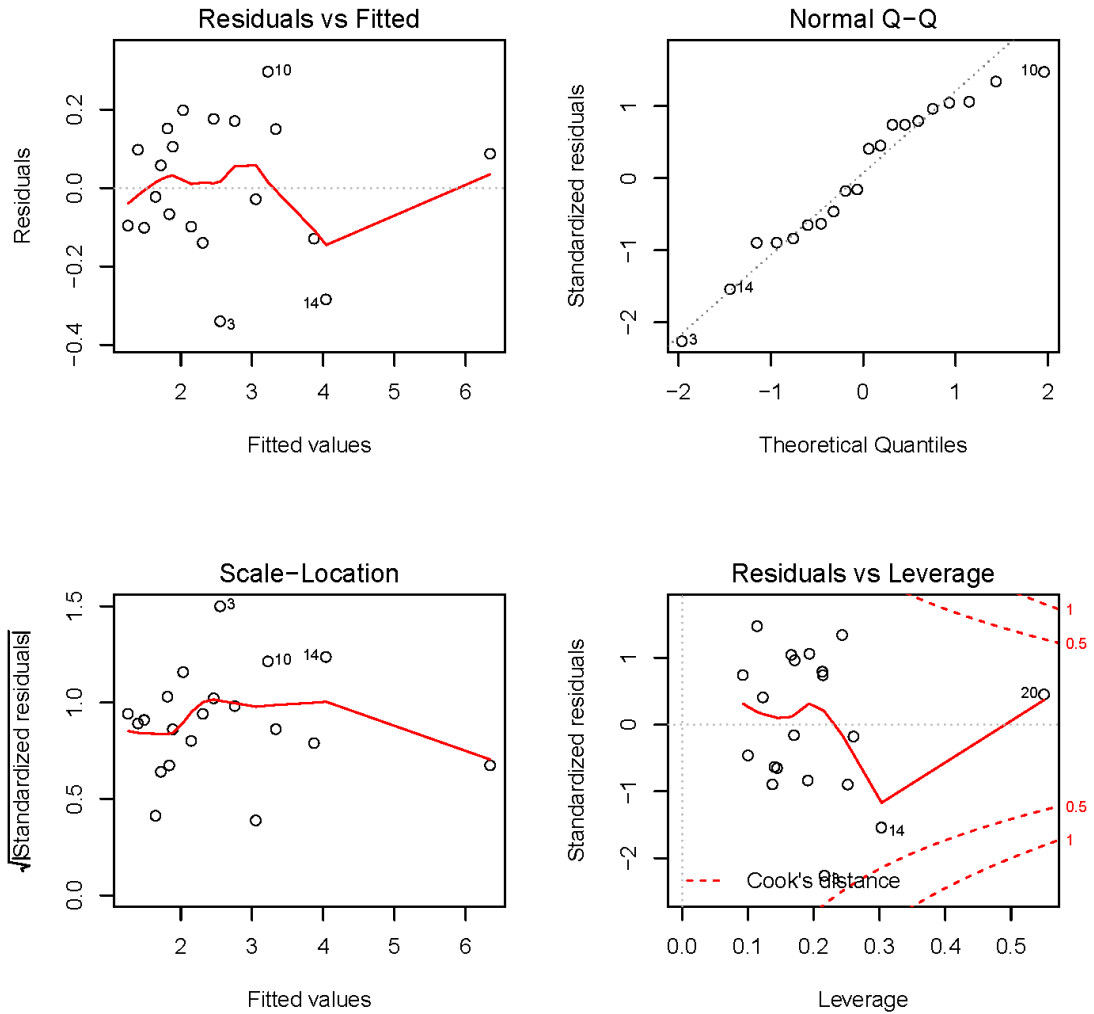


FIGURE 3.6: Analysis of Variance for the latency model.

shows the average amount that the response variable deviates from the true regression line. The models fit the data well as the residuals appear randomly scattered across the lines. However, there is also some non-linear behaviour shown through the outliers that are labelled and located furthest from the line. The bottom left plots test specifically whether the residuals increase as a function of the fitted value, and it can be seen that this is not the case for either model.

Top right plots test the normality of the residuals. Here, the quantiles of the residuals are plotted against the quantiles of the normal distribution, with a 45-degree dashed line plotted for reference. The plot is used to check that the residuals are normally distributed and if so, they should approximately follow the straight line. Finally, the bottom right plots analyse if there are any influential data points included in the model, *i.e.* outliers that do affect the regression. The upper and lower right corners of the plots are where these points would be located. If the influence or a particular sample exceeds Cook's distance, then it may have to be excluded from the model. This is the case for point 20 in the current model, which will reduce the accuracy of predictions for a high frequency and core number. This point cannot be excluded as this analysis

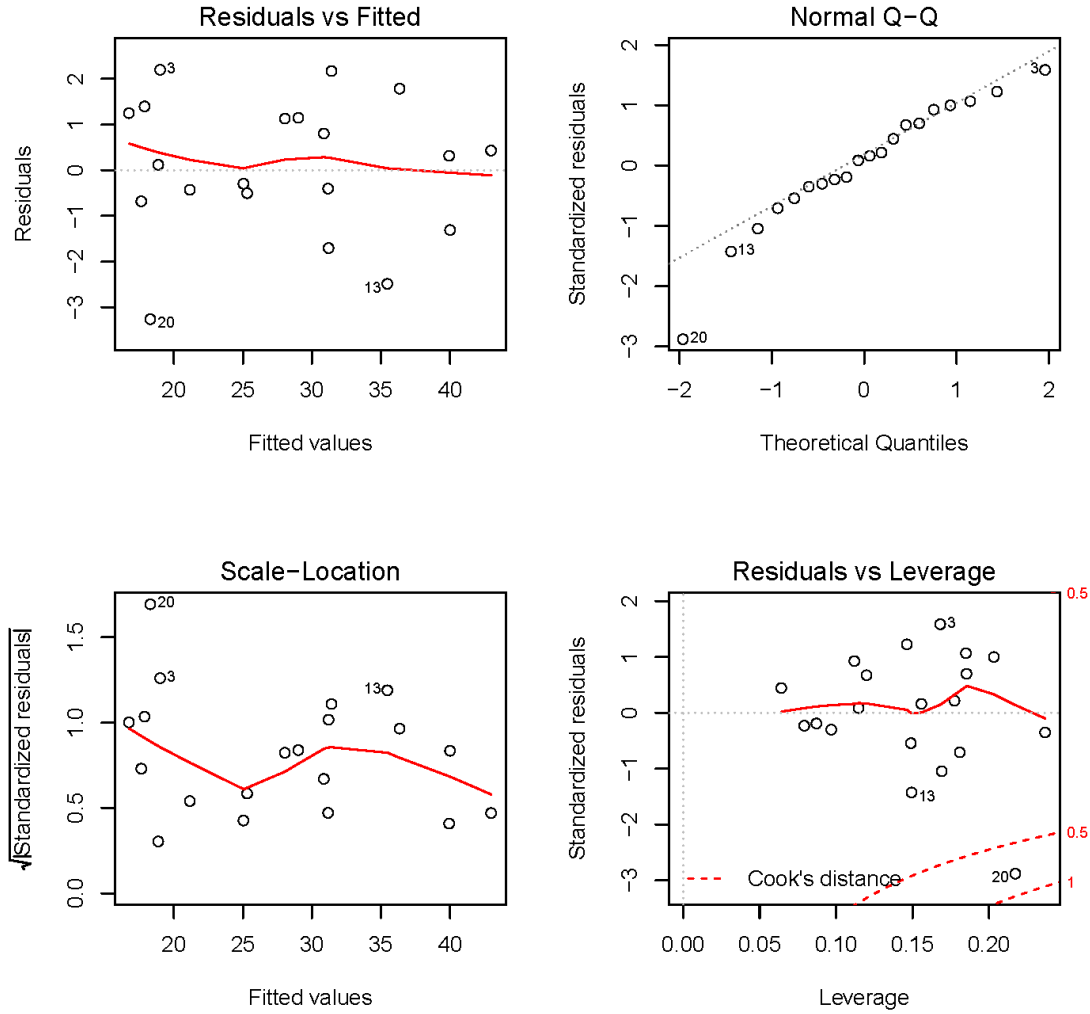


FIGURE 3.7: Analysis of Variance for the current model.

is not performed at runtime, however a weighting can be applied to points at the extremes of the model's valid input range in order to reduce the effect of these points. This is a modified form of robust regression where model coefficients are iteratively re-computed, with evaluation of the residuals and weighting of each point [139].

To summarise, through the statistical validation that has been presented, it is possible to conclude that the modelled values exhibit a significant degree of correlation with the measured values. This is due in part to the fact that the runtime model is generated using realistic component models of current (I) and latency (τ) from measured data. The high modelling accuracy will ensure that the RTM can achieve near-optimal operational conditions when optimising a platform and application during runtime. A demonstration of this process is presented in Section 4.5 for a multi-threaded application.

The next section presents a runtime optimisation method using gradient descent to find the optimal frequency and core settings under a given performance requirement.

3.4 Gradient Decent for Runtime Optimisation

The linear models are used to predict the power of the platform and performance of the application. In order to determine the optimal settings for the independent variables, *i.e.* the frequency and number of cores, an optimisation approach is required that can quickly and efficiently search the model to find the optimal operating point. This approach must also take into account any requirement thresholds by comparing them to the model predictions.

Gradient descent is a iterative approach for solving optimisation and minimisation problems. It is an alternative to classical optimisation algorithms because it requires fewer computations to reach a sufficiently accurate result. Descent methods in general produce the sequence $x^{(n)}$ as;

$$x^{(n+1)} = x^{(n)} - t^{(n)} \Delta x^{(n)} \quad (3.5)$$

where $n \geq 1$ and $t^{(n)} > 0$ [18]. $\Delta x^{(n)}$ is known as the step or search direction and $t^{(n)}$ is the step size or step length. Gradient descent can be used to find a local maxima or minima from a random or pre-selected starting point. Normally, the search direction is made following the negative gradient $\Delta x^{(n)} = -\Delta f(x)$, however performance increases as a function of frequency and number of cores therefore we choose to follow a positive gradient, $\Delta x^{(n)} = \Delta f(x)$, to a local maxima. Algorithm 1 describes the general gradient descent method for this approach where the step and update loop continues until the difference is less than the convergence criterion, ϵ .

ALGORITHM 1: General gradient descent method, adapted from [18].

Input: Starting point: x

```

1 repeat
2    $\Delta x := \Delta f(x)$ 
3   Choose step size  $t$ 
4   Update  $x := x + t\Delta x$ 
5 until  $f(x^{(n)}) - f(x^*) \leq \epsilon$ ;
```

A backtracking line search is used as an adaptive way to determine the next step size, t , where the next iteration will be along the line $x + t\Delta x$. The approach uses two constant parameters, α and β , where $0 < \alpha < 0.5$ and $0 < \beta < 1$. The step size is reduced as $t = \beta t$ while

$$f(x + t\Delta x) > f(x) + \alpha t \Delta f(x)^T \Delta x \quad (3.6)$$

Figure 3.8 illustrates the backtracking search area, where f is the model function, the lower dashed line is the extrapolation of f and the upper line is the same but at a lower slope by a factor of α . The search starts with step size t and reduces by a factor of β until Equation 3.6 no longer holds.

Algorithm 2 shows a pseudo-code for the gradient descent process in the context of the power and performance models presented in Section 3.2, when a performance target is provided as the input. The objective is to find the minimum power point, the frequency and the number of cores as the output. The operating frequency (g_n), the number of cores (c_n) and the step size are initialised (line 1). These are updated by a gradient descent (line 3-4). The step size (t) in the algorithm is based on the backtracking line search to ensure a fast convergence. While the predicted performance, F_{perf} , exceeds the specified performance target (line 5), the learning

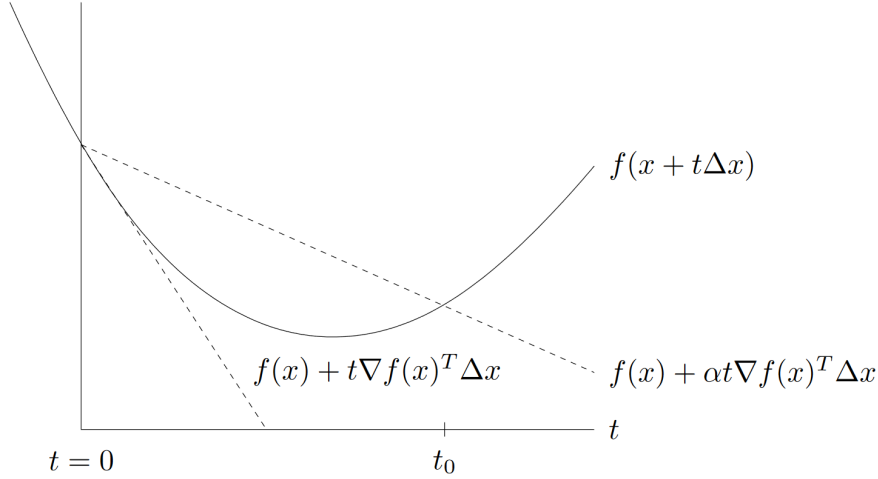


FIGURE 3.8: Backtracking line search for the step size of gradient descent, reprinted from [18].

ALGORITHM 2: Gradient descent based optimisation of performance and power.

Input: Performance Requirement: Perf_{req} **Output:** Minimum power point: Pwr_{min} , operating freq: g and number of cores: c 1 Initialise: $g_n = 0$, $c_n = 0$ and step size: $t = \frac{\partial}{\partial f} f_{\text{perf}}(g_n, c_n)$, $\beta = 0.5$ 2 **repeat**3 $g_{n+1} := g_n - t \frac{\partial}{\partial g} f_{\text{perf}}(g_n, c_n)$ 4 $c_{n+1} := c_n - t \frac{\partial}{\partial c} f_{\text{perf}}(g_n, c_n)$ 5 **while** $F_{\text{perf}}(f_{n+1}, c_{n+1}) > \text{Perf}_{\text{req}}$ **do**6 $t := \beta t$ 7 $g_{n+1} := g_n - t \frac{\partial}{\partial g} f_{\text{perf}}(g_n, c_n)$ 8 $c_{n+1} := c_n - t \frac{\partial}{\partial c} f_{\text{perf}}(g_n, c_n)$ 9 **end**10 $g_{n-1} := g_n$, $g_n := g_{n+1}$ 11 $c_{n-1} := c_n$, $c_n := c_{n+1}$ 12 **until** $g_{n+1} - g_n \leq \epsilon$ and $c_{n+1} - c_n \leq \epsilon$;13 **return** $f_{\text{perf}}(g_n, c_n)$ as Perf_{max}

rate is reduced (line 6) and g_n and c_n are further updated by another gradient descent (line 7-8). Predictions and updates are continued until the minimum performance target is met. The operating frequency and number of cores that provide the minimum power consumption, whilst meeting the specified performance target, are returned (g_n, c_n) .

An example of frequency and core selection through Algorithm 2 is shown in Figure 3.9a, illustrating how the gradient descent algorithm finds the optimum operating point. This is performed for the stereo matching application presented in Chapter 4 executing on the Xeon Phi platform, described in Section 2.6.1. Power consumption is shown using an overlaid colour map and the performance level is shown with contour lines. To determine the optimal frequency and number of cores, the power and performance are predicted using the lowest operating frequencies (g_i) and number of cores (c_i) initially. This is the bottom left corner in each figure. The step taken for each iteration is shown as an arrow with the new point at its end. For each step, the arrow points in the direction of the gradient of the performance. Table 3.4 lists the number of cores, the frequency, the predicted power and performance and the normalised power and performance for each of the steps taken by the algorithm. Power and performance are normalised to the range of the data in order for the step size of the algorithm to be updated correctly. Figure 3.9a shows

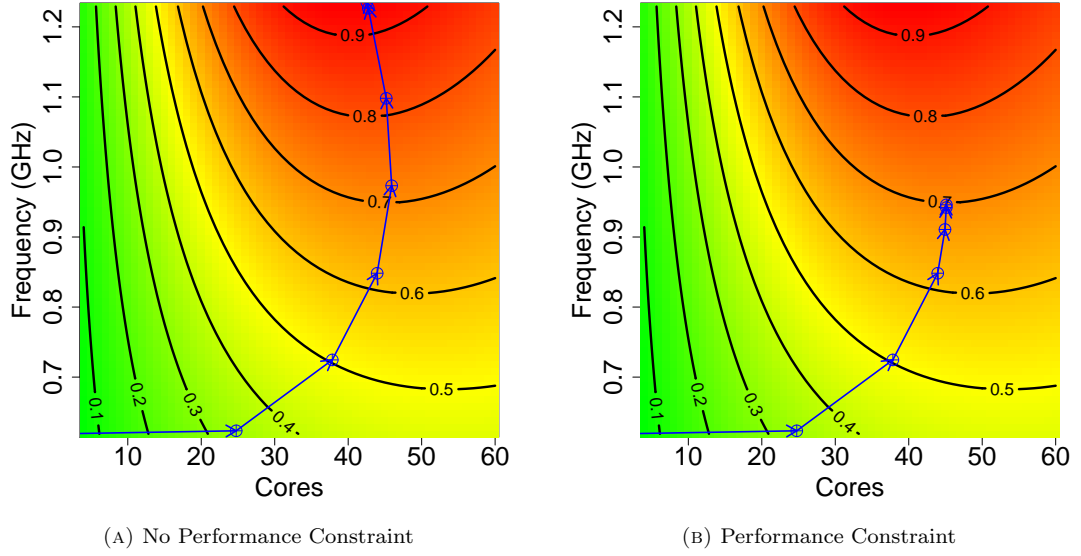


FIGURE 3.9: Runtime optimisation examples (a) without performance constraint and (b) with performance constraint. Power consumption is shown using an overlaid colour map and the performance level is shown with contour lines.

TABLE 3.4: Data for each step in the gradient descent search for performance optimisation of the stereo matching application executing on the Xeon Phi platform.

Iteration	Num. Cores	Frequency (MHz)	Predicted Power (W)	Predicted Perf. (FPS)	Step size	Norm. Power	Norm. Perf.
1	1.00	619	12.0	0.0166	0.412	0.220	0.0176
2	24.8	623	18.8	0.341	0.412	0.344	0.362
3	37.8	725	26.8	0.504	0.412	0.490	0.535
4	43.9	848	34.8	0.619	0.412	0.638	0.658
5	45.9	973	42.2	0.719	0.412	0.772	0.764
6	45.2	1100	48.6	0.820	0.412	0.889	0.872
7	42.7	1230	54.3	0.933	0.0258	0.994	0.992
8	42.5	1240	54.6	0.941	0.00644	1.00	1.00

that the performance does not vary greatly with changes in frequency at a low number of cores, hence the contours are more vertically spaced. This gradient encourages the algorithm to make a step predominantly in the x-direction and increase the number of core much more than the frequency. This can be seen in Table 3.4 where the search starts at one core with an operating frequency of 619 MHz, giving 0.0166 predicted FPS and 12.0 W, but the second step moves increases to 24 cores and only increases the frequency by 4 MHz. At higher core numbers, the performance does increase significantly with frequency, therefore the contours are predominantly horizontal. The gradient decent process continues until it converges at 43 cores with a closest operating frequency of 1240 MHz giving the highest predicted performance of 0.951 fps and a predicted power consumption of 54.6 W.

Figure 3.9b demonstrates another example of the same algorithm this time with a performance target of 0.70 FPS applied. The data in Table 3.5 lists the steps taken by the algorithm in finding the required number of core and frequency. However, this time the search only continues until the performance constraint is met, at which point the corresponding operating frequency and core allocation is selected (46 cores at 952 MHz).

TABLE 3.5: Gradient descent search data for optimisation of the stereo matching application with a performance constraint, executing on the Xeon Phi platform.

Iteration	Num. Cores	Frequency (MHz)	Predicted Power (W)	Predicted Perf. (FPS)	Step size	Norm. Power	Norm. Perf.
1	1.0	619	12.0	0.0166	0.412	0.298	0.0238
2	24.8	623	18.8	0.341	0.412	0.466	0.489
3	37.8	725	26.8	0.504	0.412	0.664	0.722
4	43.9	848	34.8	0.619	0.206	0.864	0.888
5	44.9	911	38.4	0.669	0.103	0.953	0.960
6	45.1	942	40.1	0.695	0.0129	0.995	0.996
7	45.1	946	40.3	0.698	0.00644	1.00	1.00

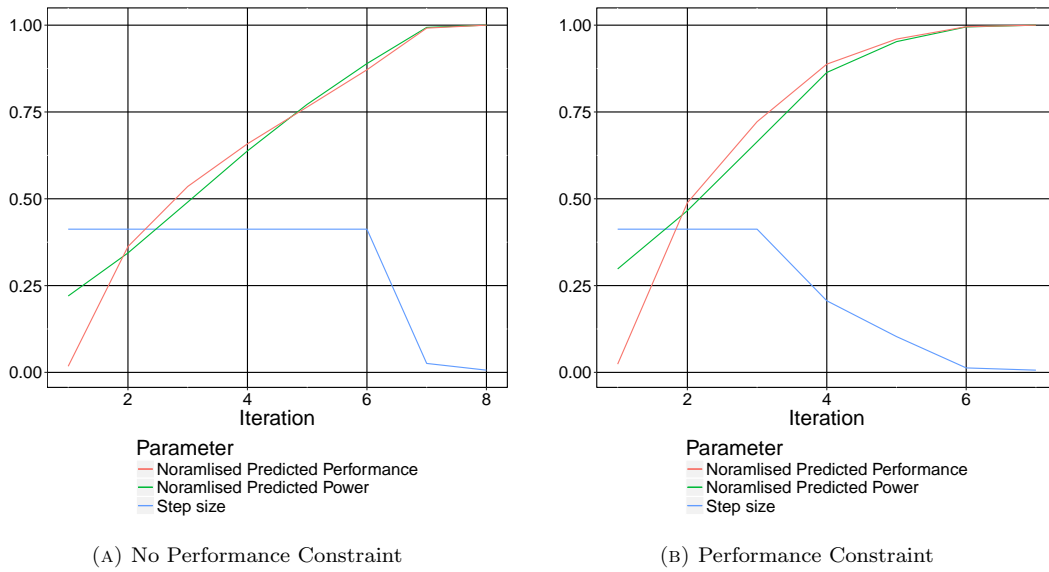


FIGURE 3.10: Step size, normalised power and normalised performance over the gradient descent iterations.

Figure 3.10 illustrates the data of Table 3.4 and 3.5 graphically to show the trend in predicted power and performance as the gradient decent algorithm iterates. It also highlights how the step size reduces as the algorithm converges towards the performance requirement in Figure 3.10b and the maximum performance in Figure 3.10a. Predicted power and performance are normalised for clarity in plotting the figure, therefore the normalising value is the power/performance of the final iteration.

3.5 Runtime Manager Overheads

The proposed approach incurs runtime overheads due to the various runtime adaptation operations, including sensor measurement, model training and gradient-descent optimisation searches. These are illustrated with a timing diagram in Figure 3.11, using the stereo matching application executing on the Xeon Phi platform. The fluctuating application performance requirement is shown in the top plot, with the periods of RTM optimisation (Optimise), power measurement (Measure) and application execution shown in the lower plot. Runtime optimisation exhibits an

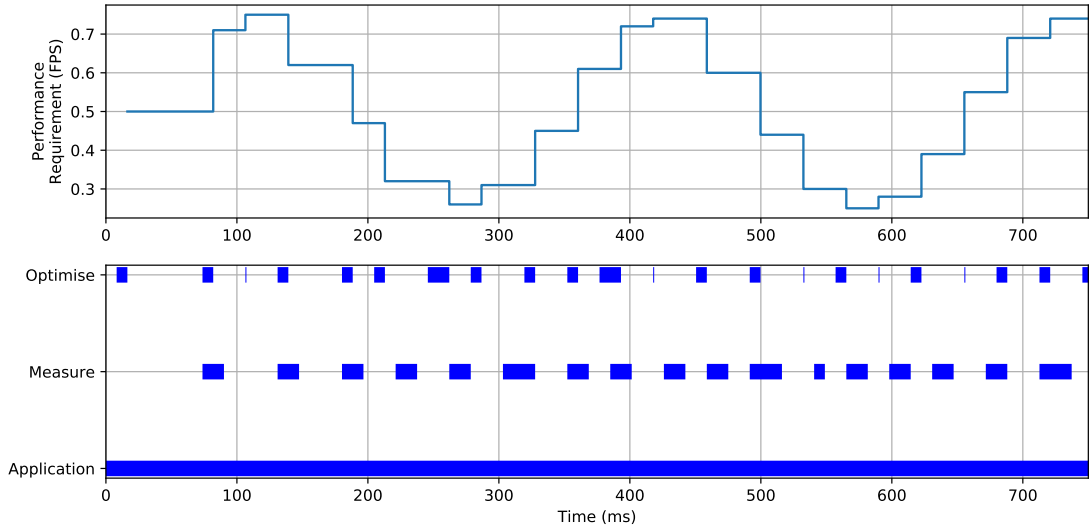


FIGURE 3.11: Timing diagram of the optimisation and measurement overheads of the runtime management processes during application execution.

average overhead of 7.06 ms, due to the gradient-descent based search operation performed on the runtime model. This occurs each time the performance target or power constraint is changed and optimisation can only be performed after the model is built. Power measurement of the Xeon Phi is conducted at regular intervals during application execution, in order to calculate the energy per frame, and this process exhibits an overhead of 16.1 ms on average. These two process occur in parallel with application execution.

Collecting training data takes a period of 20 frames, as established in Section 3.3. The length of time that this represents is application-dependent as it is affected by the time taken to repeatedly execute it over the required number of samples. For instance, the low performance operating points that are used for training the model take more time to collect. The application will have to be executed across the range of operating points as part of training, to build a complete model, however the reduced number of training samples makes this interval a smaller overhead. Executing the training function, estimating the MLR coefficients of the power and performance models, incurs an overhead of approximately 2 ms. This is not shown on the figure as it occurs only at the beginning of application execution, before optimisation takes place.

3.6 Summary

An adaptive runtime modelling approach has been presented to predict the power consumption of a platform and the performance of an application during execution. MLR has been used to build power and performance models of a multi-core system and multi-threaded application. The models have been validated in terms of their prediction error and residual error, in order to establish the minimum training interval required such that the models are accurate but can be generated at runtime.

A gradient-descent optimisation approach has been presented to predict power and performance from the models and establish optimal frequency and core number settings. The runtime manager

overheads have been quantified and a breakdown is shown for where this occurs during the training and optimisation stages.

In the next chapter, this approach is applied to a system where a dynamic, parallel computer-vision application is executing on a multi-core platform. The operating space of the application-platform combination is characterised to demonstrate that there are power and performance scaling opportunities. During runtime experiments, the power consumption of the platform is optimised by adjusting the level of multi-threading in the application, which relates directly to the number of core utilised, and changing their operating frequency. This process put the components presented in this chapter into practice in a real-world demonstration.

Chapter 4

Runtime Management of a Parallel Stereo Matching Algorithm

Stereo vision lies amongst the wider domains of computer vision algorithms, which have many real world applications, especially in new and emerging domains. Examples include gaming (VR), robotics (robot vision), automotive (self-driving, collision avoidance, adaptive cruise control), smart cities (monitoring/surveillance) with many more application domains featuring some form of computer vision.

Stereo vision considers computer vision configurations where the same 3D scene is captured by two or more different viewpoints. This configuration is recognisable for its similarity to the vision system of many biological organisms. The eyes are positioned apart from each other on the animal's head such that they can capture two overlapping perspectives to perceive a 3D interpretation of their surroundings. The proportion of the field of view of the animal that is shared by both eyes is known as the range of binocular vision.

The primary advantages of a stereoscopic camera arrangement over a single camera setup is the ability for depth information about the original 3D scene to be reconstructed from the Two Dimensional (2D) images captured. Stereo matching is the process of taking two or more images and estimating a 3D model of the scene by finding matching pixels in the images and converting their 2D positions into 3D depths [140].

The dynamic environment in which this application is typically found was part of the motivation for choosing it to demonstrate the runtime management approach developed. A significant image processing component in the application makes it suitable for GPU and Digital Signal Processor (DSP) components which are now commonly found in mobile platforms. In addition, a local algorithm is chosen for the implementation because it is scalable due to implicit parallelism and low data dependence properties. More details of the literature around this are given in Section 2.3.3. Most importantly, scalability enables operation across a range of power-performance points, depending on system constraints. Therefore the application's parallelism is exposed as a

parameter that can be tuned by the RTM during execution. Details on how this dynamic parallelism is introduced into the application is presented in Section 4.2.3, including how parallelism can be enabled across several dimensions in some stages of the algorithm.

There are numerous additional tuning parameters not investigated in this chapter that could be exposed from the application, which can directly affect its performance and/or accuracy. For example, adjusting the coarseness of the various filtering operations can lead to a performance trade-off for accuracy by changing the amount of computation that is performed per frame. These are commonly found across image processing applications, however stereo matching was chosen in particular because it combines several processing stages, enabling a multi-faceted tuning and monitoring scenario.

This chapter's contributions can be summarised as:

1. Development of a multi-threaded implementation of the disparity estimation algorithm for stereo matching.
2. Characterisation of the operating space of the application on a multi-core platform with scaling and trade-off analysis.
3. Deployment of the runtime management approach to optimise the execution of the application in response to specified performance targets.

Material from this chapter has also been published in ACM TECS as Leech *et al.* [25]. Section 4.1 begins the chapter by modelling a stereo vision system and deriving the properties of each camera. The mathematical derivation of the disparity estimation algorithm, the core process in stereo matching, is presented in Section 4.2, including evaluation of its accuracy in Section 4.2.2 and details of its multi-threaded software implementation in Section 4.2.3. Characterisation of the algorithm on a multi-core platform is shown in Section 4.3, which motivates the experimentation in Section 4.5 with the runtime management process presented in Chapter 3.

4.1 Geometric Model of a Stereo Vision System

In order to calculate the depth of objects in the scene, characterisation of the physical stereo camera configuration is required so that the mathematical algorithm, presented in Section 4.2, is fully validated and to ensure that it operates on correctly configured images.

4.1.1 Pinhole Camera Model

The pinhole camera model is used to describe the projection of a point in 3D space $M = (X, Y, Z)^T$ onto a 2D image plane, giving it coordinates $m = (u, v)$, in front of a point where every ray of the scene is direct to [20, 141, 142]. This is known as the perspective transformation and is given by the equation:

$$sm = \mathbf{P}M \tag{4.1}$$

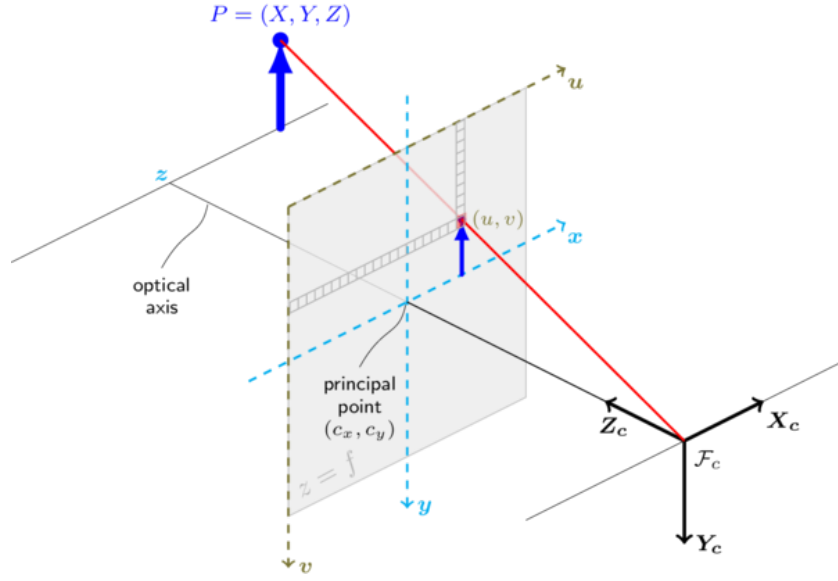


FIGURE 4.1: Pinhole camera model illustrating the projection of a 3D point in space to the image plane, reprinted from [19].

where s is the scale factor and \mathbf{P} is the projective matrix that can be decomposed to $\mathbf{P} = \mathbf{A}[\mathbf{R}|\mathbf{t}]$. \mathbf{A} is the intrinsic parameter matrix of the camera that contains the parameters (c_x, c_y) , the principle point or optical centre of the camera, and (f_x, f_y) , the focal lengths of the camera in pixels. $[\mathbf{R}|\mathbf{t}]$ is the joint rotation-translation matrix, which is the rotational camera matrix at position \mathbf{t} . This is also called the extrinsic parameter matrix. The two matrices are arranged as:

$$\mathbf{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (4.2)$$

$$[\mathbf{R}|\mathbf{t}] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \quad (4.3)$$

$[\mathbf{R}|\mathbf{t}]$ can be interpreted as the matrix that relates the camera coordinate system (u, v) to the real-world coordinate system of a point $(X, Y, Z)^T$. Equation 4.1 can now be expanded to:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (4.4)$$

Figure 4.1 illustrates the projection process of the pinhole camera model where $(X, Y, Z)^T$ are the coordinates of a 3D point in the world coordinate space and (u, v) are the coordinates of the projection point in pixels. The diagram also shows the relationship between the (u, v) coordinate system and the equivalent (x, y) coordinate at $z = f$. The (x, y) coordinate is centred on the principle point and can be mapped to the camera coordinate system through:

$$u = \frac{f_x x}{z} + c_x \quad (4.5)$$

$$v = \frac{f_y y}{z} + c_y \quad (4.6)$$

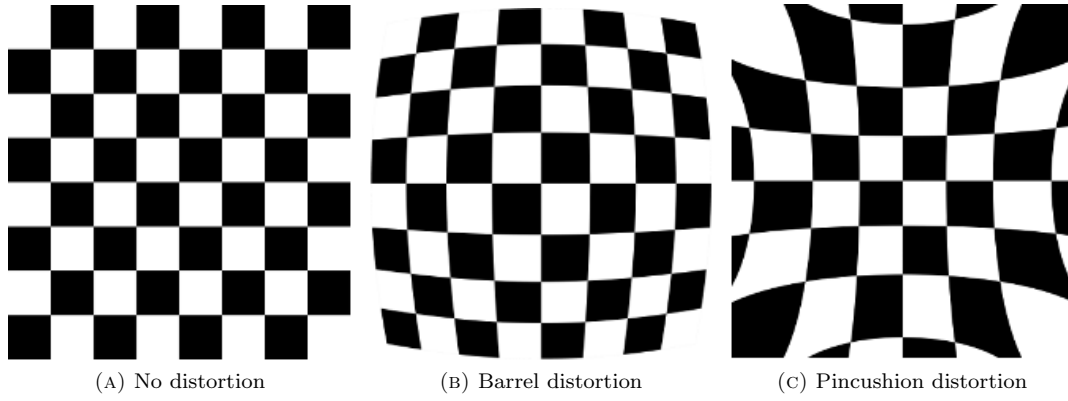


FIGURE 4.2: The two modes of radial distortion, reprinted from [19].

4.1.2 Distortion

Both radial and tangential distortion can occur in a image due to imperfections in the camera manufacturing process, which cause inaccuracies in the camera model. The effect of each type of distortion on a chessboard image is illustrated in Figure 4.2. Radial distortion is caused by the lens of the camera, which is often spherical rather than an ideal parabolic shape [20], and can be either positive (barrel distortion - Figure 4.2b) or negative (pincushion distortion - Figure 4.2c). This distortion can be corrected using the first few terms of a Taylor expansion at radius $r = 0$. For most cameras, with a small amount of radial distortion, only the first two terms are required (k_1, k_2) but k_3 may be added for more highly- or intentionally-distorted cameras such as fish-eye lenses. This leads to the following equations to correct for radial distortion:

$$\begin{aligned} x_{distorted} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) \\ y_{distorted} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6) \end{aligned} \tag{4.7}$$

where (x, y) is the original pixel location. Tangential distortion can occurs when the image plane is not parallel to the imaging lens. This can be as a result of low-cost manufacturing where the CMOS sensor is not glued flat to the back of the camera. It is corrected for using the parameters p_1 and p_2 in the formula:

$$\begin{aligned} x_{distorted} &= x + [2p_1xy + p_2(r^2 + 2x^2)] \\ y_{distorted} &= y + [p_1(r^2 + 2y^2) + 2p_2xy] \end{aligned} \tag{4.8}$$

Correcting these distortions ensures that the model of Equation 4.4 is more accurate.

4.1.3 Epipolar Geometry

The pinhole camera model can be extended to consider two cameras taking an image of the same scene, called multi-view or epipolar geometry. This is the basic geometry of a stereo imaging system and represents the most generic case where either camera can be positioned in any location. Figure 4.3 shows the epipolar plane created by the triangulation of an observed point P and the two centres of projection of the two cameras, O_l and O_r . A pair of epipoles, e_l

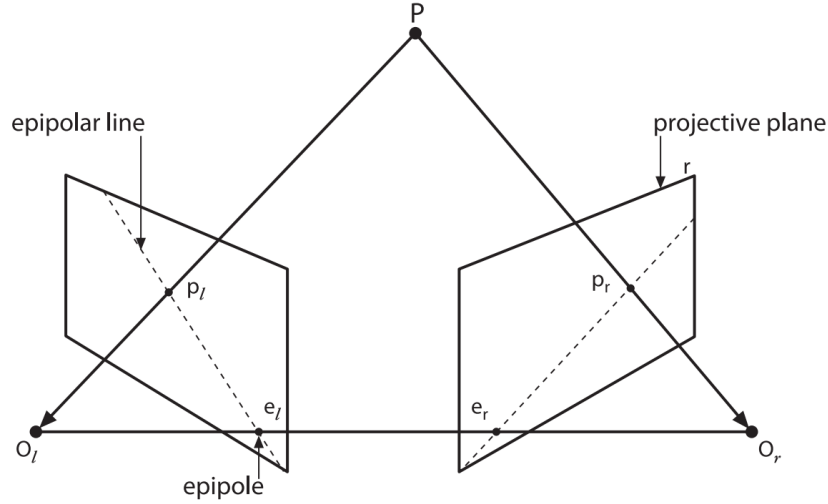


FIGURE 4.3: Epipolar geometry of a stereo imaging system, reprinted from [20].

and e_r are located at the points where the images planes are intersected by the line joining the points of projection. The point P is projected onto the two imaging planes at p_l and p_r , which form the epipolar lines from the epipoles ($e_l p_l$ and $e_r p_r$).

Taking the right image plane, the distance to point P cannot be determined with this single image as the projected point p_r can be anywhere on the line defined from O_r through p_r . However, the epipolar line of the left camera (defined by $e_l p_l$) is the projection of the line on the left image plane. Therefore all the possible locations of p_r is the line through the corresponding point p_l in the left image plane and the epipolar point. This is known as the epipolar constraint and simplifies the two-dimensional search for matching points to a single dimension along the epipolar lines.

The relationship between the projected points p_l and p_r , following the epipolar constraint, can be described in terms of a matrix transformation called the fundamental matrix [142, 20]:

$$q_r^T F q_l = 0 \quad \text{where} \quad q = \mathbf{A}p \quad (4.9)$$

\mathbf{A} is the camera intrinsics matrix from Equation 4.2 and the conversion from q to p allows \mathbf{F} to operate in pixel rather than physical coordinates. This matrix encodes the translation and rotation required to move between the left and right image planes. The derivation of this matrix is not presented here but can be found in the relevant literature, such as [142] and [20]. As is shown in the following section, several conditions of the experimental setup used in this thesis simplify the fundamental matrix to a pure translation.

4.1.4 Stereo Geometry

The fundamental matrix can be used to simplify the epipolar geometry such that the image planes are aligned so they are coplanar (*i.e.* the line joining the camera centres is perpendicular to the optical axis). This removes the rotational component such that the epipolar lines are horizontal in both image planes [140]. This simplifies the pixel correspondence search space to just the horizontal scanlines, which can be computed independently for each row. In practice, a

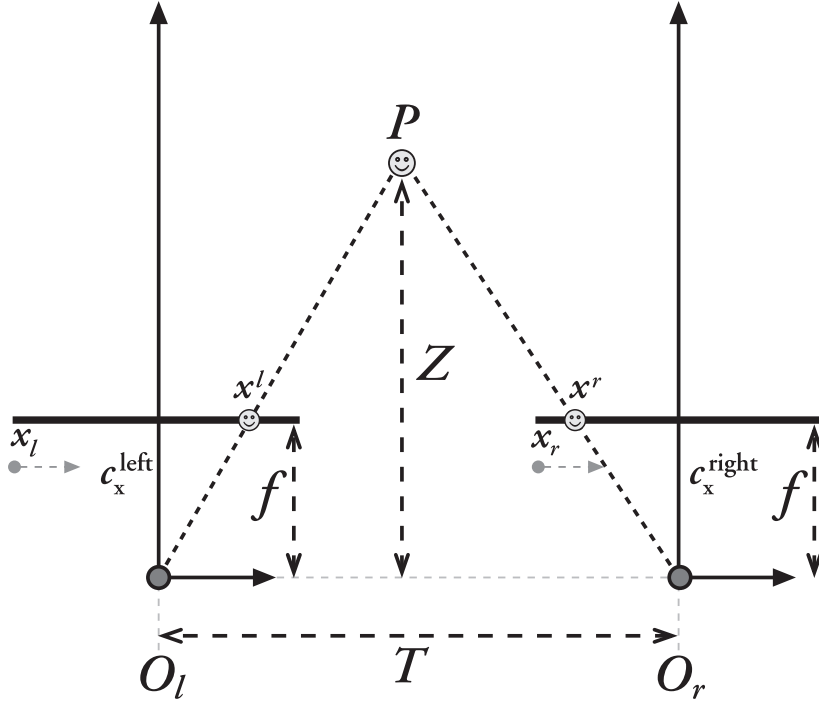


FIGURE 4.4: Coplanar geometry of a calibrated stereo imaging setup reprinted from [20].

stereo imaging setup is an approximation of this arrangement and therefore calibration may be required to remove a small rotational component. Figure 4.4 illustrates the new arrangement from a 2D perspective, since correspondence will now be along the same row in both image planes (*i.e.* $y_l = y_r$). This is known as the *standard rectified geometry* and is employed in the majority of stereo camera setups [140]. Furthermore, many stereo correspondence (stereo matching) algorithms assume this geometry for input data.

Figure 4.4 shows a point P in the scene projected onto the image planes at x_l and x_r (the horizontal coordinates of p_l and p_r). The disparity between these coordinates is defined as $d = x_l - x_r$ and a relationship between the disparity and the 3D depth Z can be derived using similar triangles as:

$$\frac{T - d}{Z - f} = \frac{T}{Z} \quad \Rightarrow \quad Z = \frac{fT}{x_l - x_r} \quad (4.10)$$

Therefore, the process of establishing the depth of the objects in a scene becomes the process of estimating the disparity map $d(x, y)$.

4.2 Disparity Estimation

Disparity Estimation (DE) is the algorithm used to perform stereo matching and extract depth information from a pair of rectified images in a stereoscopic configuration. In this work, a dense stereo correspondence method is used, meaning that the disparity of each pixel in the image plane is estimated. The method is based on the algorithm presented in Hosni *et al.* [83].

Due to the geometrical assumptions discussed in Section 4.1.4, the correspondence of a pixel at coordinate (x, y) of the reference image, can be found at the same vertical coordinate y , in the

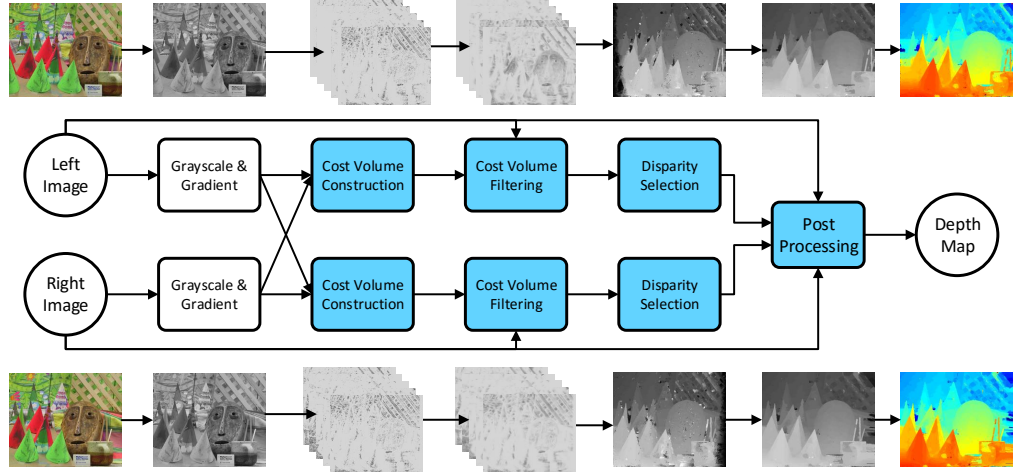


FIGURE 4.5: Block diagram of the DE algorithm composed of CVC, CVF, DS and PP stages. The cost volume is created in CVC, with D slides, each slice is filtered in CVF and then combined in DS. The shaded regions are the enhanced parallel stages which offer opportunity for runtime tuning of the threading level.

target image within a maximum horizontal bound called the disparity range $[0, D)$ [143]. The location difference of corresponding pixels in both images is called the disparity and is used to calculate the depth in metres.

The source images used in this section, for demonstration of the algorithm, are taken from the Middlebury stereo vision dataset¹ [22]. The images are rectified so that motion is purely horizontal and they are accompanied by a ground-truth disparity map so the accuracy of the algorithm's implementation can be assessed.

4.2.1 Algorithm

Depending on the correspondence approach used, DE algorithms mostly follow four high-level steps: cost computation, cost aggregation, disparity computation and disparity refinement. This section describes the structure and properties of the DE algorithm, illustrated by Figure 4.5. The algorithm that is used in this work is composed of four key stages, with the addition of pre-processing, these are; Cost Volume Construction (CVC), Cost Volume Filtering (CVF), Disparity Selection (DS) and Post Processing (PP).

4.2.1.1 Pre-processing

Assuming rectified images, the only remaining pre-processing to be conducted is greyscale conversion and gradient filtering of the input RGB colour images. The gradient image is required as an input alongside the RGB image to the CVC stage. Greyscale conversion is achieved using the standard method by taking contributions from the red, green and blue components to calculate a luminance value Y :

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (4.11)$$

¹Available online at: vision.middlebury.edu/stereo/

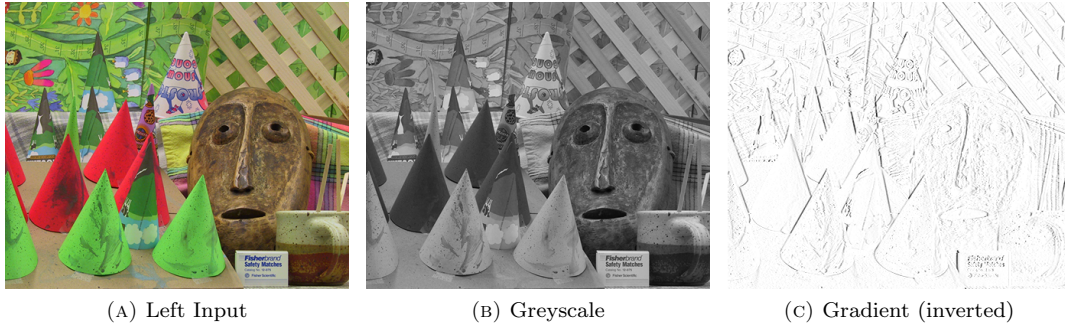


FIGURE 4.6: Left input image, greyscale and x-gradient images produced from pre-processing.

Gradient filtering is performed on the greyscale images using the Sobel operator. The filter is commonly used in image processing as an edge detection algorithm as edges often coincide with significant changes in luminance [20]. In this way, it is useful for stereo correspondence as it can provide an additional metric in the assessment of the similarity between the pixels in each image. The Sobel filter computes the x and y derivatives of an input pixel using a variable size kernel of dimensions $ksize \times ksize$. As the correspondence in the images is purely horizontal, only the x-gradient is computed, with $ksize = 1$ and the 3×1 kernel of $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$.

Figure 4.6 shows the left input RGB image (A) next to the greyscale convert image (B) and the x-gradient filtered image computed from the greyscale (C) (the gradient scale has been inverted for printing). The input image and gradient image are passed to the next stage of CVC.

4.2.1.2 Cost Volume Construction

The first correspondence stage of disparity estimation is Cost Volume Construction (CVC). This process involves the comparison of each pixel between the two images over the disparity range $[0, D)$. The resulting cost volume has dimensions width \times height $\times D$. The range of comparisons made between the two input images for each pixel is demonstrated in Figure 4.7a. A horizontal scan-line connects the two images along the same y coordinate, with the pixel whose disparity is going to be determined marked with a square box and the label p . The disparity range is marked in the right image as a rectangular box and the location of the pixel from the left image is labelled at $p - d$, where d is in the range $[0, D)$.

A cost value is assigned to each pixel p in the left image based on the dissimilarity between it and a pixel in the right image. The cost value is calculated using the absolute difference of colours (4.12) and gradients (4.13) between the two pixels. Pixels that are more similar in colour and have a similar gradient with neighbouring pixels result in a lower cost score, which indicates that they are more likely to be the same point in 3D space.

$M(p, d)$ (4.12) is the cost contribution, at pixel p and disparity d , for the difference in colours between pixel p and pixel $p - d = (x - d, y)$. I is the pixel value of each colour channel in the input image.

$$M(p, d) = \sum_{i=1}^3 |I_{left}^i(p) - I_{right}^i(p - d)| \quad (4.12)$$



(A) Input Images

(B) CVC at $d = 19$ (C) CVC at $d = 33$ (D) CVC at $d = 46$

FIGURE 4.7: (A) Left and right input images with illustration of the matching process, (B - D) slices through the cost volume after construction with Equation 4.14 at $d = 19, 33$ and 46 .

$G(p, d)$ (4.13) is the cost contribution from the x-gradient at pixel p and disparity d . This time, I represents the luminance of the left and right pixels and ∇_x is the gradient in the x direction, as found by the Sobel filter in the previous section.

$$G(p, d) = |\nabla_x(I_{left}(p)) - \nabla_x(I_{right}(p))| \quad (4.13)$$

A cost function (Equation 4.14) is used to balance the contribution from the colour difference and gradient difference using a weighting variable α . The value of α is typically found empirically but then remains constant. T_c and T_g are bounding threshold values for the colour and gradient cost contributions respectively for forming the overall cost:

$$C(p, d) = \alpha \cdot \min(T_c, M(p, d)) + (1 - \alpha) \cdot \min(T_g, G(p, d)) \quad (4.14)$$

The cost calculations for every pixel at one disparity forms a single slice of the cost volume. Three example slices at disparities 19, 33 and 46 are shown in Figure 4.7 for a near, mid-range and far matching region. Darker regions indicate a lower cost and therefore a higher match. Low cost pixels can be seen in all three CVC slices for areas that do not correlate with the ground truth and so would be erroneous were they to be matched. This occurs when a similar colour and gradient pixel has been found at a different disparity to the ground truth.

4.2.1.3 Cost Volume Filtering

The second stage in the disparity estimation process is Cost Volume Filtering (CVF), analogous to the generalised term cost aggregation. This stage is only necessary for local and window-based

disparity estimation methods [140]. The filtering operation on the cost volume is defined as in Equation 4.15, which is applied to the built cost volume. $q(p, d)$ is the filtered cost value at pixel p and disparity d and $C(p, d)$ is the unfiltered cost value at the same location.

$$q(p, d) = \sum W_{i,j}(I)C(p, d) \quad (4.15)$$

Filtering is performed with the Guided Image Filtering (GIF) method and is applied to each slice of the cost volume. The GIF performs an edge-preserving smoothing operation, similar to that of the bilateral filter, but produces more accurate results near edges [82]. It also has a non-approximate linear run time for the underlying algorithm, which is only dependent on the image size and not the kernel size.

The GIF is defined as a general linear translation-variant filtering process that uses a guidance image I to filter an input or guided image p . In the case of DE, the original colour image is used as the guidance image and the cost volume slice at d is the guided image.

A full derivation of the filter function can be found in He *et al.* [82] but the important aspect of this for the purposes of DE is the filter kernel, shown in Equation 4.16. At a high level, $W_{i,j}$ is a weighting function that favours pixels in the kernel that have similar colour (or luminance) to that of the central pixel. The filter operates in a square window ω_k , centred at pixel k with dimensions $r \times r$. The covariance and mean of the of I in the window ω_k are given by σ_k and μ_k . The number of pixels in the filter window is $|\omega|$, with ϵ used as a smoothing parameter.

$$W_{i,j} = \frac{1}{|\omega|^2} \sum_{k:(i,j) \in \omega_k} \left(1 + \frac{(I_i - \mu_k)(I_j - \mu_k)}{\sigma_k^2 + \epsilon} \right) \quad (4.16)$$

One key advantage of the GIF is that the filter weights can be computed from a set of linear equations (see [82]) which can be decomposed into a series of mean filter operations with radius r . These can be computed in $O(N)$ time where N is the number of pixel in the colour image. In practise, the filtering process can be realised as the series of steps shown in 3 with mean filtering (f_{mean}), correlation (corr), variance (var) and covariance (cov) operations.

ALGORITHM 3: Guided image filter pseudo-code, reprinted from [82]

Input: filtering input image p , guidance image I , radius r , regularisation ϵ

Output: filtering output q

```

1  $mean_I = f_{\text{mean}}(I)$ 
2  $mean_p = f_{\text{mean}}(p)$ 
3  $corr_I = f_{\text{mean}}(I * I)$ 
4  $corr_{Ip} = f_{\text{mean}}(I * p)$ 
5  $var_I = corr_I - mean_I * mean_I$ 
6  $cov_{Ip} = corr_{Ip} - mean_I * mean_p$ 
7  $a = cov_{Ip} / (var_I + \epsilon)$ 
8  $b = mean_p - a * mean_I$ 
9  $mean_a = f_{\text{mean}}(a)$ 
10  $mean_b = f_{\text{mean}}(b)$ 
11  $q = mean_a * I + mean_b$ 
12 return  $q$ 
```

The operation of the GIF on the cost volume is shown in Figure 4.8 for two slices at different disparities (from (A) to (D) and (B) to (E)). The affect of filtering the cost volume on the

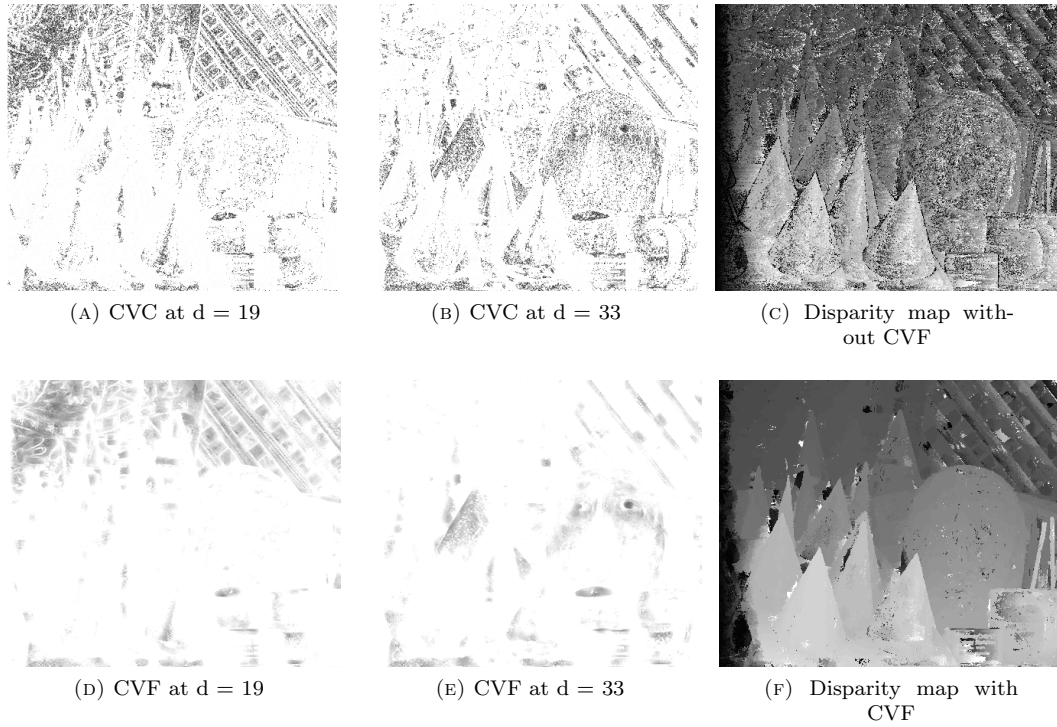


FIGURE 4.8: Cost volume slices before (A - B) and after (D - E) filtering with Equation 4.15 at disparities $d = 19$ and 33 . The disparity map built from an unfiltered (C) and filtered (F) cost volume.

disparity map is also shown (between (C) and (F)), the formation of which is covered in the next section.

Filtering the cost volume has an impact on the resolution of the disparity map, by smoothing the per-pixel costs, but it increases the overall accuracy of the disparity map as a result. Per-pixel accuracy is rarely required by subsequent processes that use the depth map.

4.2.1.4 Disparity Selection

The third stage of the DE algorithm is disparity selection, which is performed to generate the first instance of the disparity map. Selection involves the condensation of the cost volume back down to a 2D image and is performed through a winner-takes-all strategy to find the best disparity d_p value for each pixel p in the image across the disparity range. Equation 4.17 shows this process mathematically, where D represents the upper bound of the disparity range $[0, D)$, within which the best disparity value must lie.

$$d_p = \underset{d \in D}{\operatorname{argmin}} q(p, d) \quad (4.17)$$

The lowest cost value for each pixel is identified from across all disparities in the cost volume. This represents the most likely distance of the same point in space between the two images. The corresponding disparity value is encoded in the disparity map.

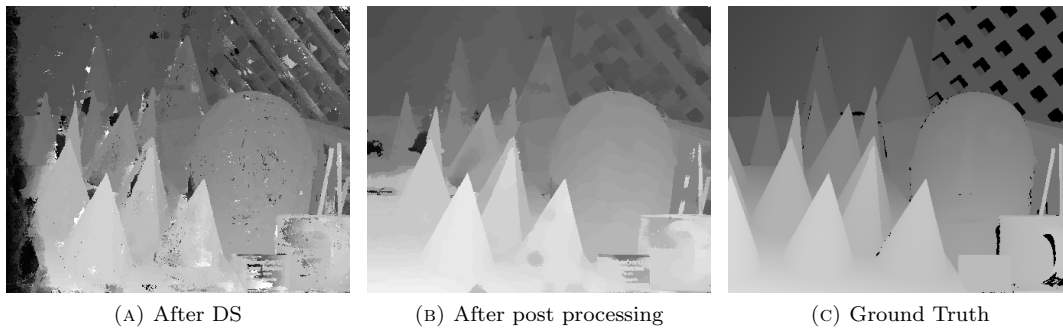


FIGURE 4.9: The affect of post processing illustrated with a comparison between the disparity map (A) after the selection stage, (B) after post processing and (C) the ground truth map from the dataset source.

The disparity map for the example image used here is shown in Figure 4.9a and a qualitative assessment shows a good similarity to the ground truth in Figure 4.9c. However, many artefacts still exist in the map that additional filtering can remove. This post processing is discussed in the following section.

4.2.1.5 Post Processing

The last stage of the algorithm is post processing, which is applied to the disparity map. This stage can be sub-divided into three distinct operations. These are occlusion detection, invalid pixel filling and weighted median filtering.

In order to perform the first two operations, an inverse disparity map is required, that is, a disparity map computed from right to left. This results in having a disparity map from both the left and right images' perspectives. Both maps are used to perform a left-right consistency check, to identify and fill mismatched pixels between the two maps. Invalid pixels are replaced with the closest consistent preceding pixel.

After the consistency check, a weighted median filter is used to remove any remaining artefacts in the output disparity map. The filter is based on that presented by Zhang *et al.* [144], chosen for its reduced computational complexity through the use of a joint-histogram representation and fast data access implementation.

The next section presents a quantitative assessment of the disparity map accuracy for multiple test image pairs, following the algorithm outlined in the past section.

4.2.2 Quantitative Evaluation of Accuracy

The quality of the output disparity map can be assessed quantitatively whenever a ground truth map is provided. The ground truth is often calculated using an alternative depth measurement technology or with physical measurement of the depth of each location in a scene. This provides a golden reference to compare with the computed disparity map.

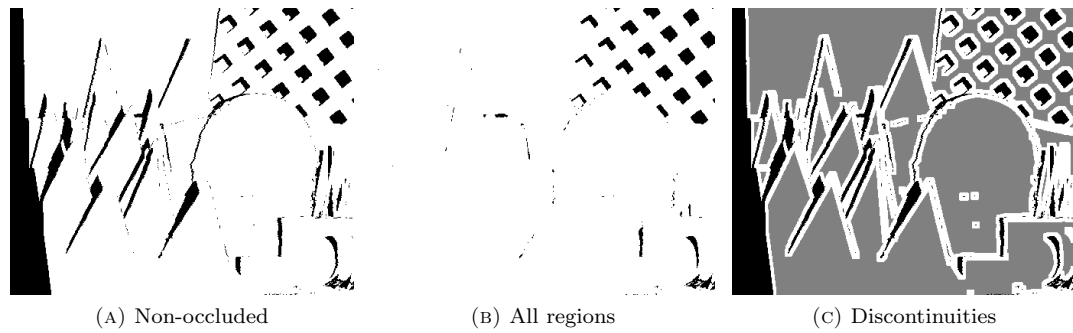


FIGURE 4.10: Masks used to evaluate the accuracy of the disparity map in reference to the ground truth. These are; non-occluded regions (nonocc), all known regions (all) and regions near depth discontinuities (disc)

The image pair used throughout this chapter for illustrative purposes is called *Cones* and is taken from the Middlebury Stereo Vision dataset², which provides a collection of rectified stereo images and other resources for experimental purposes [22]. New datasets are released periodically that contain a collection of left and right images with both disparity maps. Images used in this chapter are taken from the 2001 and 2003 datasets, which have resolutions of 348 x 252 and 450 x 375, respectively.

The Middlebury dataset specifies three standard measures to evaluate the accuracy of the disparity map (Figure 4.10). These take into account different features of a scene that commonly impact disparity estimation or mask areas that cannot be assessed:

Non-occluded (nonocc): Errors in the disparity map are only counted in non-occluded regions. The disparity of occluded regions cannot be accurately determined due to the fact that pixels in these areas only appear in one of the two images. A mask is placed over these regions to exclude any errors from the metric, shown in black in Figure 4.10a.

All (all): All regions are assessed, including occluded regions. This measure therefore benefits algorithms that are capable of inferring disparity within these regions. Note that the ground truth map does encode values for these regions, it is only stereo correspondence methods that cannot directly compute these areas. Regions where the disparity is not known in even the ground truth are still masked, shown as black in Figure 4.10b.

Discontinuities (disc): Only regions near depth discontinuities are considered. Occluded and unknown regions are masked (in black), as with nonocc, but in addition other regions away from discontinuities are also masked (grey). Only the white regions of Figure 4.10c are evaluated.

Table 4.1 shows that the algorithm is comparable with other works, in terms of pixel errors per frame, across each of the metrics presented. Pixel error numbers are calculated for the standard measures across four different image pairs called Tsukuba, Venus, Teddy and Cones, from the 2001 and 2003 Middlebury datasets. Tsukuba and Teddy are shown in Figure 4.11, where the disparity map from the algorithm is directly compared to the ground truth. The ground truth for these images was obtained using the structured light technique [145].

²<http://vision.middlebury.edu/stereo/data/>

TABLE 4.1: Comparison of the accuracy of related stereo matching algorithms using standard image pairs from the Middlebury database [22].

Author	Tsukuba			Venus			Teddy			Cones			Avg % bad pixel
	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	
Mei et al. [146]	1.07	1.48	5.73	0.09	0.25	1.15	4.10	6.22	10.9	2.42	7.25	6.95	3.97
Bleyer et al. [147]	2.09	2.33	9.31	0.21	0.39	2.62	2.99	8.16	9.62	2.47	7.80	7.11	4.59
Wang et al. [148]	2.39	3.27	8.87	0.38	0.89	1.92	6.08	12.1	15.4	2.12	7.74	6.19	5.61
Jin and Maruyama [149]	1.66	2.17	7.64	0.40	0.60	1.95	6.79	12.4	17.1	3.34	8.97	9.62	6.05
Ttofis et al. [76]	2.38	3.01	9.38	0.40	0.7	3.62	7.23	12.7	17.2	2.87	8.59	8.27	6.36
This work	3	4.48	9.1	1.5	2.54	6.41	6	9.8	12.7	4.2	8.5	8.72	6.41
Banz et al. [73]	4.1	-	-	2.7	-	-	11.4	-	-	8.4	-	-	6.7
Hirschmuller et al. [150]	3.26	3.96	12.8	1.00	1.57	11.3	6.02	12.2	16.3	3.06	9.75	8.90	7.50
Zhang et al. [81]	1.99	2.65	6.77	0.62	0.96	3.20	9.75	15.1	18.2	6.28	12.7	12.9	7.60
Shan et al. [151]	3.62	4.15	14.0	0.48	0.87	2.79	7.54	14.7	19.4	3.51	11.1	9.64	7.65
Zhang et al. [152]	3.84	4.34	14.2	1.20	1.68	5.62	7.17	12.6	17.4	5.41	11.0	13.9	8.20
Jin and Maruyama [153]	1.43	2.51	6.60	2.37	2.97	13.1	8.11	13.6	15.5	8.12	13.8	16.4	8.71
Jin et al. [154]	9.79	11.6	20.3	3.59	5.27	36.8	12.5	21.5	30.6	7.34	17.6	21.0	17.2
Shan et al. [155]	-	24.5	-	-	15.7	-	-	15.1	-	-	14.1	-	17.3
Chang et al. [156]	20.4	20.6	47.9	15.3	16.6	29.5	25.1	32.4	34.1	22.9	31.1	30.6	27.2

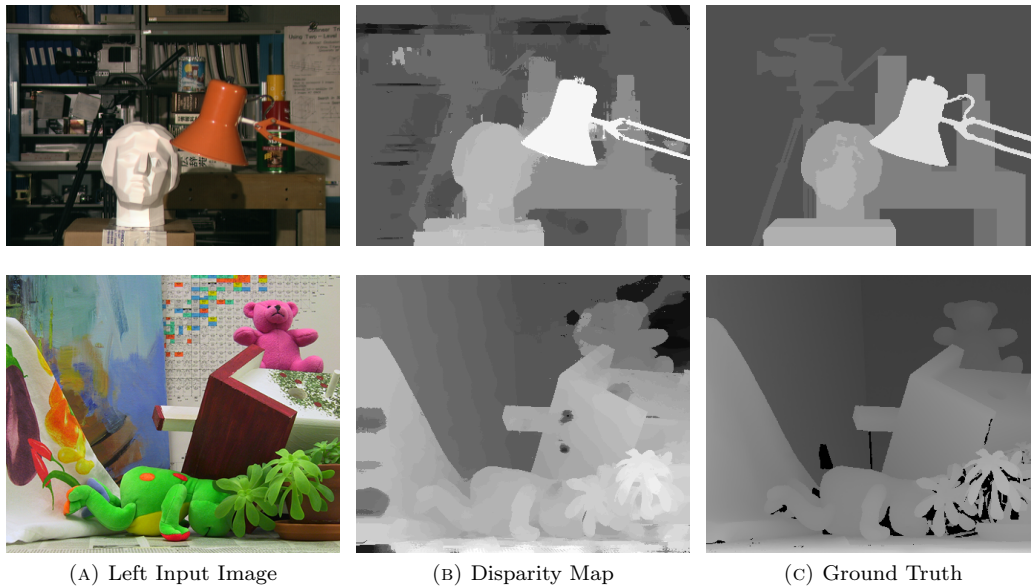


FIGURE 4.11: Comparison of the depth map output from our algorithm and the ground truth depth map provided with the dataset.

The following section presents the software implementation of the algorithm and the process of introducing multi-threading into the computationally-intensive stages.

4.2.3 Multi-threaded Software Implementation

This section describes how inherent parallelism opportunities in the structure of the DE algorithm are exploited. The intention of this process is to enable control of the degree of parallelism in order to modulate the processing of the computationally intensive parts of the algorithm and affect the execution time as a result.

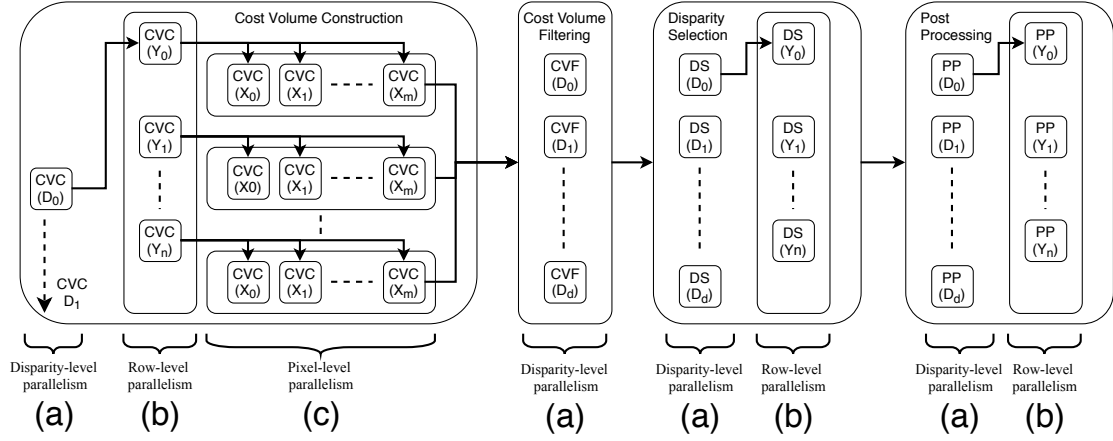


FIGURE 4.12: The dimensions of parallelism introduced to each stage of this implementation of the DE algorithm.

The central four stages of the DE algorithm (Equations 4.12 to 4.15) have the greatest parallelism opportunities, highlighted in Figure 4.5 from CVC to PP, due to the fact that they operate across the two dimensions of the image, or three dimensions with the addition of the disparity range. Parallel regions also have a low degree of data dependence. Figure 4.12 illustrates where this parallelism has been exploited, enabling the creation of a large number of independent threads. Parallelism is present across three dimensions in the CVC stage, one dimension in the CVF stage and two dimensions in the DS and PP stages. For all stages, the first level of parallelism is at the disparity level, partition (a) in Figure 4.12, where independent threads are created in the program for each disparity value. In CVC, DS and PP, additional parallelism is created within each disparity level to create per-row threads, (Y_0) to (Y_n) , shown vertically in partition (b) of Figure 4.12. For CVC, there is complete data independence therefore per-pixel threads can be created to introduce the final level of parallelism from (X_0) to (X_m) . n and m are the height and width of the input image, respectively.

Depending on the optimum level of multi-threading for a particular platform, the level of parallelism can be restricted to the row or disparity level. Furthermore, throttling of the number of active threads can be implemented at the disparity level if the optimum parallelism is less than the disparity range, *i.e.* $0 \leq par_{opt} \leq 64$. This process is achieved through a block threading approach illustrated in Listing 4.1 where blocks of threads are created, executed and are joined before the next block begins.

LISTING 4.1: Block threading approach to throttle parallelism at the disparity level.

```

1 int maxDis = 64, threads = 8
  Mat lImg, rImg, lGrdX, rGrdX, lcostVol[maxDis];
  pthread_t BCV_threads[maxDis];
  buildCV_TD buildCV_TD_Array[maxDis];
5
  for(int level = 0; level <= maxDis/threads; level++)
  {
      int block_size = (level < maxDis/threads) ? threads : (maxDis%threads);

10  for(int iter=0; iter < block_size; iter++)
      {
          int d = level*threads + iter;
          buildCV_TD_Array[d] = {&lImg, &rImg, &lGrdX, &rGrdX, d, &lcostVol[d]};
      }
  }

```

```

        pthread_create(&BCV_threads[d], &attr, CVC::buildCV_left_thread, (void *)&
15      buildCV_TD_Array[d]);
    }
    for(int iter=0; iter < block_size; iter++)
    {
        int d = level*threads + iter;
        pthread_join(BCV_threads[d], &status);
20    }
}

```

All stages of the algorithm have been implemented in C++ and parallelism has been introduced using the POSIX threads (pthreads) library, as in Listing 4.1. An extended software listing of the top-level module of the DE algorithm is provided in Appendix C. The full software program is available online in an open-source repository, as described in Section 1.3.1³.

The OpenCV library [157] has been used throughout this implementation of the DE algorithm to provide functions for several standard image processing algorithms and filters. This includes; greyscale conversion and the Sobel filter used in pre-processing (Section 4.2.1.1), box filtering operations performed in the CVF stage and standard image type or colour conversions used in multiple stages. The OpenCV core matrix class *Mat* is used throughout the algorithm as a container for input images, cost volume data, disparity maps and other intermediate data in each stage. The data allocation of *Mat* is determined by height and width variables and an array type specifier which encodes the basic data type (*e.g.* `char`, `int` or `float`) and number of channels in the image (*e.g.* three for colour images and one for greyscale). Standard ranges and definitions are used for array types, for example an RGB image has pixel values from 0 to 255 (an 8-bit unsigned character (8U)), and three channels (C3) therefore its type is `CV_8UC3`.

4.3 Experimental Characterisation of Performance and Power Consumption

To highlight the importance of system (application and platform) optimisation for highly-parallel, but not embarrassingly-parallel, algorithms, Figure 4.13 shows the power and performance operating space of the DE algorithm on the Xeon Phi platform presented in Section 2.6. Performance is measured as the FPS computed by the algorithm. Each point shows the performance and power consumption when executing the algorithm at each core count and frequency, sweeping active core number in intervals of four in the range 4 to 60 and frequency from 619 MHz through nine intervals to 1240 MHz. This gives a total search space of 135 points at this granularity, therefore this form of exhaustive search is not feasible at runtime and is made here to understand the power and performance scaling characteristics of the algorithm.

The labels attached to some of the points show the number of active cores used at those operating points. Figure 4.13 shows that there is a 46 W range in power consumption and an over 15 times range in performance that is attainable by operating at different frequency or core allocation points.

Six example operating points from Figure 4.13 are shown in Table 4.2. Traversing between these points, in a specific direction, can represent a power and/or performance optimisation

³Repository accessible at: <https://github.com/PRiME-project/PRiMEStereoMatch>

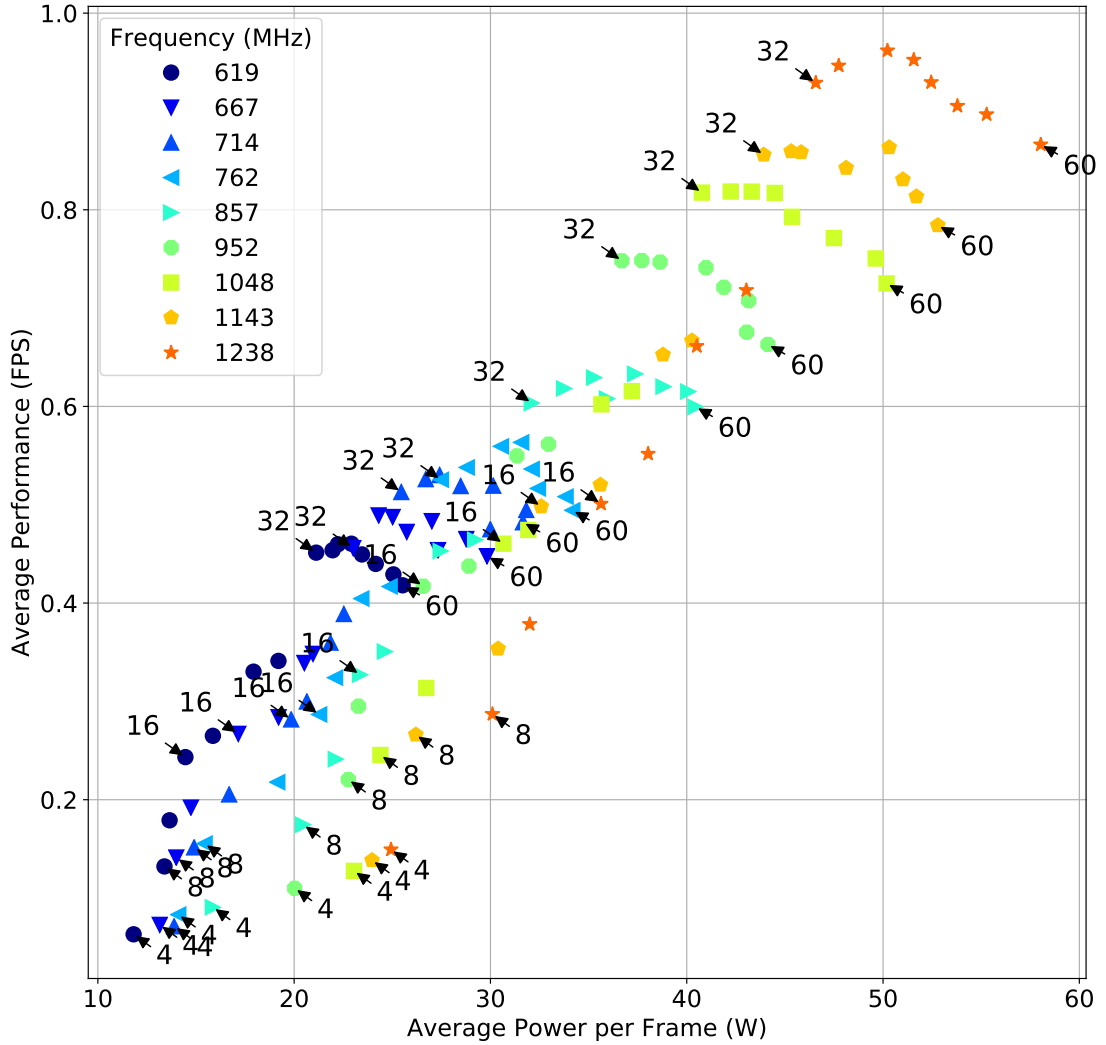


FIGURE 4.13: Power and performance operating points for the range of cores and frequencies of the Xeon Phi when executing the DE algorithm. The power and performance of each point is an average of four measurements, collected by cycling the algorithm with the image pair from Section 4.2.

or scaling. Moving from p1 to p2 is a power optimisation action as p2 can achieve similar performance (from 0.287 to 0.327 FPS) but with a 40.4% reduction in power consumption. This is illustrated in Figure 4.14a with the arrow indicating the direction of movement to achieve the optimisation. Similarly, moving from p3 to p4 is a performance optimisation action and can yield over 3x improvement in performance at approximately the same power consumption, shown in Figure 4.14b. Orthogonal to these two actions, a trade-off between power and performance produces a scaling action. This is demonstrated in Figure 4.14c between p5 and p6 where the movement is between a high power, high performance point and a low power, low performance point. This requires both frequency and core scaling to occur. Furthermore, for this application, using the maximum number of cores at maximum frequency (60 cores at 1240 MHz) does not yield the highest performance, yet it does consume the highest power. Therefore, DVFS alone is not enough to optimise power and performance, motivating the need for learning-based runtime management in this scenario.

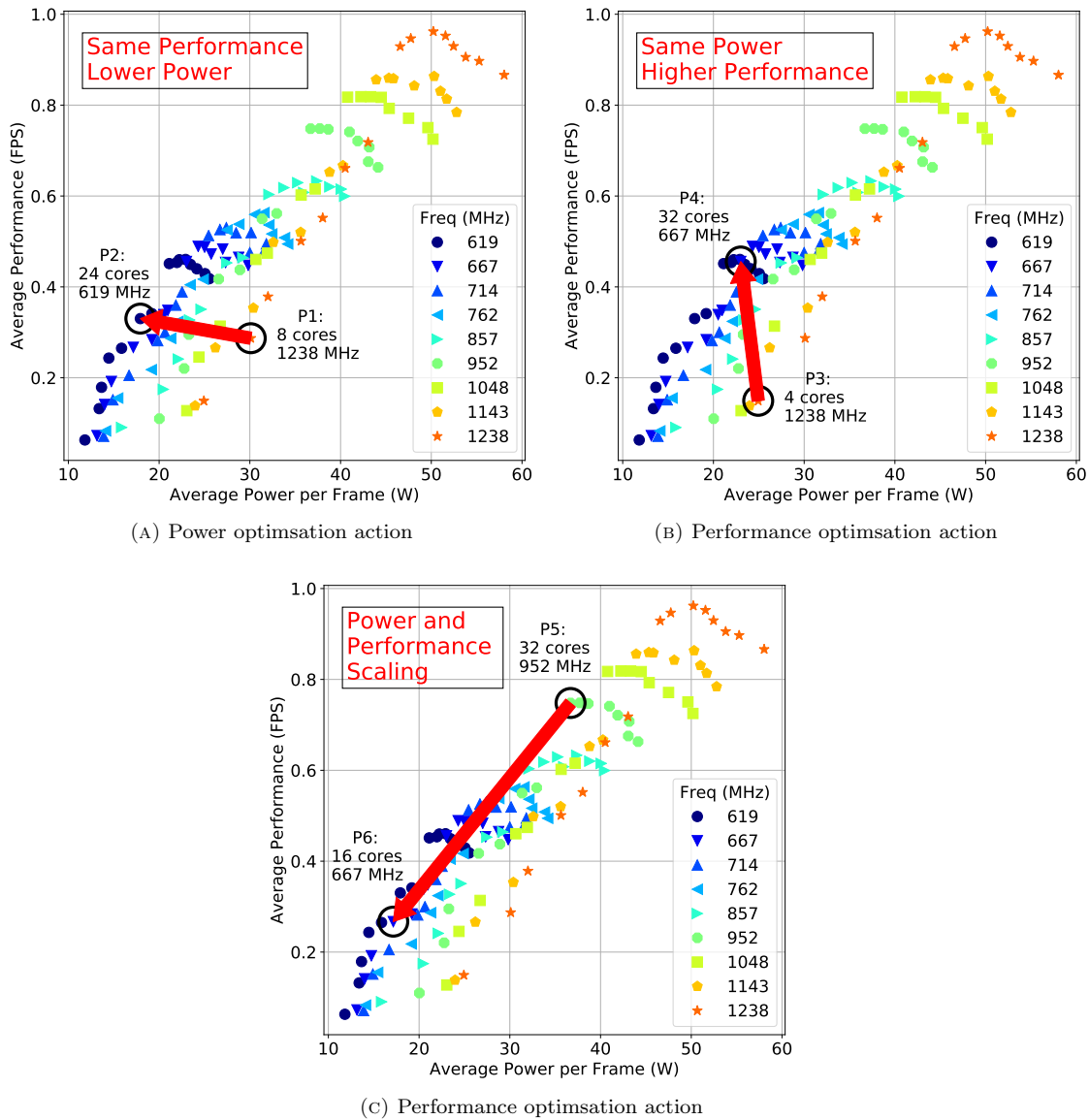


FIGURE 4.14: Optimisation and scaling actions made in the DE algorithm's operating space on the Xeon Phi platform.

TABLE 4.2: Normalised power and performance operating points. Traversing between these performs optimisation or scaling actions.

Operating Point	Frequency (MHz)	Cores	Performance (FPS)	Power (W)
p1	1240	8	0.287	30.1
p2	619	24	0.330	17.9
p3	1240	4	0.149	24.9
p4	667	32	0.456	23.0
p5	952	32	0.748	36.7
p6	619	16	0.267	17.2

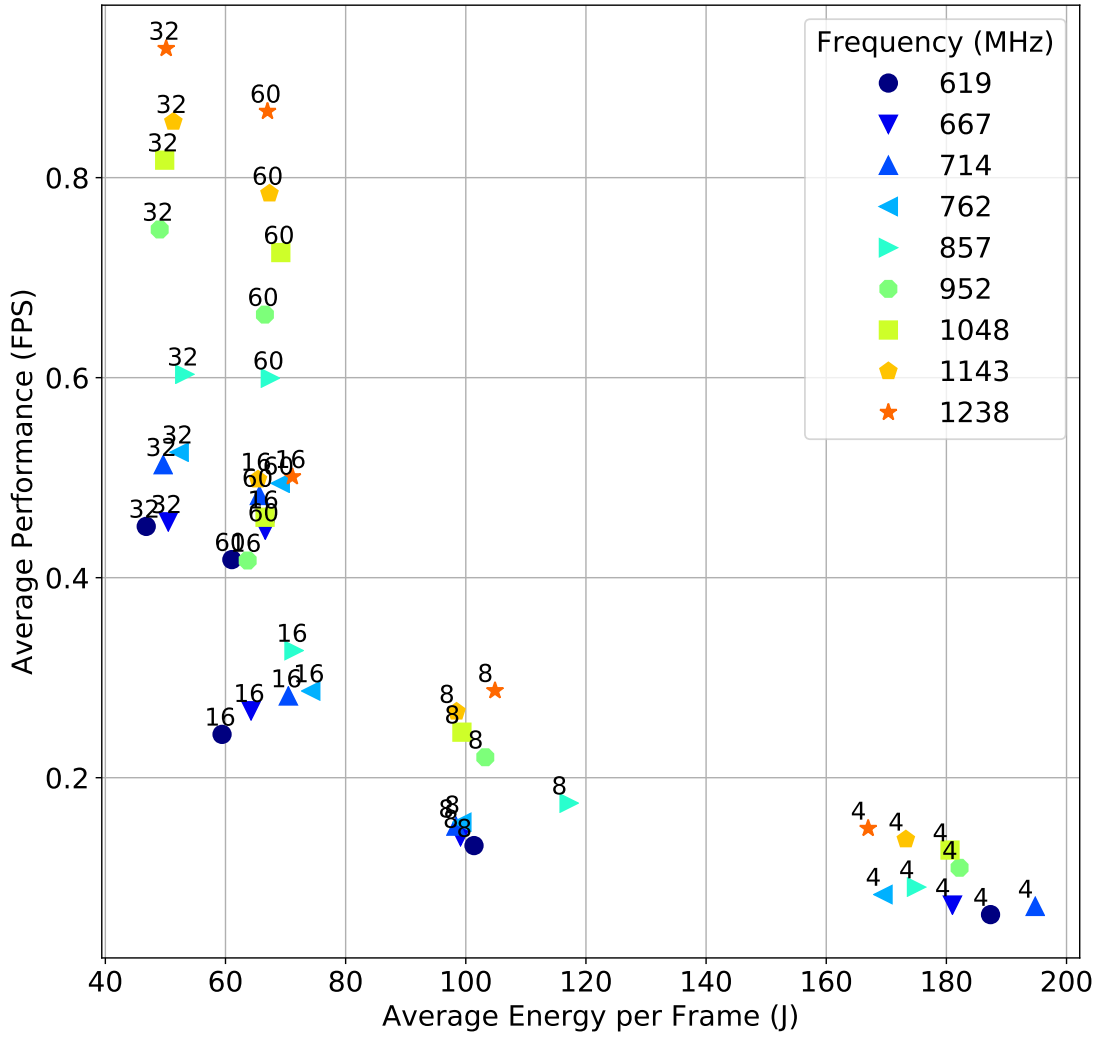


FIGURE 4.15: Energy and performance trade-offs for 2^x , ($2 \leq x \leq 5$) and 60 cores over the range of operating frequencies when executing the DE algorithm on the Xeon Phi.

Figure 4.15 shows the energy and performance operating space for the DE algorithm executing on the Xeon Phi. In the experiment, the number of cores was incremented by four and repeated for each operating frequency. Only the points at a 4, 8, 16, 32 and 60 cores are shown, to improve the clarity of the figure. The trend shows that scaling the number of cores will decrease the energy per frame cost of running the algorithm up to around 32 cores. However, the reduction in energy is not linear and after a certain point the energy per frame increases as the number of cores increases.

The points at 4 and 8 cores in Figure 4.15 are more heavily clustered than those at 16 and 32 cores, indicating that there is little variation in performance and energy at lower core numbers when the frequency is increased. Intuitively this is to be expected, as increasing the frequency for a greater number of active cores should lead to a larger increase in performance. The invariance between performance and frequency at small numbers of cores could be due to a performance bottleneck in other architectural components. In the context of the Xeon Phi, these bottlenecks may be the L2 cache size or the interconnect bandwidth. As the number of cores is increased, the load on a single cache or interconnect section is reduced because the data is distributed

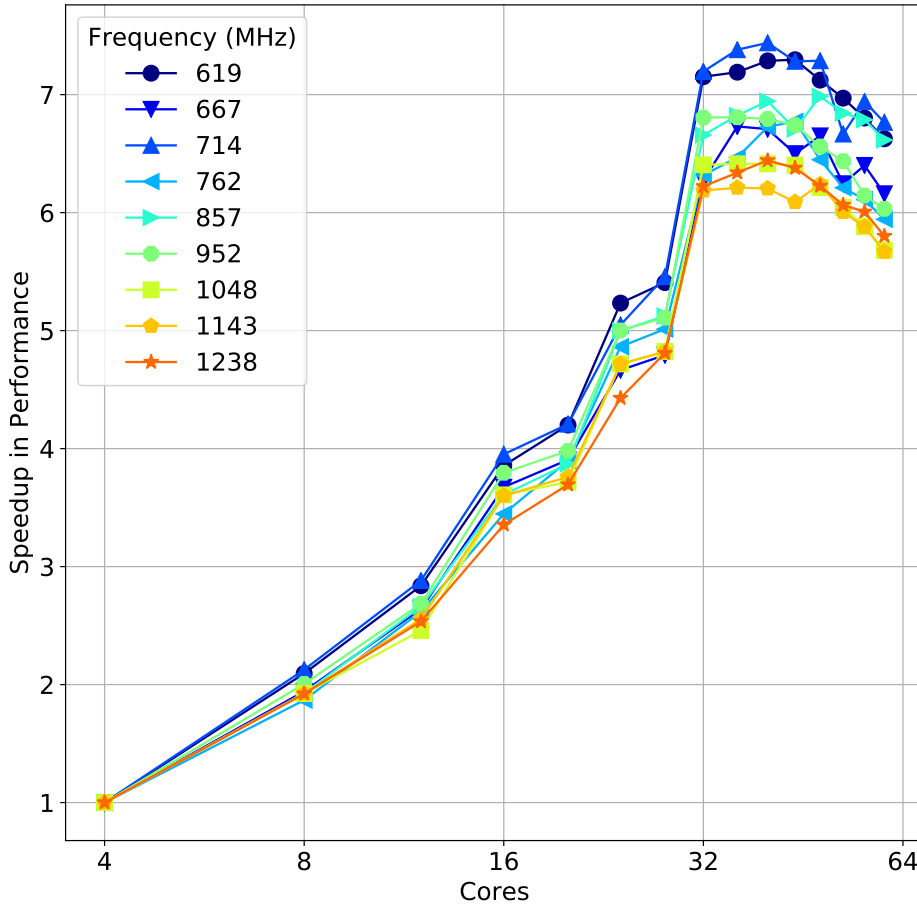


FIGURE 4.16: Speed-up in performance of the DE algorithm as a result of core scaling. Speed-up is in relation to the performance at four cores.

more widely amongst the cores. The system architecture is designed to operate most effectively at high levels of parallelism and although this has not been investigated, it can be conjectured from the architectural diagram of Figure 2.14a due to the fact that a ring-based interconnect has been used. For a different platform, such as the Odroid-XU3, the relationship between performance and energy will change. If performance scales linearly with frequency, then each cluster will appear more similar to the 32 and 60 core sets in Figure 4.15, irrespective of the number of cores utilised. In this case, performance increases but energy remains largely constant due to the increase in power consumption. If the increase in power consumption is higher than the increase in performance, the average energy consumption will increase and Figure 4.15 will appear similar to Figure 4.13.

Figure 4.13 and 4.15 have illustrated that controlling the number of active cores is the most significant way to scale power and performance. The cumulative amount of performance speed-up in the DE algorithm, as the number of active cores is incremented by four, is displayed in Figure 4.16. This trend is approximately linear up to 32 cores with the tight grouping of the different frequencies indicating that it has only a minor effect on performance.

The next section introduces the integration of the runtime management approach from Chapter 3 with the DE algorithm in order to control both frequency and the number of cores at runtime to meet a target performance set by the DE algorithm.

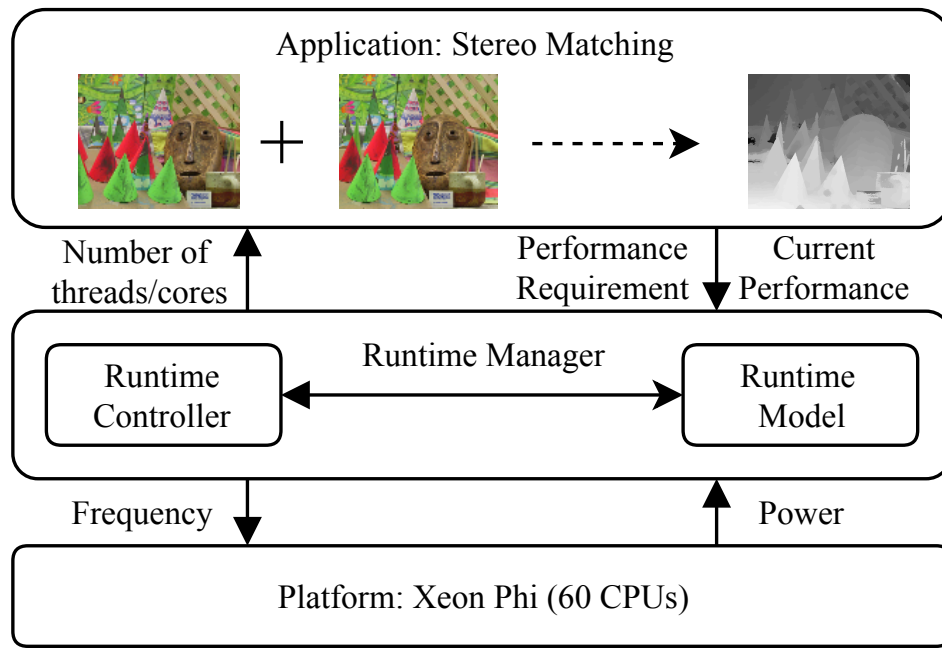


FIGURE 4.17: Block diagram of the stereo matching run-time optimisation approach

4.4 Runtime Management of Stereo Matching

Figure 4.17 shows a block diagram for the runtime management approach in the context of the DE algorithm, illustrating the data and control interactions between the algorithm, RTM and the hardware platform. In this context, the algorithm is packaged with the necessary measurement and control software that is required for runtime management, therefore it is referred to as the application, as shown in the top section of the figure. The RTM uses the time taken to compute the disparity map from input images as the performance metric. The performance of the application is communicated to the runtime manager after each frame is computed. The performance requirement is also specified to the RTM but does not necessarily change between frames. The hardware platform used for this experiment remains the Xeon Phi as it supports measurement, by the RTM, of the instantaneous power consumption of the device through the operating system. Power is read at regular intervals by the RTM in order to give a more accurate estimate of the average power consumption and the energy per frame of the application through integration of each measurement.

The runtime adaptation of application multi-threading, to set the active core number, and frequency is provided by the runtime controller back to the application and device. The RTM controls multi-threading that sets the disparity-level parallelism in the DE algorithm, as per the implementation shown in Figure 4.12. This determines the number of disparity levels that are simultaneously calculated in each thread block. The maximum threading level for this application is determined by $maxDis$, which is 64 for the chosen dataset.

The runtime manager consists of the two components presented in Chapter 3; the runtime model and runtime controller. The runtime model is sub-divided into the power model and the performance model, both derived using MLR. The runtime controller encapsulates the gradient

descent algorithm that performs optimisation at runtime and all the necessary code to determine the frequency of the platform and the multi-threading of the application.

During an initial training period, at the beginning of application execution, the model is built using runtime measurements of the application and platform by testing different frequency settings and threading levels, as shown in the flowchart of Figure 3.1. Once the minimum number of samples have been acquired (determined to be 16 in Section 3.3 for two independent variables), the model is derived using an OLS regression method. The model then guides the runtime controller in optimising the application's parallelism and the platform frequency controls to meet the application-specified performance target.

Power optimisation is a secondly target of the RTM therefore the first operating point whose performance meets the application requirement is used. Should the performance requirement drop, the controller will inspect the model to find a lower power operating point that meets the new requirement. An example of this type of optimisation is shown in Figure 4.14c as a scaling action.

The next section presents experimental results of the runtime adaptation of the DE algorithm and Xeon Phi under a simulated dynamic application requirement.

4.5 Experimental Results of Online Adaptation

In Section 4.3, the complete power and performance operating space of the application was characterised in order to demonstrate that power and performance scaling was possible for a multi-threaded application and multi-core platform. This operating space, and the scaling of power and performance motivated the need for runtime management to optimise the application and platform in response to dynamic application requirements. This section examines the effectiveness of the proposed runtime management approach in the adaptation of the DE algorithm when exposed to these dynamic requirements.

Once runtime modelling of the application's power and performance operating space is completed, the RTM can adapt to changes in the performance target through selection of the most energy-efficient threading level and DVFS settings. This process is demonstrated in the three time series graphs of Figure 4.18. Each graph is examined in detail, to assess the effectiveness of the RTM's optimisation, in the remainder of this section. Furthermore, a comparison is made to the Linux frequency governor called Ondemand, which uses the CFS scheduler. The behaviour of both of these components is described in Section 2.2. The results for these experiments are shown as dashed lines.

Changes in the performance target of an application could be due to a multitude of factors depending on the particular application. In the case of the DE algorithm, a higher performance may be required to enable the system to calculate the depth of objects in the scene that are moving at a higher speed, such as objects closer to the camera. Adaptation to changes in the performance target can be seen in the top series where the measured performance tracks the target as it changes over time. The average absolute error in measured performance is 5.56%. Excluding occasions when the measured performance exceeds the target performance, which is

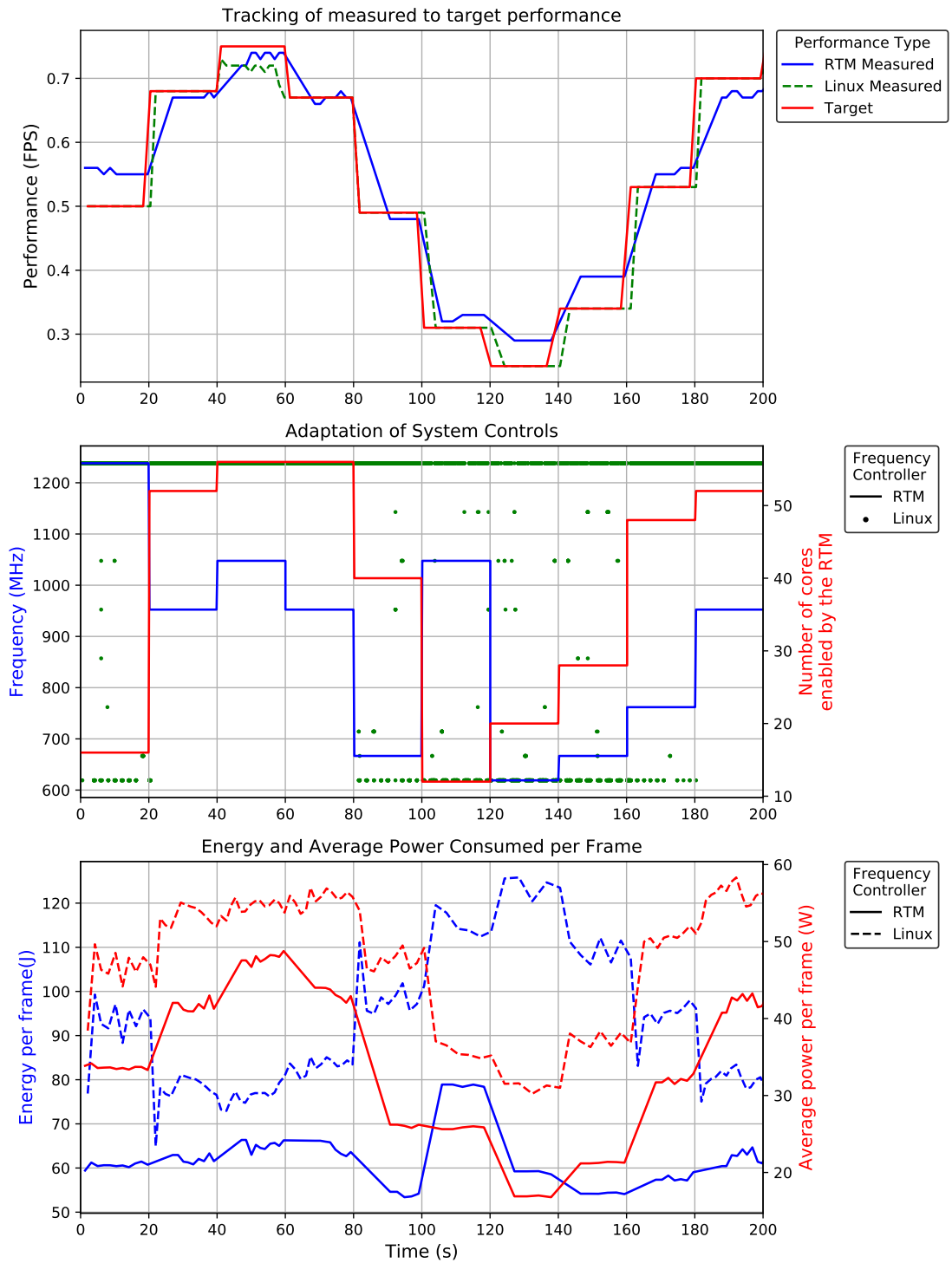


FIGURE 4.18: Time series analysis of the RTM performing online adaptations of core number and frequency to optimise power and energy whilst meeting a target performance.

TABLE 4.3: Optimised operating points, sampled from Figure 4.18, that minimised power consumption under each performance target.

Time (s)	% of Target Performance	Cores	Frequency (MHz)	Energy (J)	Average Power (W)
6.82	112	16	1240	60.6	33.6
31.4	98.5	52	952	61.3	41.0
51.3	98.7	56	1050	65.3	48.1
74.7	100	56	952	64.1	43.1
94.8	98.0	40	667	53.4	26.1
112	106	12	1050	78.4	25.9
131	116	20	619	59.3	16.9
154	115	28	667	54.4	21.4
172	104	48	762	58.3	32.3
195	95.7	52	952	63.0	42.3

not considered a penalty, this error drops to 1.16%. Performance of the application under Linux Ondemand is manually matched to the target to allow the governor to follow its DVFS policy by changing the frequency once frame processing is complete and the CPU utilisation drops. As a result, the measured performance tracks the target very closely. The Ondemand governor cannot restrict the number of cores used by the application therefore it always uses all 60.

The online adaptation of the number of active cores and the frequency is shown in the second series of Figure 4.18. The RTM interrogates the power/performance model in order to predict which operating point will give the target performance with the lowest power consumption. It can be observed that changes in core number and frequency occur independently of each other and are only loosely correlated. This is due to the overlapping of operating points, in terms of power and performance, which use a high number of cores at a low frequency with those which use a lower number of cores at a higher frequency. For example, using data from Table 4.3, at 112 seconds a target of 0.310 FPS leads to 12 cores at 1050 MHz whereas at 154 seconds, the similar target of 0.340 select a very different operating point at 28 cores at 667 MHz. This shows how the RTM model captures the power and performance trade-offs between DVFS and core scaling in an application. Moreover, there is a stronger trend in the hypothesis that an increased performance target will result in both a higher frequency and number of cores, as can be seen from 120 seconds onwards.

The final series shows the measured average power and energy consumed per frame as the performance target fluctuates, for both the RTM and the Ondemand governor. To achieve the same performance, the RTM can reduce average power consumption by 27.8% and increase energy efficiency by 30.0% and consistently gives lower power consumption and increased energy efficiency across the entire range of performance targets. This is primarily due to the core scaling ability of the RTM and its rigorous control of frequency.

4.6 Summary

This chapter has investigated the implementation of a parallel disparity estimation algorithm on a multi-core platform and presented results of scalability experiments in the application and

platform operating space. A performance and power trade-off has been reported, demonstrating that it is possible to achieve the same performance with lower power consumption and higher energy efficiency by optimising frequency and core allocation.

The contributions from this chapter include:

- a multi-threaded, dynamically tunable implementation of a stereo matching algorithm called disparity estimation;
- the runtime power and performance modelling of the algorithm;
- the runtime management and optimisation of the DE algorithm on a multi-core platform in response to changing application performance requirements;
- the evaluation of this approach with comparison to standard linux DPM techniques.

The code used for characterisation of the DE algorithm, as well as code used for the runtime management of it, can be found in Appendix C. Further development of the application has occurred since the experiments described in this chapter were conducted. The application has been made publicly available at <https://github.com/PRiME-project/PRiMEStereoMatch> where the most up-to-date implementation can be found.

The chapter has shown that an RTM approach could be applied to specific application and platform. However, the RTM approach can be applied to other applications and platforms, after the necessary modifications. In addition, direct comparison between the RTM approach and other runtime management techniques presented in the literature was not possible because of the specific multi-dimensional optimisation space modelled by the RTM and the specific measurement and control processes for the application and platform.

In order to enable experimentation with different RTM algorithms and the optimisation of many applications and platforms, a standard framework and interface between each of these components is required. The following chapter presents such a cross-layer framework and API for runtime management that is designed to enable more comprehensive comparison and evaluation of RTM approaches using a common system model and standardised cross-layer connections. The framework is also designed to make tunable applications, such as stereo matching, agnostic to the platform and RTM. This is a significant step towards enabling the runtime management of many applications and platforms through a common methodology.

Chapter 5

A Framework for Application- and Platform-agnostic Runtime Management

Application requirements drive innovations in hardware and software. An example of hardware innovation that has been presented is heterogeneous processor architectures in modern embedded platforms, which provide both high-performance and energy-efficient execution of applications. In addition, DPM techniques have become essential in ensuring that a device remains operation for a significant length of time. Power and temperature sensors are embedded in platforms to measure consumption in real time, providing greater insight into the behaviour and health of the platform. One result of these innovation is that the management and control of heterogeneous systems at runtime has become a non-trivial process.

Innovation in software, including programming languages and frameworks such as OpenCL, are developed because of the need to increase the programmability of parallel and heterogeneous architectures. In addition, they allow greater control over the execution of applications. To extract the maximum potential from these platforms, applications have become increasingly tunable and heterogeneous, with adjustable parameters that must be controlled to optimise their behaviour. A greater number of application domains are expanding into embedded systems, including those which exhibit very dynamic behaviour.

These two challenges, from hardware and software innovations, motivate the need for a runtime management approach that is cross-platform and generic in its support of applications. However, many current runtime algorithms do not support the tuning of application parameters or are tightly coupled to specific hardware platforms and/or applications. This limits the cross-application and cross-platform implementation of these techniques, preventing widespread adoption and direct comparison.

To address these challenges, this chapter presents the following contributions;

- specification of a novel framework to support application- and platform-agnostic runtime management and allow the control of both software applications and hardware platforms simultaneously via dynamic *knobs* and *monitors*.
- an increase in the mobility of applications, runtime management software and hardware platforms by separating them into three layers.
- linkage of the layers of embedded systems through a cross-layer API to standardise the runtime management of software applications executing on homogeneous and heterogeneous platforms.
- experimental demonstration of the framework across two heterogeneous platforms to show cross-platform support using the Odroid-XU3 and the Altera Cyclone V.
- presentation of trade-offs between power, performance and accuracy are presented in three application-platform scenarios, highlighting how this operating space can be traversed at runtime.
- demonstration of two state-of-the-art runtime management approaches that reduce the energy consumption of application execution by 18.2% and 17.2%, the first based on reinforcement learning and the second on PMCs.
- release of an open-source implementation of the framework and API in C++, available online at: https://github.com/PRiME-project/PRiME_Framework.

Material from this chapter has also been submitted for publication in the IEEE TCAD journal as [158]. The framework is a collaborative work within the PRiME project with researchers from within the project at both the University of Southampton and Imperial College London. As a result, some of the contributions stated were made jointly as part of the team of researchers involved but are necessary to present the complete picture of the framework. In particular, the contributions to the framework made by the author were;

- co-definition of the Framework's API;
- beta C++ implementation of the Framework's API functions;
- development of the UI and Logger modules;
- specification of the JSON protocol for communication; between the experimentation tools and the system layers;
- development of the pthread functions for the Jacobi application;
- development of the Profiler RTM.

The chapter is composed of the following sections. In Section 5.1, the fundamental concepts behind the application- and platform-agnostic framework are outlined, including the layered system model, *knob* and *monitor* constructs and the properties of both that enable agnostic runtime management. The specification of the framework's API is presented in Section 5.2, highlighting how cross-layer interactions are preformed. Aspects of the framework's software implementation are presented in Section 5.3, with focus on how the framework is developed

programmatically to preserve application- and platform-agnosticism. Section 5.4 outlines the experimental methodology used to validate the operation of the framework, including technical details of the applications, RTMs and platforms that are demonstrated in Sections 5.4.1, 5.4.2 and 5.4.3. A collection of tools that have been developed to aid experimentation with the framework are presented in 5.5. These were used to both conduct the experiments presented in the following section and analyse the data from them. Experimental results from the framework are presented in Section 5.6, including cross-layer trade-offs from application profiling and the demonstration of runtime management layer validation.

5.1 Fundamental Concepts of the Cross-layer Framework

A drawback with many existing runtime management algorithms is that they are usually designed for only specific classes of application or they have only been validated with specific benchmarks, which can lead to the over-fitting of result that are not generalisable or transferable to new applications. Similarly, the runtime algorithm may only be designed to manage specific types of architectural configurations, or the experimentation has been conducted only on specific hardware platforms. Together, these factors limit the potential of these approaches. The provision of an application- and platform-agnostic framework for runtime management algorithms will alleviate many barriers to the wider development and testing of these runtime algorithms. The proposed cross-layer framework is an application-agnostic, cross-platform approach to runtime management. In this section, the key concepts of the framework are presented that have been identified to achieve these properties.

5.1.1 System Layers

The framework introduces a defined system model that enforces the separation of systems into three distinct layers, as illustrated in Figure 5.1. The content and scope of each layer is discussed in this section.

The application layer comprises any number of software programs that present a significant workload to the system. The stereo matching algorithm presented in Chapter 4 is an example of such an application. The model of the application layer supports the concurrent interaction of multiple applications with the runtime management layer. The App N_A entities in this layer signify independently controlled applications. However, there is no restriction on related applications or processes from having data dependencies between them whilst also interacting separately with the runtime management layer. There is no hierarchy in the model of applications within the framework and each is assumed to interact from its highest level of control.

The device layer encapsulates the physical hardware of the platform and its software drivers. The platforms presented in Section 2.6, such as the Odroid-XU3, are valid examples for the device layer. The device objects within the layer translate to the processing elements contained within the platform, such as a multi-core CPU, a GPU or and FPGA. The definition extends to any processing resource which is contained within the same software image and therefore accessible from a single OS perspective. This may include discrete SoC on a Single Board Computer (SBC) or certain configurations of networked platforms. However, the model of the device layer does not

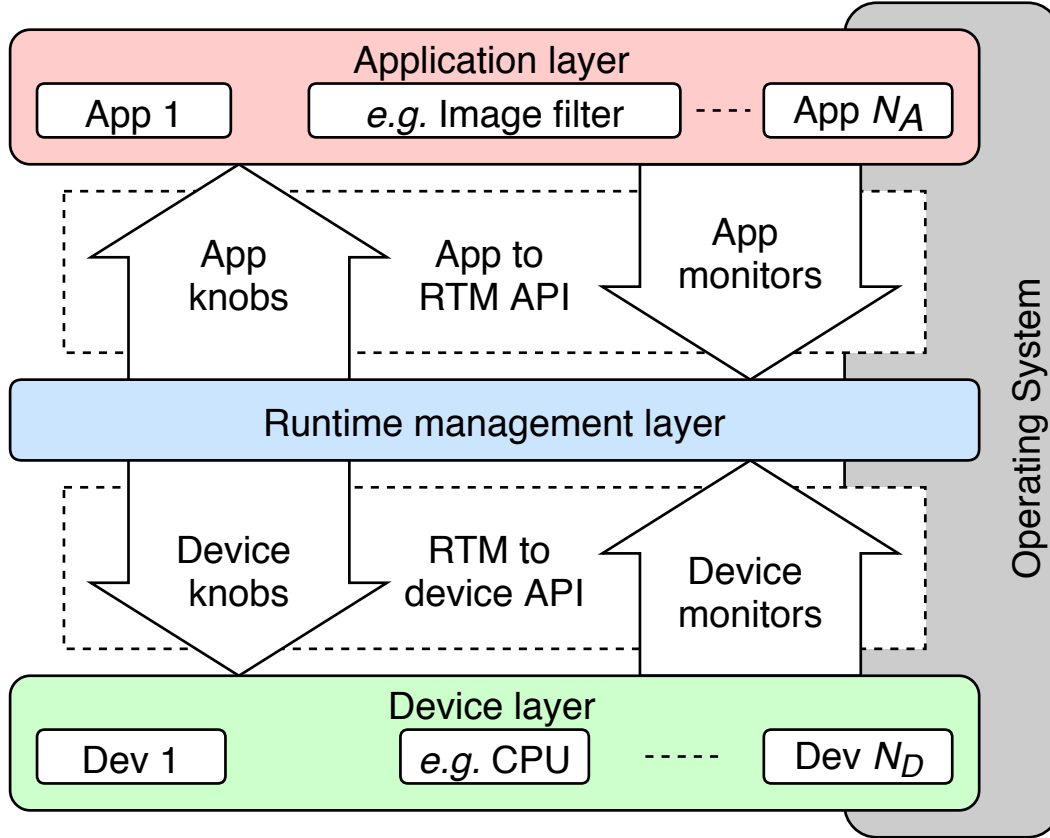


FIGURE 5.1: Cross-layer framework and APIs enabling communication between the application, runtime management and device layers using knobs and monitors. The operating system spans all three layers and hosts the execution of each.

support the concurrent operation of multiple separate platforms interfacing separately through the API to the runtime layer. This is due to some aspects of the API that is presented in the following sections.

The runtime management layer contains a software process, the RTM, which is responsible for the control of both applications and the device in order to meet application-specified performance requirements while maximising the energy efficiency of the platform upon which they execute. The RTM presented in Chapter 3 is an example of a suitable approach that could be used in this layer as it is capable of the control and optimisation of particular aspects of the application and platform. It is envisaged that any approach from the literature could be integrated to operate in this layer.

By enforcing this separation in the layers of the framework, portability and cross-compatibility are ensured. Applications and device drivers need only be written once to be used with any runtime management approach implemented within the framework. In addition, runtime management algorithms can be designed for a framework-style context, where the agnostic view of the system must be observed. The common mechanism also motivates the potential to change the active runtime algorithm in the layer to suit the optimisation problem presented. Each layer retains direct access to the operating system to enable the independent creation and execution of threads by the application and the execution of device driver. This arrangement also avoids the overhead of additional communication through the device layer by the application and RTM

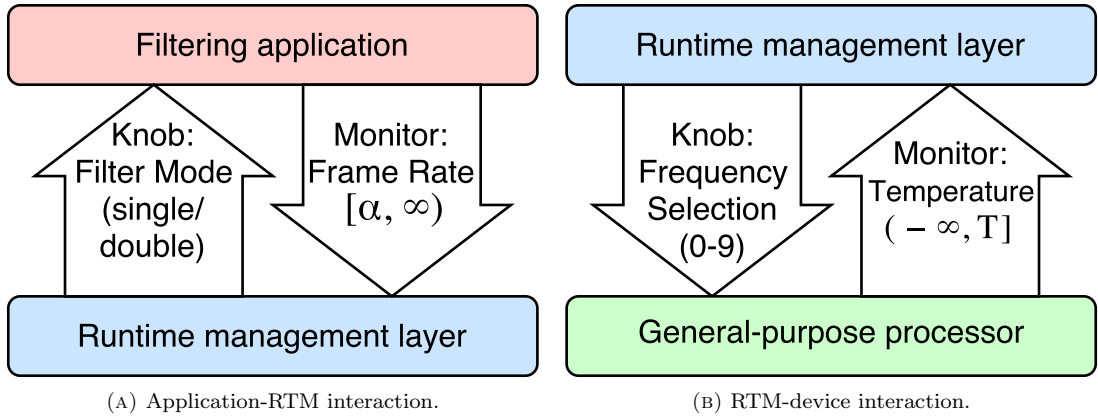


FIGURE 5.2: Cross-platform and application-agnostic knob and monitor examples used within the framework.

layers. This reduces the cost of running the RTM and enables the parallel execution of each layer.

5.1.2 Cross-layer Connections

Key to the operation of the framework are *knobs* and *monitors* that form cross-layer connections and facilitate the flow of information between the application and device layers to the runtime management layer. They are shown in Figure 5.1 as arrows between the system layers, which also indicates the direction that information is transferred in. These constructs are coupled with two cross-layer APIs that are used by the processes within each layer to update their values and settings. Knobs are framework constructs created by the application developer that connect to runtime-tunable parameters in the application, allowing the RTM to adjust them through the cross-layer API. Knobs allow the behaviour of the application to be changed in order to meet objectives. For example, tuning of thread-level parallelism in the DE algorithm (presented in Chapter 4) allows the performance requirement to be met with power optimised as well. Orthogonally, application monitors are similar constructs that allow metrics or properties of significance to an application to be observed by the runtime management layer. Monitors provide the RTM with a picture of what is important to an application and how well the RTM is achieving its requirements.

An example application monitor would be the performance or frame rate of the DE algorithm. The application could also provide the target performance to the RTM through the same monitor. The extensibility of this concept allows it to be applied to any metric, therefore monitors could be used to convey the various quantitative measures of accuracy described in Section 4.2.2. Consider the filtering application shown in Figure 5.2a, whose precision can be varied at runtime, with resultant performance (expressed as a frame rate) updated during execution. If the application exposed a precision knob and a throughput monitor, the RTM would be able to establish the effect of adjusting the application's precision, via the knob, on its performance through the monitor.

Between the device layer and the runtime management layer, knobs facilitate the tuning of runtime-adjustable parameters in the hardware platform, such as voltage or frequency. Similarly, device monitors allow hardware properties, such as power or temperature, to be read (on platforms that have the hardware to report such quantities). An example of frequency selection and subsequent temperature observation is shown in Figure 5.2b. To use another example from Chapter 4, the measurement of power on the Xeon Phi would be contained within the device layer and reported to the RTM via a device monitor. Knobs and monitors are presented in a standardised, unitless format and it is only their values that are passed between layers. Any significance attributed by the application or device that owns the knob or monitor remains within the scope of that layer. As a result, the RTM operates in a purely mathematical environment.

The framework can be viewed hierarchically ‘downwards’ since, as far as knob and monitor control is performed, applications are masters of the RTM, making calls to the API and controlling the presence and configuration of each knob and monitor. This arrangement also means that the application controls when it gets and applies knob values in its code. The RTM can assert that a particular knob value should be applied, for example, in order for the application to meet a performance target, however it cannot enforce the setting. The application must request an update on the knob in order to know whether its value has been updated.

Conversely, the device acts as a slave to the RTM since it must respond to requests made by the RTM to set a knob value or request an updated monitor value. Thus, applications ‘pull’ their knob settings from the RTM and ‘push’ monitor updates, while device knobs are pushed from the RTM and monitor values pulled.

5.1.3 The Optimisation of Targets and Constraints

The flexible combination of cross-layer knobs and monitors provides a powerful mechanism to enable the optimisation of multiple objectives across applications and devices in an agnostic manner. To direct the runtime management process, applications and devices specify bounds in the form of *minima* and *maxima*. For knobs, the bounds represent a range of *allowed* values. For example, the available precisions of the filter application shown in Figure 5.2a would be represented by the indices 0 for single- and 1 for double-precision and application knob $p \in \{0, 1\}$ controls their selection. Device knobs are similarly bounded, *e.g.* the frequency knob f , shown in Figure 5.2b, must be set such that $f \in \{0, 1, \dots, 9\}$. The same conversion process can be applied to frequency control on the Xeon Phi where the frequency intervals translate as: $\{619, 667, 714, \dots, 1240\} \text{ MHz} \rightarrow \{0, 1, 2, \dots, 8\}$. The MLR RTM would then operate this knob rather than directly setting the frequency and apply the appropriate scaling to its linear models.

For monitors, the bounds represent a range of *desired* values for application monitors and platform physical constraints for device monitors. The RTM’s primary objective is to ensure that the actual values of all application monitors remain within the specified bounds. However, these ranges are *desired* because they are soft limits that the RTM may exceed if; the bounds cannot be met, another monitor has a higher weighting (covered in the next subsection) or training is taking place, as with the MLR RTM. If a hard limit is required, the monitor bounds could represent a guard-band from this point to a soft limit. Beyond this, it is free to optimise any unbounded monitors, in either the application or device layer, in order to meet any secondary

objectives, for example to reduce power consumption. Using the example shown in Figure 5.2a, if a minimum frame rate α is required, it is indicated to the RTM with an application monitor bounded by $[\alpha, \infty)$. Similarly, the Thermal Design Power (TDP) of a SoC could be specified as the maximum bound T on a monitor that measures the temperature of the device, represented as the range $(-\infty, T]$.

5.1.4 Monitor and Application Weighting

It is possible that applications have multiple objectives that characterise their operation, and that these may have differing priorities. In addition, there may be a greater penalty if a particular objective is not met in comparison to others. Therefore, each application monitor is coupled with a numeric weight in order to signify the relative importance of it to other monitors within the same application. For example, an application that is aware of both its throughput and accuracy may wish to prioritise the optimisation of one of these metrics over the other. As a result, a weight is attached to each monitor.

In response to monitor weighting, the RTM is guided to expend an amount of *effort* that is proportional to the ratio of the weights provided in the optimisation of the monitors' values. This effort is analogous to the amount of computation devoted to optimising each monitor value or the effort may be reflected in the location of the true monitor value within its bounds. The scaling and trade-off between power and performance demonstrated in Section 4.3 of Chapter 4 are examples of processes that the RTM could perform in response to the weighting of monitors. After all bounds are met, the RTM would scale performance further if its monitor provided a higher weighting than the power monitor.

In a multi-application scenario, where a user is currently interacting with a single application, whilst others run in the background. Due to the higher impact on the QoE of the user should its performance degrade, this foreground application could be afforded greater priority by the RTM in the task process of meeting this application's monitor bounds. This priority is encoded an additional weight attached to each application, which applies evenly to all its monitors. This weighting is dynamic to reflect the changing requirements of each application.

Inter-application weighting add an additional dimension to the optimisation process for the RTM. However, it can be considered as just a hierarchical extension of the existing multi-objective optimisation process performed by the RTM within each application. The product of the application weighting and the monitor weighing will provide a single weighting vector across the entire vector of application-level monitors.

5.1.5 Knob and Monitor Types

Each knob and monitor is augmented with a *type* selectable from a discrete set of options. This field represents a compromise between complete agnosticism, where no information about knobs and monitors is afforded to the RTM, and the full provision of information to the RTM. This is worthwhile since it prevents these conclusions from having to be learnt empirically and so 'hints' are provided to the RTM about the function and consequence of adjusting a particular knob or monitor. However, there are relatively few device parameters that can be controlled or

TABLE 5.1: Structure of application and device knobs in the cross-layer framework.

Application	Device	Description
App ID		Unique ID of the owner application of the knob
ID	ID	ID of the knob, unique to each application or the device
Min	Min	Minimum possible setting of the knob
Max	Max	Maximum possible setting of the knob
Value	Value	Currently set value of the knob
	Init	Initial value for the device knob
Type	Type	Type of the knob

monitored, such as voltage, frequency, power, *etc.*, and so the overhead of providing and maintaining these lists of types carries relatively little overhead. Moreover, this encodes information about which direction optimisation should be performed in for each monitor, *e.g.* “lower power is better”.

In addition to types are the provision of discrete- and continuous-valued versions of each knob and monitor construct. This allows applications and the device to specify the nature of each knob and monitor, to aid the RTM’s optimisation process, as it affects the number of possible values in each range. For their implementation, discrete versions use signed integer values while their continuous counterparts operate using floating-point data.

5.1.6 Runtime Adaptability

Finally, to afford application developers maximal flexibility, all bounds and weights are considered to be adjustable at runtime, and there are also no restrictions on when registration and deregistration events of knobs, monitors and applications, can occur. Generally, it is envisaged that applications will register their knobs and monitors on start-up and deregister them prior to termination, although there should be no limitation to such events occurring partway through application execution. As a result the RTM should implement a management policy that is adaptive enough to account for the registration and deregistration of knobs and monitors at runtime. This model of application behaviour is one that matches most closely to real-world behaviour seen on mobile and embedded systems therefore allowing such flexibility is a key requirement.

5.2 Framework Specification

This section presents a methodology for delivering the concepts presented in the previous section as a complete framework. The realisation of the knob and monitor concept is presented first, followed by a full specification of the cross-layer APIs that support them and facilitate the communication of information between the system layers.

TABLE 5.2: Structure of application and device monitors in the cross-layer framework.

Application	Device	Description
App ID		Unique ID of the owner application of the monitor
ID	ID	ID of the monitor, unique to each application or the device
Min	Min	Minimum desired value of the monitor
Max	Max	Maximum desired value of the monitor
Value	Value	Current value of the monitor
Weight		Weight of the application monitor
Type	Type	Type of the monitor

5.2.1 Knobs and Monitors

Table 5.1 shows the structure of application and device knobs in the cross-layer framework, outlining the parameters that are contained within each. There are slight differences between application knobs and their device equivalents to reflect the subtleties of the system model. The *App ID* field enables application knobs to be associated by the RTM to their owning application and therefore potentially optimised as a group. The *ID* field exists for knobs in both layers, again to uniquely identify them from other knobs. An application knob can only be uniquely identified from both its *ID* and *App ID*. The *min* and *max* fields enable knobs to specify the bounds of their possible range of settings. As will be shown in Section 5.3, whether this range is discrete or continuous is determined by defining two separate C structs for the knob object, therefore a field is not required to convey this information. The current value is held in the *value* field with an initial field provided for device knobs to allow default device settings to be preserved. If required, an application can indicate its initial value by setting it as the first number in the value field. Lastly, the *type* field is available for knobs to provide some additional information to the RTM on its function.

Table 5.2 shows the same structures but for application and device monitors. The ID fields work in the same way as for knobs but the *min* and *max* differ in that they convey the bounds of the desired range of values, not possible settings. The remaining difference is that application monitors have a weight field to convey their relative importance. Monitors use a separate set of globally-defined types from knobs.

5.2.2 Cross-layer APIs

The cross-layer interactions of the framework are realised through an application-to-RTM API and an RTM-to-device API. These APIs connect the system layers of Figure 5.1 (dashed boxes) and enable the exposure of knobs and monitors between the three layers. The APIs define each possible function that can be called to perform a cross-layer interaction. Table 5.3 illustrates how the API functions are divided into application (**app**) and device (**dev**) categories at the top level. This separates the API functions into those that are interactions between the application and RTM layers and those that are between the RTM and device layers. Within each category, there are subcategories for knob (**knob**) and monitor (**mon**) control functions, determining which class of object is being affected. Discrete- and continuous-valued (**disc** and **cont**) versions of

each function exist across the API functions to signify the type of knob or monitor that is being targeted.

There are similar functions across the four subcategories of the API, the only difference is that the operation is performed on a different group of knobs or monitors. Therefore, each API function will not be covered individually, instead the distinct operations that are performed, and are common to each subcategory, will be discussed. To summarise these operations are; registration and deregistration, min and max bound and weighting updates and set and get updates. The following subsections present these functions for the application-to-RTM API and then the RTM-to-device API.

5.2.2.1 Application-RTM Interaction

The top half of Table 5.3 lists the application-to-RTM API functions that application developers must use to expose and update application-level knobs and monitors. In the first instance, applications must register themselves through the API using `app_reg()` in order to allow for their runtime management. Applications can deregister using `app_dereg()` when they no longer require runtime management, following knob and monitor deregistration. In both cases, the application must pass its Process Identifier (PID) to the function so that the RTM can uniquely identify it.

The function set `app_(knob|mon)_(disc|cont)_(reg|dereg)()` facilitates the dynamic registration and deregistration of individual application knobs and monitors. It is expected that applications will follow a similar approach in registering their knobs and monitors on start-up and deregistering them prior to termination. However there is no limitation to such events occurring at arbitrary points throughout application execution instead, for instance if a change in configuration means that a knob no longer has any valid settings, it could be deregistered. The number of knobs and monitors exposed is specific to the particular application. The application must pass to the function an initial value for each of the fields of the knob or monitor, including the type, min and max bounds, a weight for monitors and an initial value for knobs. The framework assigns an ID and the App ID to this newly registered knob or monitor, stores it in an internal record and returns a knob or monitor object back to the application. This object is then used in future API calls for its unique ID and App ID combination.

LISTING 5.1: API functions required to register an application (presented in Section 5.4.1.1) and create application-level knobs and monitors.

```

1 // Register application with RTM
  rtm_api.reg(get_pid());

  // Register application knobs
5 // Application knob for controlling number of iterations
  knob_disc_t iters_knob = rtm_api.knob_disc_reg(PRIME_ITR, 1, PRIME_DISC_MAX, 10);

  // Set up application knob for controlling kernel precision. Two discrete options: 0 =
    float, 1 = double
  knob_disc_t prec_knob = rtm_api.knob_disc_reg(PRIME_PREC, 0, 1, 1);
10 // Application knob for selecting device used to execute kernels. Two discrete options: 0 =
    CPU, 1 = GPU
  knob_disc_t dev_knob = rtm_api.knob_disc_reg(PRIME_DEV_SEL, 0, 1, 1);

  // Register application monitors

```


TABLE 5.3: Complete set of framework API functions.

Layer	Construct	Space	Identifier	Input(s)	Output(s)	Description
app	-	-	reg	pid	-	Register application
		-	dereg	pid	-	Deregister application
	knob		reg	type, min, max, val	knob	Register application knob
			dereg	knob	-	Deregister application knob
			min	knob, min	-	Update application knob's minimum allowed value
			max	knob, max	-	Update application knob's maximum allowed value
	mon		get	knob	value	Pull application knob's current value
			reg	min, max, weight	mon	Register application monitor
			dereg	mon	-	Deregister application monitor
			min	mon, min	-	Update application monitor's minimum desired value
dev	knob		max	mon, max	-	Update application monitor's maximum desired value
			weight	mon, weight	-	Update application monitor's relative importance
			set	mon, value	-	Push application monitor's current value
		disc / cont	reg	-	knobs	Register all device knobs
	mon		dereg	-	-	Deregister all device knobs
			min	knob	min	Pull device knob's minimum allowed value
			max	knob	max	Pull device knob's maximum allowed value
			init	knob	init	Pull device knob's initial (default) value
	dev		type	knob	type	Pull device knob's type
			set	knob, value	-	Push device knob's current value
mon			reg	-	mons	Register all device monitors
			dereg	-	-	Deregister all device monitors
			min	mon	min	Pull device monitor's minimum allowed value
			max	mon	max	Pull device monitor's maximum allowed value
			type	mon	type	Pull device monitor's type
			get	mon	value	Pull device monitor's current value

```

15 // Application monitor for observing error. Required bound is (PRIME_CONT_MIN,
    CONVERGENCE_THRESHOLD]
    mon_cont_t error_mon = rtm_api.mon_cont_reg(PRIME_ERR, PRIME_CONT_MIN,
        CONVERGENCE_THRESHOLD, 1.0);

    // Application monitor for observing throughput. Required bound is [THROUGHPUT_THRESHOLD,
    PRIME_CONT_MAX)
    mon_cont_t throughput_mon = rtm_api.mon_cont_reg(PRIME_PERF, THROUGHPUT_THRESHOLD,
        PRIME_CONT_MAX, 1.0);

```

The application code in Listing 5.1 shows example function calls to register an application and its knobs and monitors into the framework so that they are accessible from the runtime management layer. Three knobs are registered, each passing in a type, minimum, maximum and initial value field. The registration function returns a discrete knob struct (`knob_disc_t`) for each. Following this, two monitors are registered, one that reports error and the other performance, using the application monitor registration function. The application must specify the initial minimum and maximum bounds on the monitor value, as well as a weight. Similar to knobs, a continuous monitor struct (`mon_cont_t`) is returned for each.

After registration of all its knobs and monitors, the RTM holds all the information it needs about an application in order to optimise its execution and meet its specified monitor bounds. To enable the dynamic adjustment of the parameters of knobs and monitors mid-way through execution, separate API functions are provided to update each field individually. To adjust the range of allowable values for knobs, the functions `app_knob_(disc|cont)_(min|max)()` are provided, letting the application indicate a new range from which values can be chosen. Conversely, monitor functions `app_mon_(disc|cont)_(min|max|weight)()` allow the adjustment of RTM objectives, with `*_min()` and `*_max()` functions indicating a new desired lower and upper bound. Where an application demands only a maximum or minimum value, `PRIME_(DISC|CONT)_MIN` or `PRIME_(DISC|CONT)_MAX` may be used in place of the lower or upper bound, respectively. This effectively leaves the monitor unbounded at this end. Intra-application weighting values between 0.0 and `PRIME_CONT_MAX` can be used to adjust the relative monitor importance to the RTM using `*_weight()` functions, guiding its optimisations as discussed above.

LISTING 5.2: API functions required to get updated values of application knobs and set updated monitor values through the framework.

```

1 // Fetch RTM's desired knob values
  iters_knob.val = rtm_api.knob_disc_get(iters_knob);
  prec_knob.val = rtm_api.knob_disc_get(prec_knob);
  dev_knob.val = rtm_api.knob_disc_get(dev_knob);
5
  start_time = prime::util::get_timestamp();
  << Run one cycle of application >>
  end_time = prime::util::get_timestamp();

10 throughput = (prime::api::cont_t)1/(end_time - start_time);
   rtm_api.mon_cont_set(throughput_mon, throughput);
   throughput_mon.val = throughput;

   // Calculate square of L2 norm. Update RTM with currently achieved error
15 norm_sq = (prime::api::cont_t)l2norm_sq(A, b, x);
   rtm_api.mon_cont_set(error_mon, norm_sq);
   error_mon.val = norm_sq;

```

Functions `app_knob_(disc|cont)_get()` and `app_mon_(disc|cont)_set()` enable the pulling and pushing of knob and monitor values from and to the RTM, respectively, by the applications

that define them. During runtime operation, interactions between applications and the RTM predominantly consist of updates to knob and monitor values before and after an execution cycle of the application's workload. Listing 5.2 demonstrates this process; first with the application pulling knob updates through the framework; executing the main body of its workload and then updating the value of its registered monitors. The timing of these actions is entirely controlled by the application and follows the hierarchy of the framework model as outlined in the previous section. The application determines when updated monitor values are passed to the RTM and when new knob values are pulled. Therefore, for knob updates there will be a delay between when the RTM sets a new value for a knob, when that value is received by the application and when the application puts the new knob value into effect.

LISTING 5.3: API functions required to deregister an application and its knobs and monitors from the framework.

```

1 // Deregister knobs
  rtm_api.knob_disc_dereg(iters_knob);
  rtm_api.knob_disc_dereg(prec_knob);
  rtm_api.knob_disc_dereg(dev_knob);
5 // Deregister monitors
  rtm_api.mon_cont_dereg(error_mon);
  rtm_api.mon_cont_dereg(throughput_mon);
  // Deregister application with RTM
  rtm_api.dereg(proc_id);

```

Listing 5.3 shows an example of the function calls that must be made during the deregistration process. The knobs and monitors of an application must be deregistered first, followed by the application itself.

5.2.2.2 RTM-Device Interaction

LISTING 5.4: API functions for the RTM to register the device and its knobs and monitors.

```

1 // Register all device knobs
  vector<prime::api::dev::knob_disc_t> device_knobs_disc = dev_api.knob_disc_reg();
  vector<prime::api::dev::knob_cont_t> device_knobs_cont = dev_api.knob_cont_reg();
  // Register all device monitors
5 vector<prime::api::dev::mon_disc_t> device_mons_disc = dev_api.mon_disc_reg();
  vector<prime::api::dev::mon_cont_t> device_mons_cont = dev_api.mon_cont_reg();

```

Device-level knobs and monitors are registered and updated via the RTM-to-device API functions, as shown in the lower half of Table 5.3. The exposure of device knobs and monitors at the device level is performed differently to applications. Due to the framework hierarchy, the device cannot call the API to register its knobs and monitors. Instead, knobs and monitors are created internally by the device, which connects them to their respective drivers. When the RTM calls one of the function set `dev_(knob|mon)_(disc|cont)_(reg|dereg)()` the device knobs or monitors are registered or deregistered with the framework. This function call is also different in that it operates on all the knobs or monitors of a particular space (*i.e.* discrete or continuous) that are available within a device. A standard practise would be for the RTM to query for available device knobs and monitors at regular intervals, in the event that a device resource is newly or no longer available. As with application knobs and monitors, registration at the device level returns a vector of knob and monitor objects, which contains all the required

fields for the RTM to begin using them. An example set of function calls to register discrete and continuous device knobs and monitors is shown in Listing 5.4.

The knob-related functions `dev_knob_(disc|cont)_(min|max)()` are parallels of their application-level counterparts, and enable dynamic adjustment of the limits on the values that can be selected for the device knob. Additional functions `dev_knob_(disc|cont)_(type|init)()` return the type or initial value. Monitors have type-indicating functions only, probed using `dev_mon_(disc|cont)_type()`. Calls to `*_type()` functions return values from the sets presented in Table 5.5.

LISTING 5.5: The RTM pulling monitor values from the device and pushing device knob settings through the API.

```
1 // Get cycle count values for two cores from device monitors
  unsigned int cycle_count_x = dev_api.mon_disc_get(cycle_count_x_mon);
  unsigned int cycle_count_y = dev_api.mon_disc_get(cycle_count_y_mon);

5 // Run RTM providing cycle count and returning freq_choice
  controller.run(freq_choice, cycle_count_x, cycle_count_y);

  // Set the frequency using the device knob
  dev_api.knob_disc_set(frequency_knob, freq_choice);
```

At runtime, the RTM pushes device knob settings and pulls monitor values as it requires; functions `dev_knob_(disc|cont)_set()` and `dev_mon_(disc|cont)_get()` facilitate these actions. This process is demonstrated in Listing 5.5 for the case of an RTM reading two cycle count monitors, using these as inputs to its control loop, and setting the optimised frequency as an output through the API.

Additional listings containing code to demonstrate the operation of the API functions are provided in Appendix E. In the examples used throughout this section, a total of only 18 single-line function calls were added: 14 for registration and deregistration, four to pull knob setting updates and two to push monitor values. The next section discusses the additional components that have been developed in order to demonstrate the framework and API as part of a C++ software implementation.

5.3 Framework Software Implementation

An implementation of the framework has been developed in C++, which can run on any Linux-based operating system. The development of a complete software implementation is designed to demonstrate the concepts of the framework in practice using real-world example applications, RTMs and devices. This is a major step towards proving and validating that the framework is a viable approach to delivering cross-layer optimisation in an agnostic fashion. An open-source release of this implementation is available online¹, including the example applications, RTMs and device classes discussed in Section 5.4.

This section presents information on important components that have been created to realise this implementation. A programmatical view of the framework software is shown in Figure 5.3, which shows the location and connection of each software block.

¹Available at: https://github.com/PRIME-project/PRIME_Framework

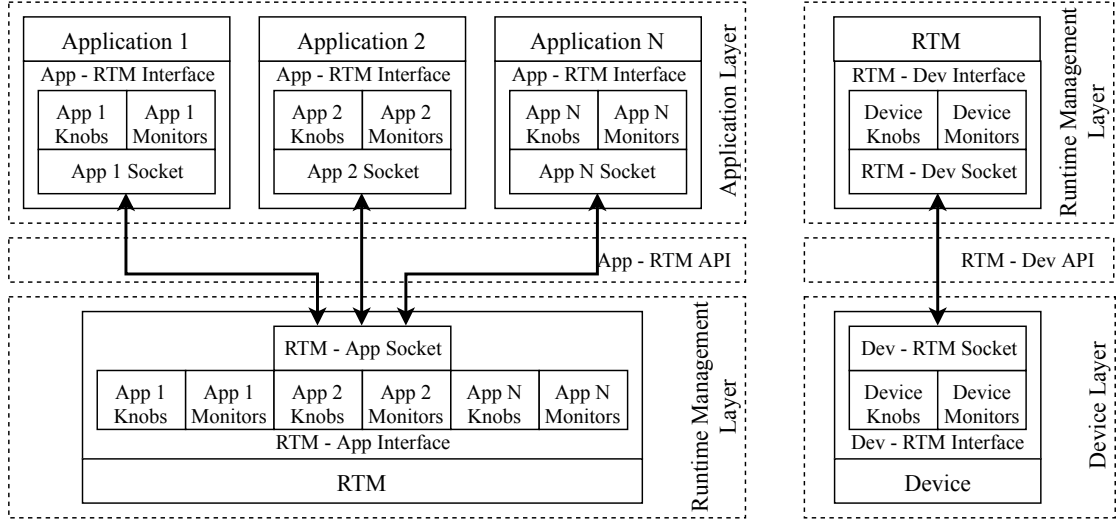


FIGURE 5.3: Implementation of the framework from a software-component perspective, including the location of knobs and monitors, interfaces and sockets in each layer.

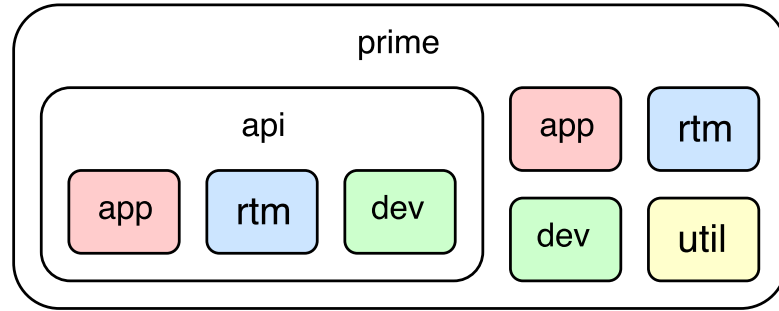


FIGURE 5.4: Namespaces in this implementation of the framework to provided code separation.

5.3.1 Namespaces

In this implementation, C++ namespaces are used to separate the code along the layers of the framework model. Figure 5.4 shows that at the top level is the **prime** namespace, below this the code for applications, RTMs and devices are separated into the **app**, **rtm** and **dev** namespaces, respectively. In addition, framework-specific components of the code are located in the **api** namespace, which is sub-divided into namespaces for each layer. A **util** namespace is provided for common functions that are used across all the layers of the framework. These namespaces mean that some API functions can be shortened as the namespaces are implicitly inferred from the context of the code.

5.3.2 API Interfaces

Interfaces are created to connect the device, RTM and each application to the framework and allow them to use the API. These interfaces are classes that act as a wrapper for the framework-specific parts of each entity in a layer, including its knobs and monitors. API calls must be made using these interfaces in order to ensure that the information is routed to the correct destination interface, *i.e.* an application monitor update is passed to the RTM layer.

The code in Listing E.4 in Appendix E shows the `rtm_interface` class, which resides in the application API header. The class body can be found in the GitHub repository. All the API functions are contained in this class header as well as private local copies of the application's knobs and monitors. These copies can only be updated through valid API calls made by the application. For the RTM, both an `app_interface` and `dev_interface` exist in its API namespace, due to its need to interact with both APIs. The device API namespace contains only an `rtm_interface`, as with applications. This contains functions required to allow the device to respond to API calls made to it by the RTM.

5.3.3 Interprocess Communication

The device, RTM and applications run as separate linux processes, therefore an interprocess communication mechanism is required to allow them to communicate using the API. This communication is performed using a message-passing protocol over local Unix Domain Sockets (UDSs). These sockets are based on the UDS implementation in the `boost::asio` library, to enable the asynchronous sending and receiving of messages between processes. To make an API call across the message-passing interface, the required information in the call must be serialised, transmitted, de-serialised and reconstructed into the form of the original API call, so that the receiving layer can uniquely identify the purpose of the message.

A JavaScript Object Notation (JSON) protocol has been developed to store and serialises the information for each API call. The `boost::property_tree` and JSON parser library are used to perform the packing and unpacking of messages passed between processes. JSON is a highly flexible and extensible notation, based on attribute-value pairs and data arrays, with a very basic syntax. This means it can be used to store arbitrary data structures with ease, which is useful given the range of signatures (number and type of inputs and outputs) of the API calls in the framework.

LISTING 5.6: Example JSON code for an application knob registration message.

```

1 {
    "ts":1509975001132193,
    "type":"PRIME_API_APP_KNOB_DISC_REG",
    "data":
5   {
        "knob":
        {
            "proc_id":18530,
            "type":"PRIME_ITR",
10         "min":1,
            "max":2147483647,
            "val":10
        }
    }
15 }
```

Listing 5.6 shows an example JSON message for the registration of an application knob into to the framework. The message is sent from the `rtm_interface` of the app to the `app_interface` of the rtm so that the RTM can receive a copy of the knob. JSON messages in the framework must contain a type field (a unique identifier string), a global timestamp, and a data field. The data field contains the contents of the message and can be of arbitrary structure, although its

structure should be defined by the type field. For the example shown, the type field identifies the original calling API function, therefore the class of knob can be inferred (**disc**) and the intended operation (**reg**). The data field contains the fields that are required to construct the knob, *i.e.* the knob type, min, max and value.

LISTING 5.7: Discrete knob registration case in the message handler function of the app interface of the RTM layer.

```

1      else if(!message_type.compare("PRIME_API_APP_KNOB_DISC_REG")) {
        pid_t proc_id = data.get<pid_t>("proc_id");
        prime::api::app::knob_disc_t knob;
        knob.proc_id = proc_id;
5      knob.id = get_unique_knob_id();
        knob.type = (prime::api::app::knob_type_t)data.get<unsigned int>("type");
        knob.min = data.get<prime::api::disc_t>("min");
        knob.max = data.get<prime::api::disc_t>("max");
        knob.val = data.get<prime::api::disc_t>("val");
10     knobs_disc_m.lock();
        knobs_disc.push_back(knob);
        knobs_disc_m.unlock();
        return_knob_disc_reg(knob);
        knob_disc_reg_handler(proc_id, knob);
15     }

```

The **app_interface** in the runtime management layer receives this message from the UDS socket and processes it using a special message handler function by matching the type field with a known list of API types. Listing 5.7 shows part of the message handler function in the **app_interface** for the parsing of a discrete knob registration message. The knob fields are extracted from the **data** node to create a new knob object, before the function links to an optional handler (**knob_disc_reg_handler**) provided by the RTM developer.

5.3.4 Component Classes

Component classes are the highest level containers of all the objects of an application, RTM or device. They can be thought of as the macro-blocks in Figure 5.3, including the interface (with the knobs, monitors and sockets) and the code for the component in that layer (*e.g.* application code or RTM algorithm). They represent all the code that executes as a single process.

These classes act as a template container for application-, RTM- or device-specific code, which sits alongside the API interfaces. This is where all the custom code is written for a particular instantiation of a layer (*e.g.* a runtime algorithm, platform or application). The class can contain local copies of knobs and monitors for its layer, such as in application classes, or copies of knobs and monitors from other layer, such as in the RTM class. However, protected versions of these knobs and monitors are always stored in API interfaces and are the ones used and modified in API calls.

The next section presents example application, RTM and device class that have been used for experimental validation of the runtime management process through the framework.

5.4 Benchmarks

The cross-layer framework and its API enable rapid experimentation with different runtime management algorithms across multiple platforms and applications. The framework promotes comparison between state-of-the-art runtime algorithms through the standard knobs and monitor constructs. These qualities are demonstrated through a series of experiments using interchangeable components in each layer that are termed benchmarks. These are described for each layer in the following sections.

5.4.1 Applications

The application layer contains application software and the requisite API functions to allow interaction with the runtime management layer. Any application used within the framework must provide at least one monitor in order for the runtime management layer to meaningfully optimise its execution.

Three applications, described in Sections 5.4.1.1, 5.4.1.2 and 5.4.1.3, were chosen to illustrate how the framework can enhance any software program to support runtime management. The knobs and monitors exposed for each application are listed in Table 5.4, including whether they are discrete or continuous. In addition, the power, performance and accuracy trade-offs of these applications are explored and their runtime control within the framework is demonstrated.

5.4.1.1 Jacobi Iterative Method

The Jacobi matrix solver is an application with an iterative pattern of behaviour. The application solves the system of N linear equations $\mathbf{Ax} = \mathbf{b}$, where \mathbf{A} is an $N \times N$ matrix and \mathbf{x} and \mathbf{b} are $N \times 1$ column vectors. If \mathbf{A} is decomposed into diagonal and remainder components \mathbf{D} and \mathbf{R} , under suitable conditions \mathbf{x} can be computed iteratively via (5.1), also shown in an element-wise fashion in (5.2), where k is the iteration index. Each element of $\mathbf{x}^{(k+1)}$ relies only upon the previously calculated value of \mathbf{x} . Therefore, (5.2) can be parallelised by computing each $x_i^{(k+1)}$ independently.

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1} (\mathbf{b} - \mathbf{Rx}^{(k)}) \quad (5.1)$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right) \quad (5.2)$$

The result of the application is evaluated against the convergence criterion $\|\mathbf{Ax} - \mathbf{b}\| < \epsilon$, which is used for establishing the final result's accuracy and is dependent on K , the number of iterations performed. Tuning K operates a trade-off between accuracy and computational speed. The application exposes three knobs and two monitors, presented in the first section of Table 5.4. In the context of this experiment, $\|\mathbf{Ax} - \mathbf{b}\|$ is captured as an error monitor, with maximum bound ϵ . A throughput monitor reports the time taken to complete K iterations of

TABLE 5.4: Application-level knobs and monitors present in Jacobi (Section 5.4.1.1), video decoder (5.4.1.2) and Whetstone (5.4.1.3).

Application	Name	Const.	Space	Allowed/target values
Jacobi	Iterations	knob	disc	$\mathbb{N} \in [1, \infty)$
	Data type	knob	disc	<code>{float,double}</code>
	Device type	knob	disc	<code>{CPU,GPU/FPGA}</code>
	Throughput	mon	cont	$\mathbb{R} \in [10, \infty)$
	Error	mon	cont	$\mathbb{R} \in (-\infty, 1e^{-12}]$
Video decoder	Throughput	mon	cont	$\mathbb{R} \in [25, \infty)$
Whetstone	Threads	knob	disc	$\mathbb{N} \in [1, \infty)$
	Throughput	mon	cont	$\mathbb{R} \in [2.5, \infty)$

the algorithm. The application was implemented as OpenCL kernel code to support execution on multiple devices; an embedded GPU and FPGA for acceleration. K work-groups, each comprised of N work-items, one per element of \mathbf{x} , are launched for each iteration. Two distinct kernel implementations are provided for single- and double-precision floating-point operations. Along with a variable number of iterations, this enables trade-offs to be made between precision and accuracy. The ARM CPU clusters are not OpenCL-compatible devices, therefore a multi-threaded implementation of the application was created using POSIX threads as well. The *device type* application knob is provided to support switching between implementations. In Section 5.6.1, the knobs and monitors are profiled to expose the Pareto-optimal operating points for a trade-off between error and performance.

5.4.1.2 Video Decoder

The video decoder is a multimedia application that uses the OpenCV VideoCapture API to open a video file and read in frames. The application exposes a single monitor, as shown in the second section of Table 5.4. This performance monitor gives the time taken to decode each video frame. The target frame rate can be controlled using the bounds of the monitor.

5.4.1.3 Whetstone Benchmark

Whetstone is a benchmark that measures the performance of integer and floating-point arithmetic with a variety of numerical functions [159]. While it is a synthetic benchmark, Whetstone exercises many of the subsystems of a CPU used by numerical applications, making it a suitable proxy for assessing their performance within the framework. The benchmark is divided into ten modules representing a different arithmetic, data movement or control flow operation, in the patterns which they occur, for real-world numerical applications. The knobs and monitors exposed by the application are shown in last section of Table 5.4. The knob controls the number of threads that are executed by the application simultaneously. Each thread executes an independent benchmark, rather than threading multiple benchmarks, to ensure that all cores in a multi-core system are equally exercised. The monitor represents the performance of the

application measured in thousands of Whetstone instructions per second (KIPS). This is calculated as $KIPS = 100^{N_{loop}/t_{exec}}$, where N_{loop} is the loop multiplier, a scaling factor applied to the number of repetitions of each module to increase the execution time, and t_{exec} is the time taken to execute all of the modules.

5.4.2 Runtime Managers

The runtime management layer is central to the operation of the framework as it is responsible for controlling the application and device layers through the knobs and observing the state of the system through the monitors, as illustrated in the central layer of Figure 5.1. The contents of this layer can take any form, such as those discussed in Section 2.4.1. An example structure is shown in Figure 5.5, which places an RTM in the context of the framework and highlights the act-observe-adjust process. This RTM arrangement is centred around a runtime model of the system, which encodes information on the trade-offs between all the application and device knobs and monitors exposed by the framework. Registrations, bound, weighting and type changes feed into the application and device profiles. Profiles contain information about the structure of applications and devices and are modified when registration or configuration updates occur. Data is fed into the RTM's state recorder from monitor value updates and is filtered into a cache of application and device data histories. This enables the layer to both observe and learn the system's behaviour.

The profiles and histories are used as inputs to construct the runtime model. The runtime controller interrogates the runtime model to predict optimal knob settings based on bounds set by the monitors and attempts to co-optimize any that are unbounded. For example, gradient descent can be used to optimize power consumption under a performance constraint. This achieves the act and adjust elements of the methodology.

The operation of the runtime management layer within the framework is demonstrated with two runtime algorithms from the literature. These are discussed in more detail in the following two sections and their operation within the framework is experimentally validated in Section 5.6.2.

5.4.2.1 Profiler Runtime Manager

A profiler is used to explore the runtime operating space by testing every combination of device and application knob values and recording the resulting monitor values. This exposes the relationships between knobs and monitors and the potential trade-offs between monitors, which are demonstrated experimentally in Section 5.6.1. The operation of the profiling is an extension of the characterisation technique used in Section 4.3 of Chapter 4 to characterise the operating space of the DE algorithm. Here it is extended to use the generic knobs and monitors to test and record data about the behaviour of the system.

5.4.2.2 Q-Learning Runtime Manager

This RTM uses a Q-Learning algorithm to ensure that application-specific performance requirements are met while reducing system energy by scaling the operating voltage and frequency

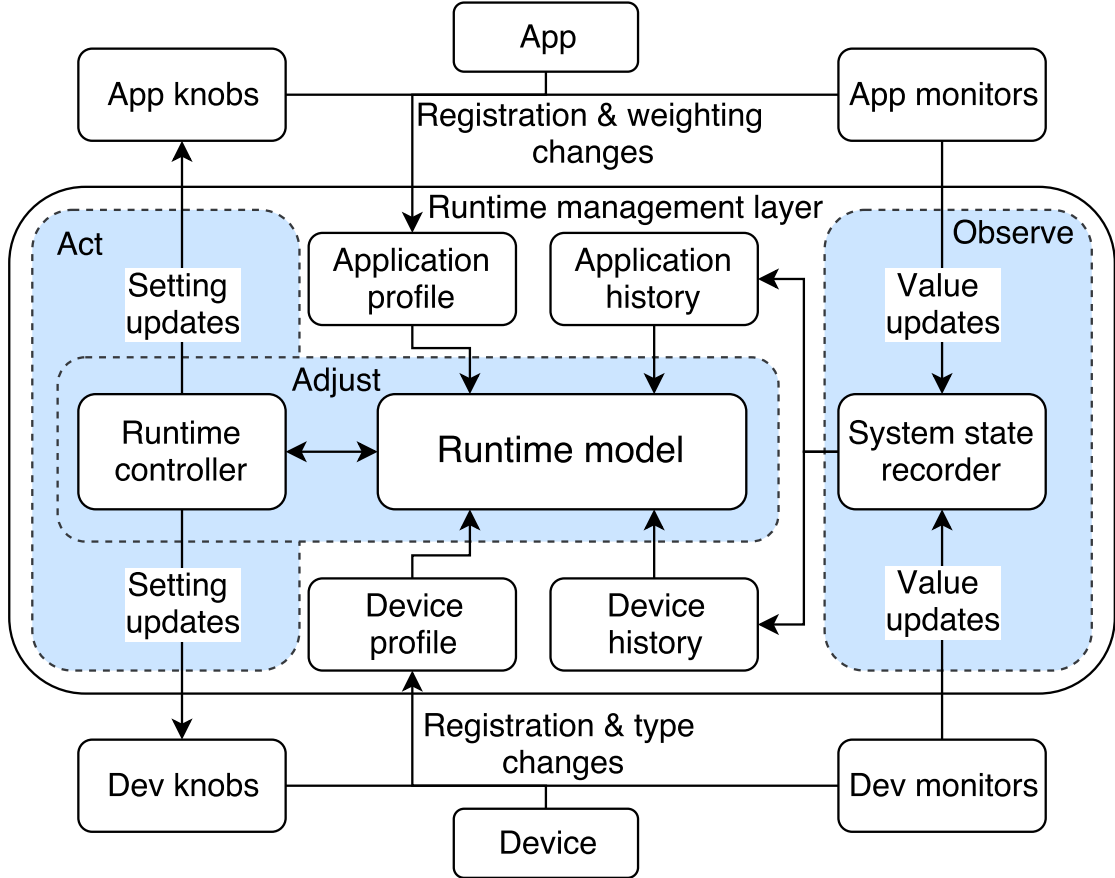


FIGURE 5.5: Example structure of the runtime management layer with the act-observe-adjust phases highlighted.

[160]. In a standard Q-learning approach, a learning agent (the RTM) repeatedly observes the state of the system at a set rate in order to predict the next state with predictions updated and stored in a Q-table. Based on this prediction an appropriate action is selected to achieve the specified optimisation objective. The objective, for example application performance or power reduction, is quantified using a numeric payoff, with positive payoffs being classed as a reward and negative ones as a penalty.

Initially the RTM has no knowledge of how its actions will affect the state of the system or what rewards its actions will produce. As such, the RTM starts with an *exploration* phase where it experiments with various actions in different states to determine the predicted payoff for each action. Over time, the confidence in the selected action improves and the algorithm always selects the action corresponding to the highest payoff. This is known as the *exploitation* phase. The algorithm balances the two modes of operation with an epsilon-decreasing strategy, where the best action is selected for the proportion $(1 - \epsilon)$ of states and a random state is selected for the proportion ϵ . ϵ decreases over time in order to favour *exploitation* of known high rewards. This form of runtime algorithm is best suited to frame-based applications and the RTM is updated every 10 frames or cycles of the application with the state of the system, determined using the CPU cycle count. The operation of this RTM with the video decoder application, operating in the framework, is validated in Section 5.6.2.

5.4.2.3 PMC-based Runtime Manager

This RTM classifies workloads based on their memory intensiveness using the Memory Reads Per Instruction (MRPI) metric [161]. Workloads with a high MRPI are classified as memory intensive and the V - f level is scaled down to minimise power consumption, as memory-intensive workloads can be run at a lower frequency with little to no performance loss. MRPI is derived from two CPU performance counters; the number of last-level cache data read refills and instructions retrieved. These show a high correlation with the memory intensiveness of a workload with little variation due to frequency scaling when compared to CPU cycle count and memory reads per cycle.

Unlike the Q-Learning RTM, the PMC-based RTM uses offline profiling to support workload classification and V - f selection. Workload prediction is carried out using an exponentially weighted moving-average filter and, to minimise workload mispredictions, the predicted workload for the interval $t - 1$ to t is compared with the actual workload measured from hardware PMCs. The computed prediction error—the difference between actual and predicted workloads—is used to improve the workload prediction for t to $t + 1$.

This RTM is used to demonstrate that approaches targeted at non-frame-based applications, or those that rely on offline profiling, can be used within the context of the framework. Validation of this RTM is presented in Section 5.6.2.

5.4.3 Devices

The device layer is responsible for interfacing between the runtime management layer and the underlying hardware, facilitating the control of device parameters, such as operating frequency and voltage, and the presentation of device data sources in a standardised manner across different platforms. One platform can incorporate multiple devices and, for the purposes of the framework, a device is defined as a single functional unit, such as a CPU cluster, graphics processing unit (GPU), digital signal processor (DSP) or field-programmable gate array (FPGA) fabric. Specific types are assigned to the knobs and monitors exposed by the device layer as summarised in Table 5.5. Additionally, to allow the runtime management layer to be as device-agnostic as possible, the device layer communicates information about which knobs and monitors are available, and their relationships to each other, to the runtime management layer at runtime.

The runtime management layer interfaces directly to the operating system to control CPU thread-to-core affinity. This is preferable to the provision of a knob from the device layer as it avoids transferring application-specific information to the device. Two heterogeneous multi-core platforms have been used to demonstrate knob and monitor support for devices within the framework:

5.4.3.1 Odroid XU3

The Odroid XU3 contains a rich set of parameters that are exposed through the framework as knobs and monitors. These are summarised in the first half of Table 5.6. The six knobs come from frequency and governor control of each of the CPU clusters and frequency control of the

TABLE 5.5: Available device-level knob and monitor types.

Construct	Type	Description
knob	EN	Enables device
	VOLT	Controls voltage
	GOVERNOR	Controls CPU frequency governor
	FREQ_EN	Enables <i>userspace</i> CPU frequency control
	FREQ	Controls frequency
	PMC_CTRL	Controls performance counter
mon	POW	Reports power consumption
	TEMP	Reports temperature
	PMC	Reports performance counter data
	CYCLES	Reports cycle count data

TABLE 5.6: Device-level knobs and monitors for Odroid-XU3 (Section 5.4.3.1) and Cyclone V (5.4.3.2) platforms.

Plat.	Const.	Space	Type	For	No.
Odroid-XU3	knob	disc	GOVERNOR	A7 cluster	1
		disc	GOVERNOR	A15 cluster	1
		disc	FREQ	A7 cluster	1
		disc	FREQ	A15 cluster	1
		disc	FREQ_EN	GPU DVFS	1
		disc	FREQ	GPU	1
		disc	PMC_CTRL	A7 cores	16
		disc	PMC_CTRL	A15 cores	24
	mon	cont	POW	Clusters, RAM, GPU, SoC	5
		cont	TEMP	A15 cores	4
		cont	TEMP	GPU	1
		disc	CYCLE	A7 cores	4
		disc	CYCLE	A15 cores	4
		disc	PMC	A7 cores	16
		disc	PMC	A15 cores	24
Cyclone V	knob	cont	VOLT	A9 cluster, peripherals	4
		cont	VOLT	FPGA, peripherals	3
	mon	cont	POW	A9 cluster, peripherals	5
		cont	POW	FPGA, peripherals	4
		cont	POW	SoC	1

GPU. The 58 monitors are derived from a variety of sources included in the platform; five power measurements from hardware sensors on the board, each A7 core has a cycle counter and four PMCs, each A15 core has a cycle counter, six PMCs and a temperature sensor and the GPU has a temperature sensor.

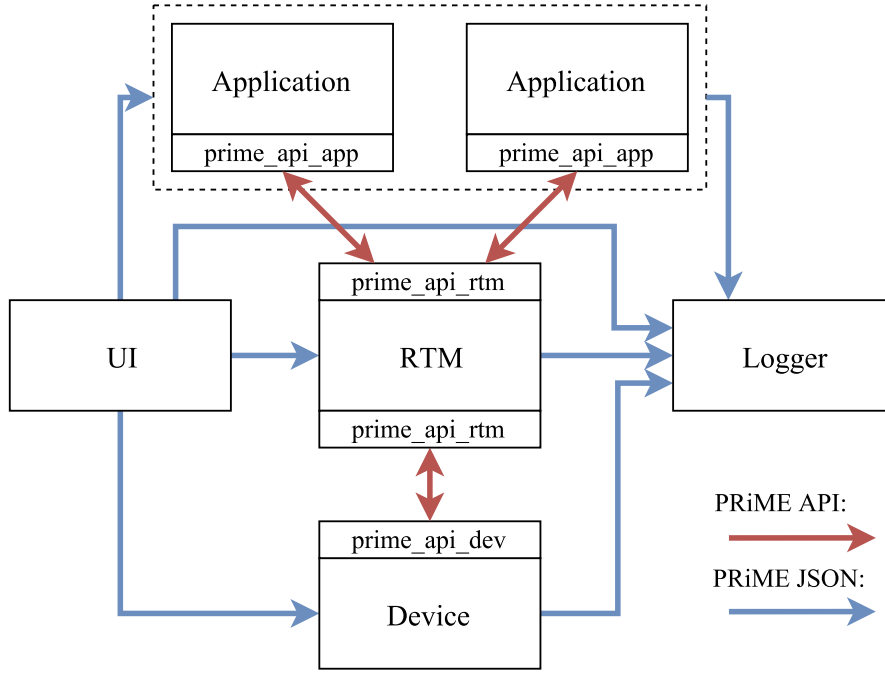


FIGURE 5.6: Block diagram of the tools (UI and Logger) integrated into the framework to aid experimentation.

5.4.3.2 Cyclone V

The Altera Cyclone V SoC development board tests the operation of the framework with a CPU-FPGA platform. Knobs and monitors exposed through the device's class are detailed in the lower half of Table 5.6, including monitors for the three power regulators present on the board and monitoring of the CPU and FPGA core voltages.

5.5 Framework Experimentation Tools

This section introduces some experimentation tools that have been developed for the framework in order to standardise the experimental process and increase its capabilities. The tools also aid in the comparison of different RTM approaches by providing a standard experimental setup.

Figure 5.6 shows how the tools fit around the existing layers of the framework and interact with them in order to control certain processes and provide external stimuli. The tools are made up of two components; the User Interface (UI) and the Logger. These are introduced in more detail in the next two sections. They enable greater debugging capabilities, through the standardised collection of data and the formation of scripted, repeatable experiments. These tools would not be included in a deployment of the framework in a final commercial system but could be used to analyse a prototype of the system during its development. For example, to examine if a runtime algorithm is operating correctly and does perform energy optimisation.

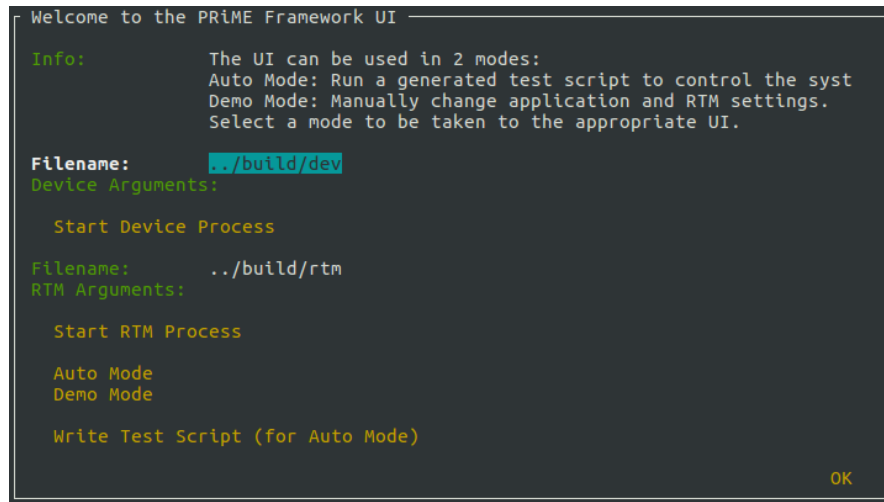


FIGURE 5.7: Main menu of the UI.

5.5.1 User Interface

The UI is composed of a set of Python scripts which provide a terminal-based graphical interface to control components of the framework. This includes; starting and stopping the application, RTM and device processes, registering and deregistering applications, and adjusting any parameters that are intended to be controlled by an external user. In a deployed system, many of these operations would be automated, fixed or exposed to the user through the application's UI. When commands are activated, they are sent to the target application through the same message passing interface as is used between the system layers, except a specific `ui_interface` and UI UDS are used.

The main menu (Figure 5.7) allows a developer to start and stop the device and RTM processes as well as provide command-line arguments to each. The remaining buttons link to two modes for using the UI:

Demonstration (Demo) Mode (Figure 5.8) enables manually control of application behaviour. The user can control the triggering of some of the application's API calls including; registration and deregistration, application weighting, monitor weighting and bound updates.

Automatic (Auto) Mode allows a developer to run scripted experiments that perform the same actions as are possible in demo mode but using an accurate timing system (similar to a testbench in SystemVerilog). The test script is composed of JSON commands, which encode the API call to instruct the application to make and all the necessary additional parameters. This mode can be used to create particular application-behavioural patterns, either in terms of their registration and deregistration or in terms of the fluctuations in their monitor bounds. This method could be used to create the fluctuation in performance requirement demonstrated in Figure 4.18 of Section 4.5.

```

odroid@prime-odroid-xu3-arm-1: ~/PRIME_Framework/ui
PRIME Framework UI - Demo Mode

Info:      Demo Mode: Manually change application and RTM settings.
           Options: Add or detect Apps, Start/Stop Apps, Register/Deregister Apps with RTM
           - Adjust App Monitor min/max/weight, Adjust App weight.

Application Search:
  Add App/Find Apps      /home/odroid/PRIME_Framework/build/app

Applications:  (X)  app

Arguments:
  Stop
  Deregister
App Weighting:  [Progress Bar] 58.0%

Application Monitors:
  (X)  1

Minimum:      20
Maximum:      1000
Weighting:     0

Log:
Found Application: app
Application Path: /home/odroid/PRIME_Framework/build/app
Selected app Application
Choose your next action
Starting Application: app
Sending message to address: /tmp/logger.uds
Application Started: app
UDS Address: /tmp/app.ui.3068.uds
Registering Application: app
Sending message to address: /tmp/app.ui.3068.uds
Application Registered: app
Choose your next action
Monitor registered: Application PID = 3068, Monitor ID = 1 (0 < val < 1000) weight = 0.

```

FIGURE 5.8: Demo mode of the UI.

```

PRIME Framework UI - Auto Mode

Info:      Auto Mode: Run a generated test script to control the system.
           Type the filename of the test script or browse to its location.
           Select "Run Test" to load and start the test script.

Filename:   [Enter] Select a test file
Time Mode:  ( ) Cumulative
           ( ) Additive
Parse Test Script
Run Test Script

Trace Log:

OK

```

FIGURE 5.9: Auto mode of the UI.

5.5.2 Logger

The logger is a tool designed to aggregate all the information about the state of the system and framework at runtime in order to analyse and debug its behaviour. Every message sent between two interfaces in the framework is also echoed to the logger module. This is illustrated with the blue arrows in Figure 5.6, which route from every module to the logger. The logger performs basic processing on the input messages to extract the useful information and stores it in a standard format.

LISTING 5.8: Extract from the logger output of an experiment using the framework.

```

1 api:PRIME_API_APP_REG,ts:1128182,proc_id:18530,
  api:PRIME_API_APP_RETURN_APP_REG,ts:1130323,proc_id:18530,
  api:PRIME_API_APP_KNOB_DISC_REG,ts:1132193,proc_id:18530,type:PRIME_ITR,min:1,max
    :2147483647,val:10,

```

```

5  api:PRIME_API_APP_RETURN_KNOB_DISC_REG,ts:1134841,proc_id:18530,id:1,type:PRIME_ITR,min:1,
    max:2147483647,val:10,
    api:PRIME_API_APP_KNOB_DISC_REG,ts:1137749,proc_id:18530,type:PRIME_PREC,min:0,max:1,val:1,
    api:PRIME_API_APP_RETURN_KNOB_DISC_REG,ts:1139623,proc_id:18530,id:2,type:PRIME_PREC,min:0,
    max:1,val:1,
    api:PRIME_API_APP_KNOB_DISC_REG,ts:1142256,proc_id:18530,type:PRIME_DEV_SEL,min:0,max:1,val
    :1,
    api:PRIME_API_APP_RETURN_KNOB_DISC_REG,ts:1144149,proc_id:18530,id:3,type:PRIME_DEV_SEL,min
    :0,max:1,val:1,
10 api:PRIME_API_APP_MON_CONT_REG,ts:1146771,proc_id:18530,type:PRIME_ERR,min:-inf,max:1e-14,
    weight:1.0,
    api:PRIME_API_APP_RETURN_MON_CONT_REG,ts:1148841,proc_id:18530,id:1,type:PRIME_ERR,min:-inf
    ,max:1e-14,val:0.0,weight:1.0,
    api:PRIME_API_APP_MON_CONT_REG,ts:1152037,proc_id:18530,type:PRIME_PERF,min:10.0,max:inf,
    weight:1.0,
    api:PRIME_API_APP_RETURN_MON_CONT_REG,ts:1154002,proc_id:18530,id:2,type:PRIME_PERF,min
    :10.0,max:inf,val:0.0,weight:1.0,
    api:PRIME_API_APP_RETURN_KNOB_DISC_GET,ts:4523063,id:1,val:1,proc_id:18530,type:PRIME_ITR,
15 api:PRIME_API_APP_KNOB_DISC_GET,ts:4522559,id:1,proc_id:18530,type:PRIME_ITR,
    api:PRIME_API_APP_RETURN_KNOB_DISC_GET,ts:4524138,id:2,val:0,proc_id:18530,type:PRIME_PREC,
    api:PRIME_API_APP_KNOB_DISC_GET,ts:4523774,id:2,proc_id:18530,type:PRIME_PREC,
    api:PRIME_API_APP_RETURN_KNOB_DISC_GET,ts:4524878,id:3,val:1,proc_id:18530,type:
    PRIME_DEV_SEL,
    api:PRIME_API_APP_KNOB_DISC_GET,ts:4524610,id:3,proc_id:18530,type:PRIME_DEV_SEL,
20 api:PRIME_API_APP_MON_CONT_SET,ts:6037504,id:2,val:0.66132,proc_id:18530,type:PRIME_PERF,
    api:PRIME_API_APP_MON_CONT_SET,ts:6646844,id:1,val:17852.8,proc_id:18530,type:PRIME_ERR,

```

An extract from the logger’s output during an experiment is shown in Listing 5.8. The logger output is formatted with a combination of a JSON-style key-value pair and comma separator for each field of the message. This format allows simple parsing by post-processing scripts. Such scripts are included in Appendix F and were used to generate the figures in Section 5.6.1 from the data collected by the profiler RTM described in Section 5.4.2.1.

5.6 Experimental Evaluation and Results

Using the application, RTM and device components described in Section 5.4, a series of experiments have been conducted to demonstrate that application- and platform-agnostic runtime management can be achieved through the framework. Firstly, in Section 5.6.1, the three applications described in Section 5.4.1 have been profiled across the two platforms described in Section 5.4.3 to validate the application-agnostic properties of the framework. To demonstrate the flexibility of the runtime management layer in the framework, Section 5.6.2 presents validation of the operation of the two state-of-the-art approaches from Section 5.4.2, taking advantage of the cross-layer API to dynamically adjust and observe application and device knobs and monitors, respectively.

5.6.1 System Profiling and Trade-off Analysis

Design space exploration was performed using the profiling approach described in Section 5.4.2 as the runtime management layer on the three applications of Section 5.4.1 and across both platforms for the Jacobi application. For Jacobi, the dimensions of the matrix and vectors, N , is set to 4096. The error, performance and power characteristics of each application-platform

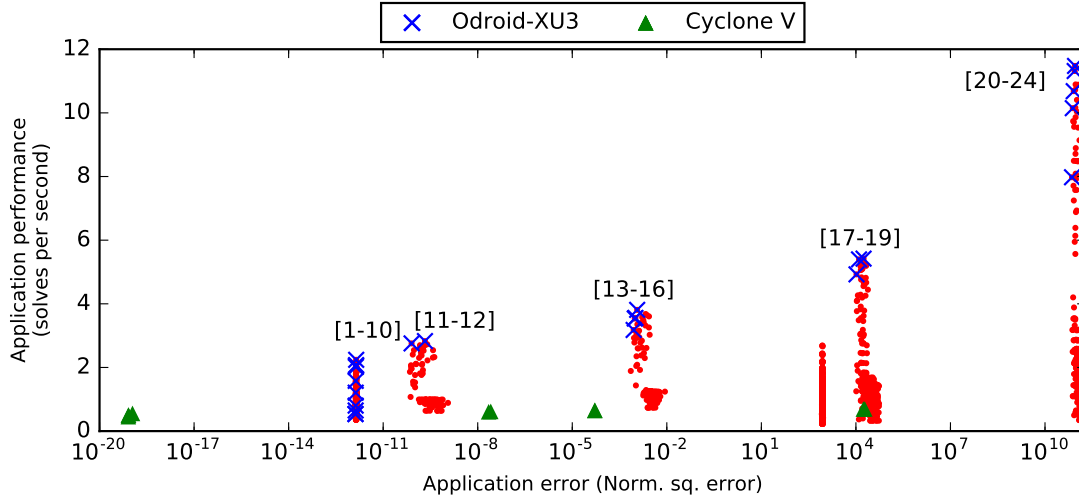


FIGURE 5.10: Pareto-optimal states for the error and performance trade-off of the Jacobi application on the Odroid-XU3 and Cyclone V. Label numbers link each operating point to the rows of Table 5.7, which show the knob and monitor values that each point encodes.

TABLE 5.7: Knob and monitor values for Pareto-optimal points of the Odroid platform in Figure 5.10.

No.	Error $\ Ax - b\ $	Time (ms)	Iters K	Prec. (bits)	Device	Frequency (MHz)		
						A7	A15	GPU
1	1.32×10^{-12}	0.537	8	double	GPU	1400	1200	420
2	1.33×10^{-12}	0.537	8	double	GPU	1400	1200	350
3	1.34×10^{-12}	0.649	6	double	GPU	1400	600	480
4	1.34×10^{-12}	0.801	5	double	GPU	1400	1100	266
5	1.34×10^{-12}	1.21	6	double	CPU	1400	800	GOV
6	1.34×10^{-12}	1.58	5	double	CPU	600	1200	GOV
7	1.36×10^{-12}	2.02	5	double	CPU	1200	1600	GOV
8	1.37×10^{-12}	2.05	5	double	CPU	1000	1800	GOV
9	1.38×10^{-12}	2.07	5	double	CPU	1400	1800	GOV
10	1.39×10^{-12}	2.24	5	double	CPU	1400	200	GOV
11	7.97×10^{-11}	2.76	4	double	CPU	600	200	GOV
12	2.12×10^{-10}	2.83	4	double	CPU	1400	200	GOV
13	8.86×10^{-4}	3.18	3	double	CPU	800	1600	GOV
14	9.60×10^{-4}	3.54	3	double	CPU	600	1800	GOV
15	1.01×10^{-3}	3.56	3	double	CPU	1400	1800	GOV
16	1.15×10^{-3}	3.81	3	double	CPU	600	200	GOV
17	1.04×10^4	4.93	2	double	CPU	1400	1600	GOV
18	1.26×10^4	5.40	2	double	CPU	600	200	GOV
19	1.75×10^4	5.42	2	double	CPU	1400	200	GOV
20	7.21×10^{10}	7.98	1	double	CPU	1000	1000	GOV
21	7.61×10^{10}	10.1	1	double	CPU	1000	1800	GOV
22	8.16×10^{10}	10.7	1	double	CPU	600	1600	GOV
23	8.75×10^{10}	11.3	1	double	CPU	1400	200	GOV
24	8.99×10^{10}	11.5	1	double	CPU	600	200	GOV

configuration were evaluated and the Pareto-optimal points were extracted to create the trade-offs shown in Figures 5.10, 5.12 and 5.13, with results for both platforms shown in Figure 5.10. Optimal points are highlighted as blue crosses for the Odroid XU3 and green triangles for the Cyclone V, with sub-optimal points shown as red dots.

Table 5.7 lists the application and device knob values for each of the labelled points in Figure 5.10, for the Odroid platform. The optimal points of Figure 5.10 are clustered in their error value, due to having the same number of iterations K , this confirms the behaviour predicted from (5.2). Furthermore, performance decreases as the number of iterations increases, due to the additional latency in computing $x[k + 1]$. The points within each cluster are distributed across a range of performances due to their different frequency settings. Power is not a dimension of importance in this trade-off therefore lower performance points are considered sub-optimal even if they lead to a better performance-per-watt. This trade-off is considered in 5.12 for the video decoder application. Device selection determines whether frequency control is activated. DVFS control is passed back to the default frequency governor for the GPU when it is not executing the application, shown as GOV in Table 5.7. The Cyclone V platform can achieve the lowest error levels (10^{-20} for 9 iterations at double precision using the FPGA-based kernel), however its performance is lower than the Odroid even at higher errors due to its limited frequency range and CPU cores with lower compute capabilities.

The runtime selection of Pareto-optimal operating points for the Jacobi application and Odroid platform was demonstrated by using a look-up table process in the runtime management layer. Figure 5.11 gives insight into the RTM's behaviour with the adjustment of knob settings over time. The knob settings at every operating point ensure that the predicted application monitor values were within their target ranges (*i.e.* $\text{perf}_{\min} \leq \text{perf}_{val|_t} \leq \text{perf}_{\max}$) and device monitors were optimised (*i.e.* minimum power and temperature). In order to sweep across a range of points in the operating space, the minimum bound of the performance monitor of the Jacobi application incremented by 0.40 solves per second every 40 seconds, shown in the top sub-plot of Figure 5.11. The value and upper bound of the error monitor are shown in the next sub-plot in the figure. The error value is free to fluctuate within the specified bounds if it allows other monitor values to be optimised (*i.e.* *power*). The next two sub-plots show the value of the application knobs, specifically iterations and device type. The data type knob remained set at double throughout the experiment and so it is not plotted. Iterations had a minor effect on performance and a value above 5 was sufficient to ensure that the error remained in the order of 10^{-12} , with 3 iterations being the least that ensured the maximum bound was met. The CPU and GPU frequencies (bottom sub-plot) had the strongest affect on performance and during periods of high performance requirement higher frequencies were used. A low performance bound allowed the reduction of the CPU or GPU frequency, leading to a reduction in power consumption. Finally, all the temperature monitors are plotted in the fifth sub-plot down in Figure 5.11, which all show a strong correlation with frequency and power consumption.

5.6.2 Runtime Manager Validation

Cross-layer connections created through the knobs and monitors of the framework increase the separability of the runtime management layer from the device and application layers. This promotes the development of interchangeable runtime management approaches that can be enabled

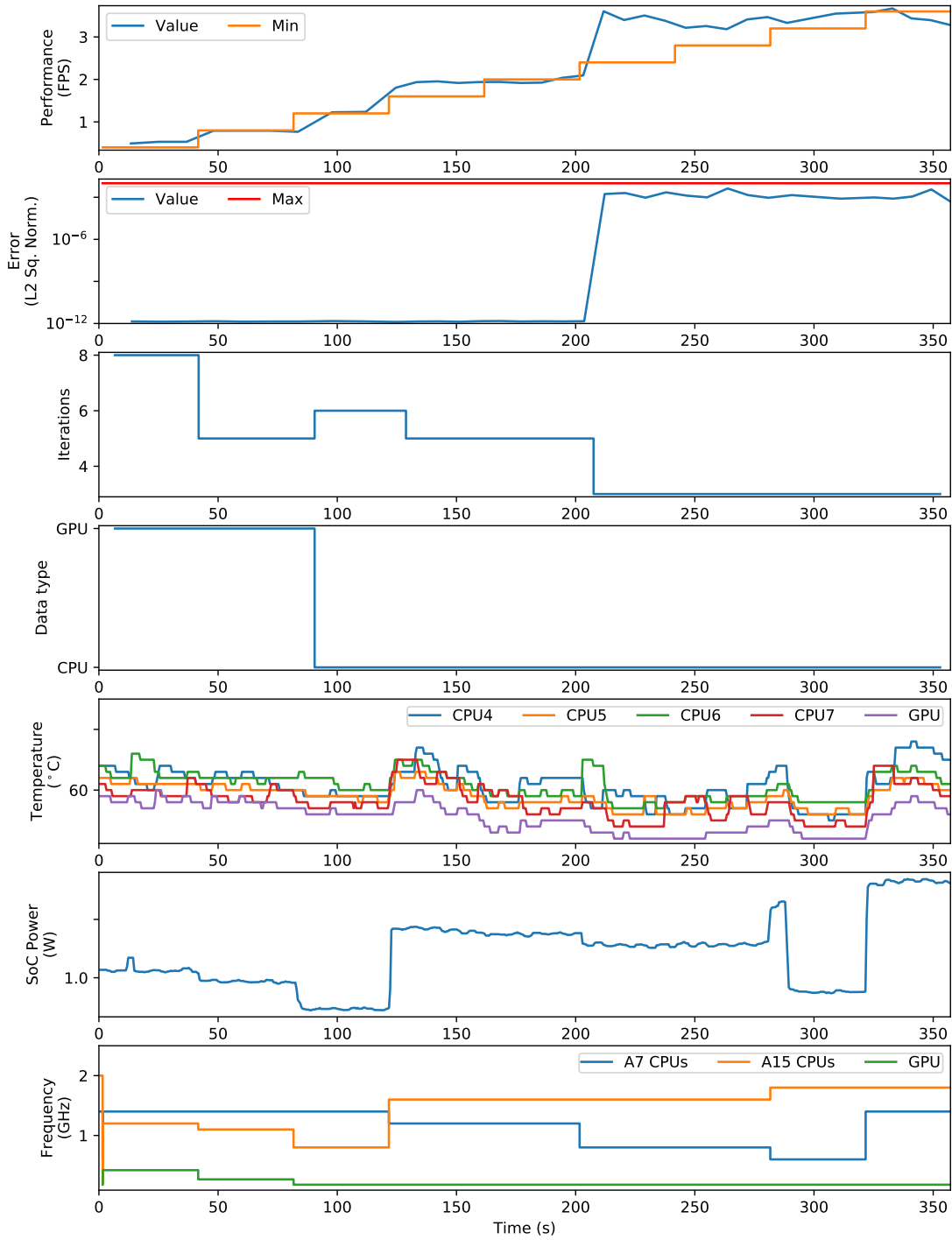


FIGURE 5.11: Runtime management layer updating application and device knob settings in response to step increases in the performance monitor minimum bound of the Jacobi application running on the Odroid platform. Application monitors and knobs are shown in graphs 1-2 and 3-4, respectively, with device monitors and knobs shown in 5-6 and 7, respectively. The RTM uses a look-up table with a control loop that filters it based on the monitor bounds.

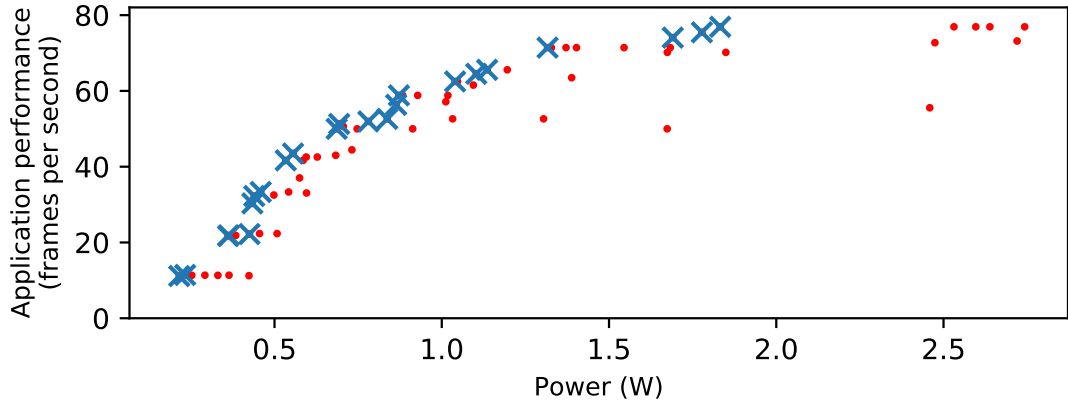


FIGURE 5.12: Pareto-optimal states identified by the RTM for the power and performance trade-off of the video decoder application on the Odroid-XU3.

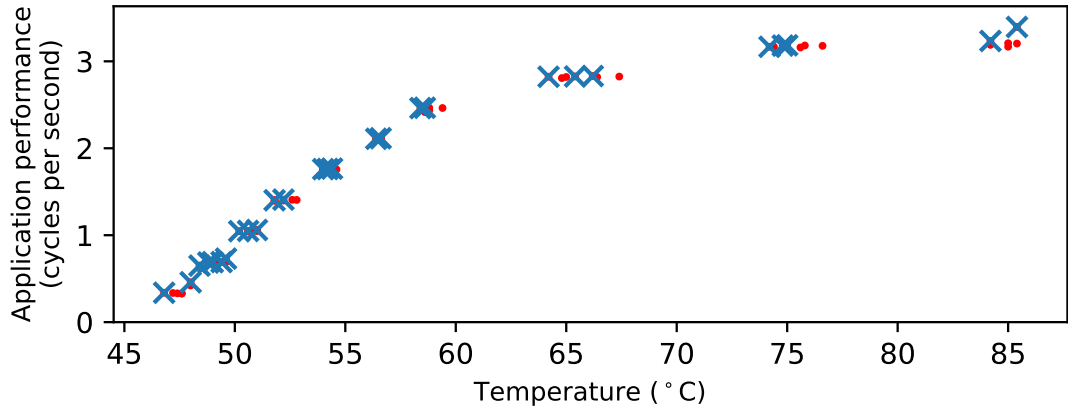


FIGURE 5.13: Pareto-optimal states identified by the RTM for the temperature and performance trade-off of the Whetstone application on the Odroid-XU3.

and disabled to suit the current workload. It has been validated that the two runtime management layers operate within the framework in the same way as in their source publications. The same platform and application (Odroid-XU3 and video decoder) are used to show that multiple runtime algorithms can provide energy savings. The experiment is repeated 50 times for each runtime algorithm, in order to establish the average energy saving.

5.6.2.1 Q-Learning RTM

Experimental validation of the operation of the Q-Learning algorithm described in Section 5.4.2.2 is now presented in Figure 5.14 in terms of the power consumption over time, while running the the video decoder application. Comparison was made against the Ondemand Linux frequency governor (red crosses). The two phases of the Q-Learning approach, exploration and exploitation, are highlighted, from 0 to 85 seconds and from 85 seconds onwards, respectively. A greater variation in power consumption was observed during the exploration phase due to a higher ϵ value as the Q-Table was being populated. In the exploitation phase, the variance in the system

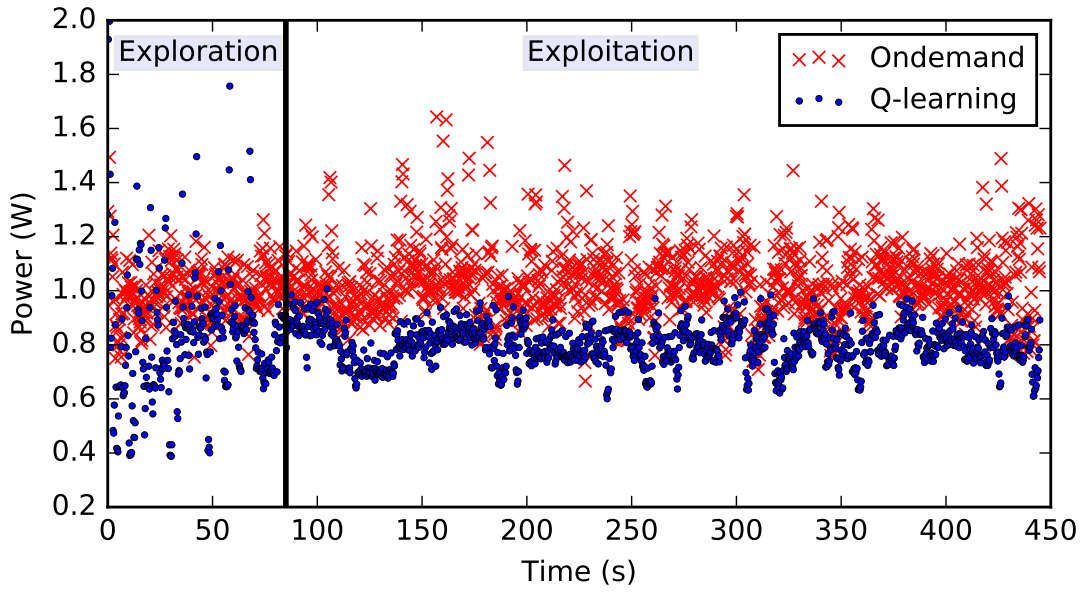


FIGURE 5.14: Comparison between Q-learning RTM and Ondemand governor for the power consumption of the video decoder executing on the Odroid.

state reduced as ϵ decreased and the power consumption was more consistent as a result. Fluctuations during the exploitation phase were in response to changes in the application processing requirements for specific frames.

The average energy per frame for the Q-learning RTM was 3.38mJ, lower than the Ondemand governor's 4.13mJ. Therefore, the RTM achieved an 18.2% energy saving across the program's whole execution. However, this came at a performance impact of 7.66% in terms of FPS.

5.6.2.2 PMC-based RTM

The PMC-based RTM described in Section 5.4.2.3 is now used as an alternative runtime management layer. The operation of the algorithm is demonstrated in the top part of Figure 5.15 in terms of the power consumption required to decode each frame. Comparison is made against the Ondemand frequency governor, which showed consistently higher average energy per frame. The RTM achieved an average energy saving of 17.2%, however this came at the cost of a performance impact of 6.90% FPS.

The calculated MRPI and measured frequency are shown in the middle and lower parts of Figure 5.15. This demonstrates the inverse relationship between MRPI and selected frequency, with a larger MRPI resulting in a lower core frequency being selected. This relationship comes from the knowledge that a higher MRPI correlates with a memory-bound workload, therefore computational speed can be reduced to match the memory bottleneck, and power consumption is reduced as a result.

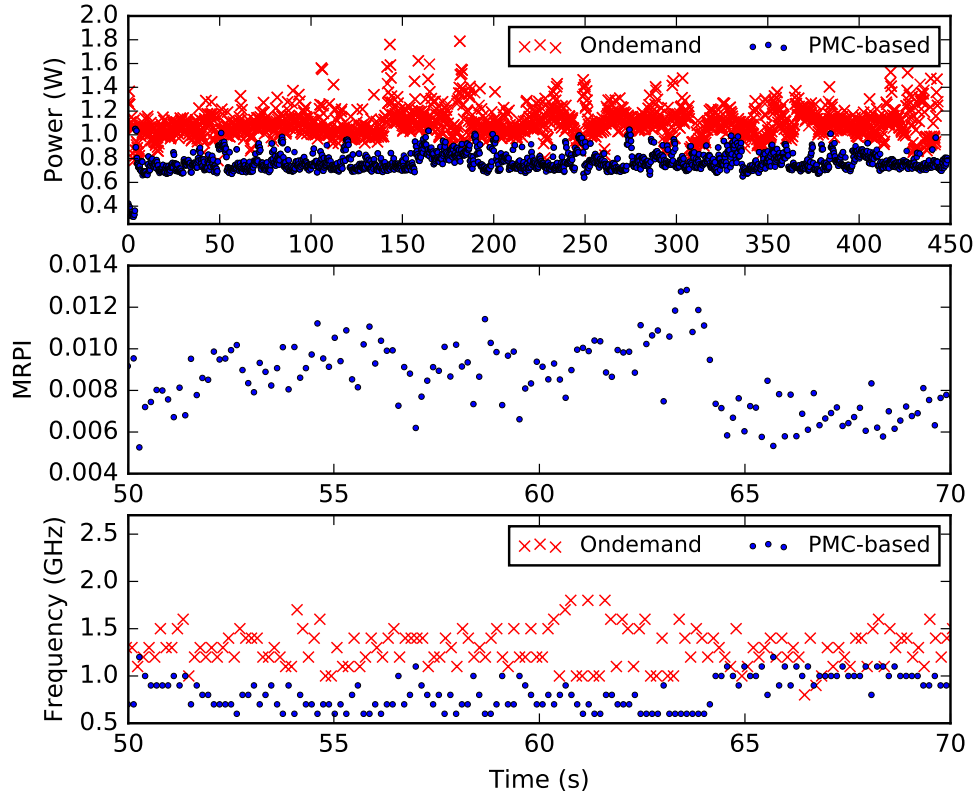


FIGURE 5.15: Top: Comparison between PMC-based RTM and Ondemand governor for the power consumption of the video decoder executing on the Odroid. Middle and lower: the MRPI metric and the frequency choices of the PMC-based RTM and Ondemand governor.

5.7 Summary

As heterogeneous embedded platforms increase in complexity, the need to meet application requirements and operate energy efficiently is the primary challenge in embedded system software design. In this chapter, a novel framework for the runtime management and co-optimisation of multiple objectives sought by concurrently executing applications has been presented, incorporating a cross-layer API, which exposes knobs and monitors between the system layers. This framework increases the mobility of applications, runtime management software and hardware platforms by providing a standardised interface for communication and control. Experimental validation of the framework was conducted using three applications on two heterogeneous platforms. Power, performance and accuracy trade-offs were shown for these application-platform scenarios to develop operating spaces that can be traversed at runtime.

Two state-of-the-art runtime management algorithms have been demonstrated operating through the framework, one based on reinforcement learning and the other on hardware performance metrics. The runtime managers achieved 18.2% and 17.2% energy savings, respectively, for a video decoding application compared to Ondemand Linux governor-controlled DVFS.

An open-source implementation of the framework has been released to provide a standard approach to runtime management and encourage other researchers to develop additional applications, runtime algorithms and support more platforms using the API. The framework enables

this research to be conducted more efficiently and thoroughly, including facilitating comparison between competing runtime algorithms.

In the future, additional applications, devices and runtime algorithms will be integrated with the aim of increasing the framework's validation and providing greater demonstration of the contributions of application- and platform-agnostic runtime management.

Chapter 6

Conclusion and Future Directions

6.1 Conclusions

Multi-core homogeneous and heterogeneous architectures are now pervasive in mobile and embedded systems. As a result, these systems are capable of much higher and more diverse levels of computation, including graphics processing and multi-tasking. Applications that require these high levels and specialised types of computation are utilising embedded platforms in order to deliver systems that are more portable and ergonomic than their predecessors.

However, limited energy supply in many environments presents a challenge to high performance mobile and embedded systems. Advancements in battery technology have not kept pace with increases in power consumption as a result of the greater computational capacity of SoCs. Consideration must now be given to where and when available energy is spent if a platform is to remain operational for a significant length of time.

Energy efficiency is of equal importance to performance capability in modern mobile and embedded platforms. To improve the energy efficiency of hardware, low power design features are included during development. Voltage and frequency scaling, power domains and more efficient cache designs provide some additional energy saving. Architectural advancements, such as multi-core and heterogeneous processors, enable the provision of more energy efficient processors as well as high performance cores. These architectures rely on operating systems to effectively implement task allocation and migration. In addition, operating systems integrate several low-level power saving operations that are performed at runtime to improve energy efficiency dynamically.

However, the behaviour of software applications cannot be predicted at design time and energy efficiency is often reduced by the sub-optimal use of resources by an application in a multi-core or heterogeneous platform. Applications commonly execute at a performance level above that required by the user or another dependent application, therefore energy is effectively wasted performing unnecessary computation. This can also result in the increase in temperature of mobile devices, which can impact on a users comfort and a platform's reliability. Therefore, management and optimisation of applications and platforms at runtime is required in order to maximise energy efficiency whilst delivering the required level of performance.

Many approaches exist to perform the management of applications and platforms at runtime, including controlling the level of parallelism and the allocation of application tasks to specific processing resources. Moreover, many of these use advanced machine learning algorithms to accurately predict the expected workload and behaviour of applications. However, a drawback of many of these approaches is their reliance on design-time information, often collected from profiling applications in advance. Clearly, this is not an acceptable model to use in modern mobile and embedded systems, where information on which applications will execute when and how cannot be predicted at design time.

This work has overcome this challenge by implementing a simple multiple linear regression modelling approach that can be trained at runtime in order to predict the power consumption of a multi-core platform and the performance of a multi-threaded application. MLR is chosen because a sufficiently accurate model can be built at runtime with a small number of variable parameters. Moreover, the approach can predict the optimal frequency and number of cores in order to meet an application performance requirement but minimise power consumption. A short training time is an essential requirement of this type of approach in order to deliver power optimisation within an acceptable window and also due to the short-lived nature of many applications.

In Chapter 3, MLR models have been presented for application latency and platform current. Translation to performance and power is made as they are not inherently linear functions of frequency and the number of cores. The model is built from training samples collected at runtime by testing a range of frequencies and numbers of cores. The error convergence of these models has been characterised to determine a minimum required training period of 16 samples. The model coefficients are statistically significant after this point and the models can therefore be used for prediction of power and performance. A gradient descent optimisation process is used to establish the optimum frequency and number of cores at runtime from the models. It is chosen because of its balance of accuracy and computational efficiency.

In Chapter 4, the runtime modelling and optimisation approach is applied to a proof-of-concept application and platform to demonstrate the management of energy on a multi-core system. Stereo Matching was chosen as an application that is both highly parallel and presents a significant workload. Stereo matching is the process of taking two or more images and estimating a 3D model of the scene by finding matching pixels in the images and converting their 2D positions into a depth value. It is an application found in potentially dynamic environments, such as autonomous driving, which can produce a fluctuating performance requirement. The disparity estimation algorithm is derived, as the core mathematical process performed in stereo matching, and analysed in terms of its accuracy against several standard metrics. A state-of-the-art algorithm is used to achieve both high accuracy and performance. However, the algorithm itself is not a contribution in this thesis.

The contribution is from the multi-threaded implementation of the algorithm, which is exposed as a control parameter, as well as the runtime management of the algorithm on the 60 core Intel Xeon Phi homogeneous multiprocessor platform. Experimental characterisation of the application is carried out on this platform, to expose the power and performance operating space. This shows that scaling and trade-off actions can be performed within the operating space at runtime, by changing the number of core used and their frequency. The runtime approach

is applied to the application and platform, building linear models of power and performance. Runtime adaptation to changes in the performance requirement is demonstrated, with significant savings compared to a linux frequency governor. This case study shows that there are energy saving opportunities that can only be identified through runtime modelling and optimisation.

The final contribution of this thesis comes from the establishment of a cross-layer framework and API for runtime management. The framework is designed to provide an environment where the process of runtime management is both application-agnostic and cross-platform. This will increase the adoption of runtime management techniques in future systems as it promotes plug-and-play style runtime management techniques, which change to suit the current workload. The cross-layer framework has been presented from its conceptual basis, introducing the *knob* and *monitor* constructs as generic methods for communicating information between applications, RTMs and devices. These concepts are supported by the specification of an API to enable cross-layer interactions and deliver the concepts of the framework. A software implementation of the framework provides additional tools for experimenting with runtime algorithms, applications and devices. The framework is validated as an essential component in the management of applications on heterogeneous multiprocessor systems with a series of case studies. The operation of existing runtime algorithms is demonstrated but now in an application- and platform-agnostic manner. This is a major step in extending runtime management to mobile and embedded platforms, where the combination of active applications is continuously changing and there are many permutations for the configuration of the platform.

6.2 Research Questions

Through the work presented in the preceding chapters, this thesis has aimed to address the research questions presented in Section 1.2. These questions are presented here, with a summary of how and where contributions have been made to address them:

1. *How can a multi-core system be modelled at runtime to optimise energy efficiency and performance requirements?*

Chapter 3 has demonstrated that an MLR modelling approach can produce accurate models of the power and performance of a system within 16-20 training samples. These models have then been used to perform runtime optimisation of power and performance with a gradient descent-based method, which determines the frequency and number of cores required to meet a performance target. The models have been developed in the context of a multi-core system considering the affect of core scaling on power consumption, using the Intel Xeon Phi platform in Chapter 4 and the Odroid-XU3 and Cyclone V in Chapter 5 as cases studies.

2. *What are the potential energy and performance trade-offs when applying runtime energy management to next-generation mobile and embedded applications?*

A stereo matching application has been developed as an example of a next-generation application that would make use of mobile and embedded platforms. In Chapter 4, the operating space of the application-platform combination has been explored to expose power and performance scaling and trade-offs. The runtime management approach from Chapter 3 was

introduced to perform energy management of the application and platform through runtime optimisation. The chapter has shown that a significant energy saving can be achieved when the RTM is able to use predictive models to adapt the system to changes in the performance target of the application during execution.

3. *How can the runtime management of energy and performance be made cross-platform and application-agnostic? How can this be extended to the management of other properties?*

The experimentation presented in Chapter 3 and 4 has demonstrated a case study of the runtime management of a specific application and platform, taking advantage of prior knowledge about the tunable parameters of each and the performance metric. To apply this approach to additional applications and platforms, including those that may be unknown a priori, the development of a application- and platform-agnostic environment for runtime management is the first logical step. Therefore, Chapter 5 has presented the creation of a cross-layer framework and API that enables application-independent and cross-platform runtime management by creating three distinct system layers, connected via dynamic *knobs* and *monitors*. A specification and implementation of this framework has been presented in order to enable the experimental validation of its fundamental concepts. Validation of two existing runtime management approaches as well as integration of three application and two platforms has shown that this framework can extend the capabilities of runtime algorithms to the optimisation of many system properties.

6.3 Future Investigations

This section concludes the thesis by outlining some potential future investigations to extend the work presented and provide additional research contributions and validation.

In this thesis, the development of a runtime management approach has been presented and a single application and platform have been used as its case study. The next logical step would be to validate the approach with additional applications which can also benefit from runtime energy management. A selection of such applications were presented in the literature review however, due to time constraints, these were not investigated. Modelling and optimising additional applications for mobile and embedded systems would increase the contributions of the runtime management approach. Another extension of this would be to investigate modelling multiple concurrently executing applications with an aim to optimising or balancing the objectives of each.

Furthermore, the development of the runtime management approach to model and optimise heterogeneous platforms would increase its contributions. Taking advantage of the cross-layer framework would expedite this process through the development of device classes for each platform. Due to time constraints, the MLR approach was not integrated into the framework for use in the runtime management layer. This step would enable further validation of the approach with applications and platforms that expose their tunable parameters through knobs and their performance metrics as monitors. These parameters can translate to the independent and dependent variables of the MLR models, so long as they exhibit linear behaviour. This would extend the RTM approach to make it application- and platform-agnostic.

The changing performance targets of the stereo matching application used as a demonstration of the runtime optimisation process were synthetically generated and driven manually from within the application. Extension of the application to use an external or content-based stimulus would make the approach more realistic and enable experimentation in a real-world environment. For example, in an autonomous vehicle the performance requirement would be governed by the vehicle's speed and the depth of objects in the scene. Creating a feedback loop in this situation would allow the application to present truly dynamic behaviour. Moreover, the development of the application to allow execution across the resources of a heterogeneous system would both improve the performance of the application but also present a new dimension to be considered by the RTM. This work is ongoing and so is not presented in this thesis but is expected to provide additional contributions.

Chapter 5 has presented the concept, specification and an implementation of the cross-layer framework and API. Continued validation and demonstration of the framework is required through example applications, RTMs and platforms to improve it and encourage adoption by the research community. Several properties of the framework are still to be demonstrated such as the co-optimisation of multiple objectives from different concurrently executing applications. It is expected that the environment presented by the framework will encourage greater focus on the development of advanced runtime management approaches, which focus on exploiting its application- and platform-agnostic properties.

Appendix A

RTM Modelling and Analysis Scripts

This appendix contains the scripts used to build and analyse the MLR models as well as test the gradient decent optimisation process. It also contains scripts used to analyse data from testing of the MLR RTM on the multi-core platform, which is shown in Chapter 3.

LISTING A.1: R code for MLR analysis and the plotting of Figure 3.3.

```
1 #####
  # linearModel.R
  # Created by Sheng Yang
  # Modified by Charles Leech
5  # Email: cl19g10 at ecs dot soton dot ac dot uk
  #####
  plotfile = function(x) pdf(paste('figs/',x,'.pdf',sep=''))

  library(data.table) #read.table
10 library(lattice) #wireframe
  library(ggplot2) #qplot

  #####
  # Reading data into data tables
15 #####
  dt = data.table(read.csv('data_lrrtm.txt', header=TRUE, sep=" "))
  dt = dt[,.(perf=mean(perf), energy=mean(energy)), by=.(freq,cores)] #avg repeat exps

  freq_list = c(619047, 666666, 714285, 761904, 857142, 952380, 1047618, 1142856, 1238094)
20 core_list = seq(4,60,4)
  d.fnv = read.table("freq.vdd.csv", header=TRUE, sep=",")
  vdd = d.fnv$vdd[findInterval(dt$freq, freq_list)]

  dts = dt[,.(freq, cores, perf, lat=1/perf, power=energy*perf, energy, vdd, vfc=vdd*freq*
    cores, period=1/freq, ppc=1/(freq*cores), current=(energy*perf)/vdd)]
25 #####
  # Plotting generated models in 3D
  #####

30 train_3d = dts[,.I[sample(1:nrow(dts),16)]]
  model.current = lm(current ~ vdd + vfc, data=dts[train_3d], weight=1/current)
  model.lat = lm(lat ~ period + ppc + cores, data=dts[train_3d], weight=1/lat)
  model.fnv = lm(vdd ~ freq, d=d.fnv)

35 x = seq(min(dts[,freq]),max(dts[,freq]), 1e4)
```

```

y = seq(min(dts[,cores]),max(dts[,cores]),1)
grid = expand.grid(x,y)
colnames(grid) = c('freq', 'cores')
grid.dts = data.table(grid)
40 vdd = predict(model.fnv, newdata=grid.dts)
grid.dts = grid.dts[,.(freq, cores, period=1/freq, ppc=1/(freq*cores), pc=1/cores, vfc=vdd*
    freq*cores, vf=vdd*freq, vc=vdd*cores, vdd=vdd, cores, freq)]
power.pred = predict(model.current, newdata=grid.dts)*vdd
perf.pred = 1/predict(model.lat, newdata=grid.dts)
#performance model
45 plot.model.perf = wireframe(perf.pred ~ (grid.dts[,freq]/1e6)*grid.dts[,cores],
    shade=TRUE,
    xlab=list('Frequency (GHz)', cex=1.5, rot=30),
    ylab=list('Num of Cores', cex=1.5, rot=-30),
    zlab=list('Performance (FPS)',cex=1.5, rot=90),
50 scales=list(arrows=FALSE, cex=1.5),
    par.settings = list(axis.line = list(col="transparent")))

#power model
plot.model.pwr = wireframe(power.pred ~ (grid.dts[,freq]/1e6)*grid.dts[,cores],
55 shade=TRUE,
    xlab=list('Frequency (GHz)', cex=1.5, rot=30),
    ylab=list('Num of Cores', cex=1.5, rot=-30),
    zlab=list('Power (W)',cex=1.5, rot=90),
    scales=list(arrows=FALSE, cex=1.5),
60 par.settings = list(axis.line = list(col="transparent")))

plotfile('model_perf')
print(plot.model.perf)
dev.off()
65 plotfile('model_pwr')
print(plot.model.pwr)
dev.off()

```

LISTING A.2: R code for MLR error convergence analysis and the plotting of Figure 3.4.

```

1 #####
# linearModel_samples.R
# Created by Charles Leech
# Email: cl19g10 at ecs dot soton dot ac dot uk
5 #####
plotfile = function(x) pdf(paste('figs/',x,'.pdf',sep=''))
library(data.table) #read.table
library(lattice) #wireframe
library(ggplot2) #qplot
10 #####
# Reading data into data tables
#####
dt = data.table(read.csv('data_lrrtm.csv', header=TRUE, sep=" "))
15 dt = dt[,.(perf=mean(perf), energy=mean(energy)), by=.(freq,cores)] #avg repeat exps

freq_list = c(619047, 666666, 714285, 761904, 857142, 952380, 1047618, 1142856, 1238094)
d.fnv = read.table("freq.vdd.csv", header=TRUE, sep=",")
vdd = d.fnv$vdd[findInterval(dt$freq, freq_list)]
20
dts = dt[,.(freq, cores, perf, lat=1/perf, power=energy*perf, energy, vdd, vfc=vdd*freq*
    cores, period=1/freq, ppc=1/(freq*cores), current=(energy*perf)/vdd)]

#####
# Model validation
25 #####
##Function to perform model creation, testing and validation
model.err = function(train,test){
    #model creation
    model.lat.v = lm(lat ~ period + ppc + cores, data=dts[train], weight=1/lat)
30 model.current.v = lm(current ~ vdd + vfc, data=dts[train], weight=1/current)
    #model prediction (testing)

```



```

lat.pred = predict(model.lat.v, newdata=dts[test])
pwr.pred = predict(model.current.v, newdata=dts[test])*dts[test]$vdd
eng.pred = predict(model.current.v, newdata=dts[test])*dts[test]$vdd*lat.pred
35 #model validation (error calculation)
lat.err.pc = abs(lat.pred - dts[test]$lat)/dts[test]$lat
pwr.err.pc = abs(pwr.pred - dts[test]$power)/dts[test]$power
eng.err.pc = abs(eng.pred - dts[test]$energy)/dts[test]$energy
return (c(mean(lat.err.pc), mean(pwr.err.pc)))
40 }

#####
# Test num samples vs error
#####
res = matrix(, nrow=0, ncol=5)
45 colnames(res) = c('samples', 'num_freq', 'num_cores', 'perf', 'power')
num_repeats = 40

for(j in c(8:40))
{
50 freq_levels = numeric(num_repeats)
core_levels = numeric(num_repeats)
errors = matrix(, nrow=0, ncol=2)
colnames(errors) = c('perf', 'power')

55 for(i in c(1:num_repeats))
{
train_data = dts[,.I[sample(1:nrow(dts),j)]]
test = -train_data
freq_levels[i] = nlevels(factor(dts[train_data]$freq))
60 core_levels[i] = nlevels(factor(dts[train_data]$cores))
errors = rbind(errors, model.err(train_data, test))
}
errors = data.table(errors)
freq_levels_mean = mean(freq_levels)
65 core_levels_mean = mean(core_levels)
errors_mean = c(mean(errors$perf), mean(errors$power))
res = rbind(res, c(j, freq_levels_mean, core_levels_mean, errors_mean))
}

70 res = data.table(res)

plot.model.err.samples = ggplot() +
geom_line(data=res, aes(x=samples, y=perf*100, color="Performance")) +
geom_line(data=res, aes(x=samples, y=power*100, color="Power")) +
75 xlab('Training samples') + ylab('Mean absolute error (%)') +
labs(color="Dependent variable") +
theme(legend.position="bottom", text=element_text(size=20)) +
theme(panel.background = element_rect(fill='white', colour='black')) +
theme(axis.text=element_text(colour='black')) +
80 theme(panel.grid.major = element_line(colour='black')) +

plotfile('model_err_samples')
print(plot.model.err.samples)
dev.off()

85
plot.model.err.samples_diff = ggplot() +
geom_line(data=res[-1], aes(x=samples, y=abs(diff(res$perf))*100, color="Performance")) +
geom_line(data=res[-1], aes(x=samples, y=abs(diff(res$power))*100, color="Power")) +
xlab('Training samples') + ylab('Mean absolute error difference (%)') +
90 labs(color="Dependent variable") +
theme(legend.position="bottom", text=element_text(size=20)) +
theme(panel.background = element_rect(fill='white', colour='black')) +
theme(axis.text=element_text(colour='black')) +
theme(panel.grid.major = element_line(colour='black'))

95
plotfile('model_err_samples_diff')
print(plot.model.err.samples_diff)
dev.off()

```

LISTING A.3: Complete R code for gradient decent analysis and the plotting of Figure 3.9 and 3.10.

```

1 #####
  # gd_opt.R
  # Created by Sheng Yang
  # Edited by Charles Leech
5 # Email: cl19g10 at ecs dot soton dot ac dot uk
#####
plotfile = function(x) pdf(paste('figs/',x,'.pdf',sep=''))
plotfile.png = function(x) png(paste('figs/',x,'.png',sep=''))

10 library(data.table) #read.table
   library(lattice) #wireframe
   library(ggplot2) #qplot

#####
15 # Reading data into data tables
#####
dt = data.table(read.csv('data_lrmt.csv', header=TRUE, sep=" "))
dt = dt[,.(perf=mean(perf), energy=mean(energy)), by=.(freq,cores)] #avg repeat exps

20 freq_list = c(619047, 666666, 714285, 761904, 857142, 952380, 1047618, 1142856, 1238094)
   core_list = seq(4,60,4)
   d.fnv = read.table("freq.vdd.csv", header=TRUE, sep=",")
   vdd = d.fnv$vdd[findInterval(dt$freq, freq_list)]

25 dts = dt[,.(freq, cores, perf, lat=1/perf, power=energy*perf, energy, vdd, vfc=vdd*freq*
   cores, period=1/freq, ppc=1/(freq*cores), current=(energy*perf)/vdd)]

   freq.sel = c(619047,857142,1142856,1238094)
   cores.sel = c(4,16,32,60)
30 train = dts[,.I[cores %in% cores.sel & freq %in% freq.sel]]
   #train = dts[,.I[sample(1:nrow(dts),20)]]
   test = -train #remove the above indicies from the data set used to test the models

   model.current = lm(current ~ vdd + vfc, data=dts[train], weight=1/current)
35 model.lat = lm(lat ~ period + ppc + cores, data=dts[train], weight=1/lat)
   model.fnv = lm(vdd ~ freq, d=d.fnv)

#####
# Optimization function with performance constraints
#####
40 opt_pwr = function(req_perf)
{
  #scalevi = function(v) v*c(714285,60)+c(619047,1)
  scalevi = function(v) v*c(619047,59)+c(619047,1)
45
  pred_pwr = function(x){
    x=scalevi(x)
    f=x[1];c=x[2]
    vdd = predict(model.fnv, data.frame(freq=f))
50    return(predict(model.current, data.frame(vdd,vfc = f*c*vdd^2))*vdd)
  }

  pred_perf = function(x){
    x=scalevi(x)
    f=x[1];c=x[2]
55    return(1/predict(model.lat,data.frame(period=1/f,ppc=1/(f*c),cores=c)))
  }

  dydx = function(fy,x,dx) c(fy(x+dx[,1])-fy(x),fy(x+dx[,2])-fy(x))/diag(dx)

60
  cnt=0
  x = c(0,0);
  alpha=(pred_perf(x+1)-pred_perf(x))/2;
  beta=0.5

```

```

65  dx= matrix(c(1e-3,0,0,1e-3),nrow=2,ncol=2);
    conv=c(1e-3,1e-3);
    done=0
    y1_lst=NULL; xc_lst=NULL; xf_lst=NULL; pwr_lst=NULL; perf_lst=NULL; alpha_lst=NULL;
    prev_dpdx=c(1,1)
70  repeat{
      dpdx = dydx(pred_perf,x,dx)
      repeat{
        repeat{
          xn = x+dpdx*alpha
          if(all(xn>0&xn<1)) {
85             break()
          }
          alpha = alpha*beta #reduce step size if xc is out of range
        }
        if(all(abs(xn-x)<conv)) {
          done=1; #delta x is within convergence threshold - finished
          break()
        }
        perf = pred_perf(xn)
        if(perf>req_perf){
          #reduce step size if perf is greater than perf_req
          alpha = alpha*beta
        }
90      else {
        pwr = pred_pwr(x)
        perf = pred_perf(x)
        y1_lst = c(y1_lst,perf)
        xsvi = scalevi(x)
95      xf_lst = c(xf_lst, xsvi[1])
        xc_lst = c(xc_lst, xsvi[2])
        pwr_lst=c(pwr_lst,pwr)
        perf_lst=c(perf_lst,perf)
        alpha_lst=c(alpha_lst,alpha)
100     x = xn;
        break()
      }
    }
    if(prev_dpdx %>% dpdx<0) {
105      alpha=alpha*beta
    }
    prev_dpdx=dpdx;
    if(done) break()
  }
110 if(length(y1_lst)==0){
  print('optimisation result is not found')
  return(data.frame(y1_lst=0, xc_lst=0, xf_lst=0, pwr_lst=0, perf_lst=0, req_perf, alpha_
    lst))
}
else{
115  print('optimisation result found')
  return(data.frame(y1_lst, xc_lst, xf_lst, pwr_lst, perf_lst, req_perf, alpha_lst))
}
}

120 #####
# Contour map for optimization with/without performance constraints
#####
pred_pwr2 = function(f,c){
  vdd = predict(model.fnv, data.frame(freq=f))
125  return(predict(model.current, data.frame(vdd,vfc = f*c*vdd))*vdd)
}

pred_perf2 = function(f,c){
  return(1/predict(model.lat,data.frame(period=1/f,ppc=1/(f*c),cores=c)))
130 }

```

```

f = seq(min(dts[,freq]),max(dts[,freq]), 1e4)
c = seq(min(dts[,cores]),max(dts[,cores]), 1)
#z = sapply(f, pred_pwr2, c=c)
135 z2 = sapply(f, pred_perf2, c=c)

#####
#with perf const
#####

140 plotfile('opt-pwr-c')
par(mar=c(5,6,2,2))
pc = opt_pwr(0.7)
dtbs.res = data.table(pc)
s = seq(nrow(dtbs.res)-1)

145 my_palette <- colorRampPalette(c("red", "yellow", "green"))(n = 299)
image(c,f*1e-6,-z2, col=my_palette, xlab='Cores', ylab='Frequency (GHz)', cex.lab=2, cex.
      axis=2)
contour(c,f*1e-6, z2, lwd=3, labcex=1.3, method='flattest', add=TRUE)
arrows(dtbs.res[s,xc_1st], dtbs.res[s,xf_1st]*1e-6, dtbs.res[s+1,xc_1st], dtbs.res[s+1,xf_
      1st]*1e-6, col = 'blue', length = 0.15, lwd=2)
150 points(dtbs.res[,xc_1st], dtbs.res[,xf_1st]*1e-6, pch=10, cex=2, col='blue')

dev.off()

pwr_max = max(dtbs.res$pwr_1st)
155 dtbs.res[,pwr_norm:=pwr_1st/pwr_max]
perf_max = max(dtbs.res$perf_1st)
dtbs.res[,perf_norm:=perf_1st/perf_max]

print(dtbs.res)

160 step_size_c = ggplot() +
  geom_line(data=dtbs.res,aes(x=seq(1:nrow(dtbs.res)), y=alpha_1st, color="Step size")) +
  geom_line(data=dtbs.res,aes(x=seq(1:nrow(dtbs.res)), y=pwr_norm, color="Noramlised
    Predicted Power")) +
  geom_line(data=dtbs.res,aes(x=seq(1:nrow(dtbs.res)), y=perf_norm, color="Noramlised
    Predicted Performance")) +
  xlab('Iteration') + ylab("") +
  labs(color="Parameter") +
  theme(legend.position="bottom", legend.direction="vertical",text=element_text(size=20)) +
  theme(panel.background = element_rect(fill='white',colour='black')) +
  theme(axis.text=element_text(colour='black')) +
170 theme(panel.grid.major = element_line(colour='black'))

plotfile('step_size_c')
print(step_size_c)
dev.off()

175 #####
#without perf const
#####

plotfile('opt-pwr-nc')
180 par(mar=c(5,6,2,2))
pnc = opt_pwr(1)
dtbs.res = data.table(pnc)
s = seq(nrow(dtbs.res)-1)

185 image(c,f*1e-6,-z2, col=my_palette, xlab='Cores', ylab='Frequency (GHz)', cex.lab=2, cex.
      axis=2)
contour(c,f*1e-6, z2, lwd=3, labcex=1.3, method='flattest', add=TRUE)
arrows(dtbs.res[s,xc_1st], dtbs.res[s,xf_1st]*1e-6, dtbs.res[s+1,xc_1st], dtbs.res[s+1,xf_
      1st]*1e-6, col = 'blue', length = 0.15, lwd=2)
points(dtbs.res[,xc_1st], dtbs.res[,xf_1st]*1e-6, pch=10, cex=2, col='blue')

190 dev.off()

pwr_max = max(dtbs.res$pwr_1st)
dtbs.res[,pwr_norm:=pwr_1st/pwr_max]

```

```

perf_max = max(dtbs.res$perf_1st)
195 dtbs.res[,perf_norm:=perf_1st/perf_max]

step_size_nc = ggplot() +
  geom_line(data=dtbs.res,aes(x=seq(1:nrow(dtbs.res)), y=alpha_1st, color="Step size")) +
  geom_line(data=dtbs.res,aes(x=seq(1:nrow(dtbs.res)), y=pwr_norm, color="Noramlised
    Predicted Power")) +
200 geom_line(data=dtbs.res,aes(x=seq(1:nrow(dtbs.res)), y=perf_norm, color="Noramlised
    Predicted Performance")) +
  xlab('Iteration') + ylab("") +
  labs(color="Parameter") +
  theme(legend.position="bottom", legend.direction="vertical",text=element_text(size=20)) +
  theme(panel.background = element_rect(fill='white',colour='black')) +
205 theme(axis.text=element_text(colour='black')) +
  theme(panel.grid.major = element_line(colour='black'))

plotfile('step_size_nc')
print(step_size_nc)
210 dev.off()

```

LISTING A.4: Python script to plot experimental characterisation data for Figure 3.2, 4.15 and 4.16.

```

1 import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from matplotlib.lines import Line2D
5 import math

NUM_FREQS = 9
NUM_CORES = 15
NUM_SAMPLES = 5 #length of thread_idx_set
10 freq_series = np.zeros([NUM_FREQS, NUM_CORES],dtype=[('F',int),('C',int),('perf', float),('E', float),('S', float)])
freq_series_sample = np.zeros([NUM_FREQS, NUM_SAMPLES],dtype=[('F',int),('C',int),('perf', float),('E', float),('S', float)])

filename="de_phi_profile_upto60_v2.csv"
15 fx_ex = np.genfromtxt(filename, delimiter=' ', names=True) #Read in csv data from the file
#freq_set=[619047, 666666, 714285, 761904, 857142, 952380, 1047618, 1142856, 1238094]
freq_set=[619, 667, 714, 762, 857, 952, 1048, 1143, 1238]
thread_idx_set = [0,1,3,7,14] #to pick out profile entries at 4,8,16,32,60
markers = Line2D.filled_markers
20
# Create a figure, specific size & dots per inch
fig1 = plt.figure(figsize=(10, 10), dpi=128)
scatter_pe = fig1.add_subplot(111)
scatter_pe.set_xlabel('Average Energy per Frame (J)', fontsize=16)
25 scatter_pe.set_ylabel('Average Performance (FPS)', fontsize=16)
#scatter_pe.set_title('Performance & Energy Points', fontsize=16)
scatter_pe.tick_params(axis='y', labelsize=16)
scatter_pe.tick_params(axis='x', labelsize=16)
scatter_pe.grid(True)
30
# Create a figure, specific size & dots per inch
fig2 = plt.figure(figsize=(20, 8), dpi=128)
line_ceng = fig2.add_subplot(111)
line_ceng.set_xlabel('Cores', fontsize=24)
35 line_ceng.set_ylabel('Average Energy per Frame (J)', fontsize=24)
#line_ceng.set_title('Energy Consumption with Thread Scaling', fontsize=20)
line_ceng.tick_params(axis='y', labelsize=24)
line_ceng.tick_params(axis='x', labelsize=24)
line_ceng.grid(True)
40 #Log scale settings:
#line_ceng.set_xscale('log', basex=2)
line_ceng.xaxis.set_major_formatter(ticker.ScalarFormatter())

```

```

line_ceng.xaxis.set_major_formatter(ticker.FormatStrFormatter('%0d'))

45 # Create a figure, specific size & dots per inch
fig3 = plt.figure(figsize=(20, 8), dpi=128)
line_cperf = fig3.add_subplot(111)
line_cperf.set_xlabel('Cores', fontsize=24)
line_cperf.set_ylabel('Performance (FPS)', fontsize=24)
50 #line_cperf.set_title('Performance with Thread Scaling', fontsize=20)
line_cperf.tick_params(axis='y', labels=24)
line_cperf.tick_params(axis='x', labels=24)
line_cperf.grid(True)
#Log scale settings:
55 #line_cperf.set_xscale('log', basex=2)
line_cperf.xaxis.set_major_formatter(ticker.ScalarFormatter())
line_cperf.xaxis.set_major_formatter(ticker.FormatStrFormatter('%0d'))

# Create a figure, specific size & dots per inch
60 fig4 = plt.figure(figsize=(10, 10), dpi=128)
line_sup = fig4.add_subplot(111)
line_sup.set_xlabel('Cores', fontsize=16)
line_sup.set_ylabel('Speedup in Performance', fontsize=16)
#line_sup.set_title('Speed-up with Thread Scaling', fontsize=20)
65 line_sup.tick_params(axis='y', labels=16)
line_sup.tick_params(axis='x', labels=16)
line_sup.grid(True)
#Log scale settings:
line_sup.set_xscale('log', basex=2)
70 line_sup.xaxis.set_major_formatter(ticker.ScalarFormatter())
line_sup.xaxis.set_major_formatter(ticker.FormatStrFormatter('%0d'))

# Create a figure, specific size & dots per inch
fig5 = plt.figure(figsize=(20, 8), dpi=128)
75 line_cpwr = fig5.add_subplot(111)
line_cpwr.set_xlabel('Cores', fontsize=24)
line_cpwr.set_ylabel('Power (W)', fontsize=24)
#line_cpwr.set_title('Power with Thread Scaling', fontsize=20)
line_cpwr.tick_params(axis='y', labels=24)
80 line_cpwr.tick_params(axis='x', labels=24)
line_cpwr.grid(True)
#Log scale settings:
#line_cpwr.set_xscale('log', basex=2)
line_cpwr.xaxis.set_major_formatter(ticker.ScalarFormatter())
85 line_cpwr.xaxis.set_major_formatter(ticker.FormatStrFormatter('%0d'))

# Create a figure, specific size & dots per inch
fig6 = plt.figure(figsize=(12, 12), dpi=128)
scatter_pp = fig6.add_subplot(111)
90 scatter_pp.set_xlabel('Average Power per Frame (W)', fontsize=16)
scatter_pp.set_ylabel('Average Performance (FPS)', fontsize=16)
#scatter_pp.set_title('Performance & Power Points', fontsize=16)
scatter_pp.tick_params(axis='y', labels=16)
scatter_pp.tick_params(axis='x', labels=16)
95 scatter_pp.grid(True)

max_threads = 60
#Data table creation
j=0
100 for freq_idx in range (0,NUM_FREQS,1):
    i=0
    for thread_idx in range (0,NUM_CORES,1):

        p1 = fx_ex[j]['perf']
105         e1 = fx_ex[j]['energy']
        j+=1
        p2 = fx_ex[j]['perf']
        e2 = fx_ex[j]['energy']
        j+=1
110         p3 = fx_ex[j]['perf']

```

```

        e3 = fx_ex[j]['energy']
        j+=1
        p4 = fx_ex[j]['perf']
        e4 = fx_ex[j]['energy']
115     j+=1

        perf_av = (p1 + p2 + p3 + p4) / 4
        e_av     = (e1 + e2 + e3 + e4) / 4

120     freq_series[NUM_FREQS-1-freq_idx][NUM_CORES-1-i] = (freq_set[NUM_FREQS-1-freq_idx],
                                                             max_threads-(4*thread_idx), perf_av, e_av,
0.0)
        i+=1

125 #Speed-up calculation
    for freq_idx in range (0,NUM_FREQS,1):
        for threads in range (0,NUM_CORES,1):
            freq_series[freq_idx][threads]['S'] = \
            freq_series[freq_idx][threads]['perf']/freq_series[freq_idx][0]['perf']
130
        freq_series_sample[freq_idx] = freq_series[freq_idx][thread_idx_set]

    #Plot Graphs
    #scatter_pe & scatter_pp
135 for f in range(0,NUM_FREQS):
        scatter_pe.plot(freq_series_sample[f]['E'],freq_series_sample[f]['perf'], label=str(
            freq_set[f]),
                        marker=markers[f], linestyle='None',markersize=10, color=plt.cm.jet(f
/10.))
        for k in range(0,NUM_SAMPLES):
            scatter_pe.annotate(freq_series_sample[f]['C'][k], xy=(freq_series_sample[f]['E
'] [k],freq_series_sample[f]['perf'][k]),
140                        xytext=(-10,5), textcoords='offset points', fontsize=14)

    #scatter_pp
    for f in range(0,NUM_FREQS):
        scatter_pp.plot(freq_series[f]['E']*freq_series[f]['perf'],freq_series[f]['perf'],
            label=str(freq_set[f]),
145                        marker=markers[f], linestyle='None',markersize=10, color=plt.cm.jet(f
/10.))
        for k in range(0,NUM_CORES):
            if k in thread_idx_set:
                if k in [0,1,14]:
                    scatter_pp.annotate(freq_series[f]['C'][k], xy=(freq_series[f]['E'][k]*
freq_series[f]['perf'][k],freq_series[f]['perf'][k]),
150                    xytext=(15,-20), textcoords='offset points', arrowprops=dict(
                        arrowstyle='->',color='k'), fontsize=18)
                else:
                    scatter_pp.annotate(freq_series[f]['C'][k], xy=(freq_series[f]['E'][k]*
freq_series[f]['perf'][k],freq_series[f]['perf'][k]),
                    xytext=(-40,15), textcoords='offset points', arrowprops=dict(
                        arrowstyle='->',color='k'), fontsize=18)

155     line_ceng.plot(freq_series[f]['C'], freq_series[f]['E'], label=str(freq_set[f]),
                    marker=markers[f], markersize=10, color=plt.cm.jet(f/10.))
    line_cperf.plot(freq_series[f]['C'], freq_series[f]['perf'], label=str(freq_set[f]),
                    marker=markers[f], markersize=10, color=plt.cm.jet(f/10.))
    line_sup.plot(freq_series[f]['C'], freq_series[f]['S'], label=str(freq_set[f]),
160                    marker=markers[f], markersize=10, color=plt.cm.jet(f/10.))
    line_cpwr.plot(freq_series[f]['C'], freq_series[f]['E']*freq_series[f]['perf'], label=
str(freq_set[f]),
                    marker=markers[f], markersize=10, color=plt.cm.jet(f/10.))

    legend = scatter_pe.legend(title='Frequency (MHz)', numpoints=1, fontsize=16)
165 plt.setp(legend.get_title(),fontsize=16)
    legend = line_ceng.legend(loc=1, title='Frequency (MHz)', ncol=2, numpoints=1, fontsize=22)
    plt.setp(legend.get_title(),fontsize=22)

```

```
    legend = line_cperf.legend(loc=2, title='Frequency (MHz)', ncol=2, numpoints=1, fontsize
        =22)
    plt.setp(legend.get_title(), fontsize=22)
170 legend = line_sup.legend(loc=2, title='Frequency (MHz)', numpoints=1, fontsize=16)
    plt.setp(legend.get_title(), fontsize=16)
    legend = line_cpwr.legend(loc=2, title='Frequency (MHz)', ncol=2, numpoints=1, fontsize=21)
    plt.setp(legend.get_title(), fontsize=21)
    legend = scatter_pp.legend(loc= 2, title='Frequency (MHz)', numpoints=1, fontsize=16)
175 plt.setp(legend.get_title(), fontsize=16)

fig1.savefig("scatter_pe.pdf", bbox_inches='tight')
fig2.savefig("line_ceng_wide.pdf", bbox_inches='tight')
fig3.savefig("line_cperf_wide.pdf", bbox_inches='tight')
180 fig4.savefig("line_sup.pdf", bbox_inches='tight')
fig5.savefig("line_cpwr_wide.pdf", bbox_inches='tight')
fig6.savefig("scatter_pp.pdf", bbox_inches='tight')
```

Appendix B

RTM Experimental Data

This appendix contains the data collected during experimentation with the MLR RTM. This includes data required for figures shown in Chapter 3.

TABLE B.1: Characterisation data for a multi-core platform and multi-threaded application, including the power consumption and performance, in FPS, resulting from a particular frequency and number of active cores.

Freq (MHz)	Cores	Perf (FPS)	Energy (J)	Freq (MHz)	Cores	Perf (FPS)	Energy (J)
1238094	60	0.8663	66.989175	1142856	16	0.4982	65.40915
1238094	56	0.897025	61.622	1142856	12	0.3535	85.9661
1238094	52	0.9056	59.396575	1142856	8	0.2661	98.457375
1238094	48	0.92965	56.420625	1142856	4	0.13835	173.24945
1238094	44	0.952475	54.135825	1047618	60	0.7251	69.20685
1238094	40	0.96195	52.2118	1047618	56	0.750475	66.12865
1238094	36	0.946575	50.435625	1047618	52	0.7712	61.586075
1238094	32	0.92915	50.124025	1047618	48	0.792875	57.212925
1238094	28	0.718175	59.92485	1047618	44	0.81715	54.436425
1238094	24	0.661275	61.262275	1047618	40	0.818475	52.9134
1238094	20	0.551725	68.934125	1047618	36	0.81855	51.604775
1238094	16	0.500925	71.133075	1047618	32	0.817475	49.87715
1238094	12	0.378525	84.535225	1047618	28	0.615575	60.435525
1238094	8	0.28705	104.864575	1047618	24	0.6021	59.21275
1238094	4	0.149325	166.979925	1047618	20	0.47435	67.30995
1142856	60	0.78415	67.3229	1047618	16	0.460425	66.5866
1142856	56	0.813475	63.553075	1047618	12	0.31365	85.21785
1142856	52	0.830775	61.40195	1047618	8	0.245525	99.35515
1142856	48	0.863375	58.271225	1047618	4	0.127625	180.60135
1142856	44	0.8425	57.116525	952380	60	0.66305	66.562975
1142856	40	0.858425	53.38345	952380	56	0.675525	63.73585
1142856	36	0.85935	52.741175	952380	52	0.70755	60.9971
1142856	32	0.8559	51.3179	952380	48	0.721175	58.084075
1142856	28	0.6669	60.374475	952380	44	0.741125	55.294925
1142856	24	0.652575	59.4399	952380	40	0.746825	51.743175
1142856	20	0.520225	68.4353	952380	36	0.748375	50.38645

TABLE B.2: Characterisation data continued from Table B.1.

Freq (MHz)	Cores	Perf (FPS)	Energy (J)	Freq (MHz)	Cores	Perf (FPS)	Energy (J)
952380	32	0.74815	49.04695	714285	48	0.5194	58.030875
952380	28	0.561625	58.68805	714285	44	0.519025	54.8775
952380	24	0.54975	57.022025	714285	40	0.5302	51.69875
952380	20	0.437575	66.0541	714285	36	0.52605	50.7656
952380	16	0.417175	63.69095	714285	32	0.512925	49.630175
952380	12	0.295125	78.877675	714285	28	0.388875	57.94205
952380	8	0.22045	103.246975	714285	24	0.359775	60.73015
952380	4	0.109925	182.19525	714285	20	0.299875	68.849175
857142	60	0.59955	67.459975	714285	16	0.281675	70.465725
857142	56	0.615125	65.15485	714285	12	0.205375	81.258975
857142	52	0.62015	62.589275	714285	8	0.151525	98.362725
857142	48	0.63305	59.033025	714285	4	0.071275	194.794425
857142	44	0.607925	59.112425	666666	60	0.447675	66.620025
857142	40	0.62935	56.058525	666666	56	0.465125	61.84465
857142	36	0.6183	54.5878	666666	52	0.45385	60.216675
857142	32	0.603375	53.20795	666666	48	0.483325	55.890725
857142	28	0.46435	62.9281	666666	44	0.472375	54.4853
857142	24	0.452925	60.639	666666	40	0.4875	51.318225
857142	20	0.350675	70.21145	666666	36	0.489125	49.69305
857142	16	0.32715	71.393725	666666	32	0.455875	50.476175
857142	12	0.241225	91.70645	666666	28	0.348325	60.174975
857142	8	0.174575	117.18765	666666	24	0.33905	60.52265
857142	4	0.090625	175.015975	666666	20	0.283825	67.67185
761904	60	0.49435	69.12715	666666	16	0.266975	64.255675
761904	56	0.50825	66.556375	666666	12	0.191925	76.798125
761904	52	0.516575	62.759925	666666	8	0.141175	99.125025
761904	48	0.53635	59.853275	666666	4	0.072675	181.007075
761904	44	0.56345	56.021575	619047	60	0.418075	61.057325
761904	40	0.55945	54.58505	619047	56	0.429175	58.379925
761904	36	0.53795	53.5214	619047	52	0.439825	54.92195
761904	32	0.525425	52.301275	619047	48	0.44945	52.189925
761904	28	0.4169	59.6411	619047	44	0.460375	49.778025
761904	24	0.404575	57.895225	619047	40	0.459775	48.345425
761904	20	0.32415	68.124925	619047	36	0.4537	48.39805
761904	16	0.2867	74.201575	619047	32	0.451275	46.8052
761904	12	0.21785	87.855875	619047	28	0.341175	56.277575
761904	8	0.155375	99.35575	619047	24	0.330225	54.3001
761904	4	0.083175	169.458975	619047	20	0.264975	59.855325
714285	60	0.482175	65.648675	619047	16	0.243325	59.42775
714285	56	0.494725	64.320425	619047	12	0.17905	76.260625
714285	52	0.4751	63.132975	619047	8	0.132125	101.369325
				619047	4	0.0631	187.34855

Appendix C

Stereo Matching Algorithm Code

This appendix contains sample code for the stereo matching algorithm. This was used to conduct the experiments described in Chapter 4. The full, updated application code can be found at: <https://github.com/PRiME-project/PRiMEStereoMatch>.

LISTING C.1: Top-level code for the disparity estimation algorithm.

```
1  /*-----
    DispEst.cpp - Disparity Estimation Class
    -----
    Author: Charles Leech
5   Email: cl19g10 [at] ecs.soton.ac.uk
    -----*/

#include "DispEst.h"

float get_rt_de(){
10  struct timespec realtime;
    clock_gettime(CLOCK_MONOTONIC,&realtime);
    return (float)(realtime.tv_sec*1000000+realtime.tv_nsec/1000);
}

15 DispEst::DispEst(SVImage l, SVImage r, const int d, int t)
    : lImg(l), rImg(r), maxDis(d), threads(t)
{
    printf("Disparity Estimation for Stereo Matching\n" );
    int retval = 0;
20
    hei = lImg.height;
    wid = lImg.width;

    lcostVol = new float*[maxDis];
25  for (int i = 0; i < maxDis; i++) {
        retval = posix_memalign((void **)&lcostVol[i], 64, hei*wid*sizeof(float));
    }
    rcostVol = new float*[maxDis];
    for (int i = 0; i < maxDis; i++) {
30     retval = posix_memalign((void **)&rcostVol[i], 64, hei*wid*sizeof(float));
    }

    retval = posix_memalign((void **)&lGray, 64, hei*wid*sizeof(float));
    retval = posix_memalign((void **)&rGray, 64, hei*wid*sizeof(float));
35  retval = posix_memalign((void **)&lGrdX, 64, hei*wid*sizeof(float));
    retval = posix_memalign((void **)&rGrdX, 64, hei*wid*sizeof(float));

    lImg_rgb = new float*[3];
    rImg_rgb = new float*[3];
```

```

40  mean_lImg = new float*[3];
    mean_rImg = new float*[3];
    var_lImg = new float*[6];
    var_rImg = new float*[6];

45  for (int i = 0; i < 6; i++) {
        if(i < 3){
            retval = posix_memalign((void **)&lImg_rgb[i], 64, hei*wid*sizeof(float));
            retval = posix_memalign((void **)&rImg_rgb[i], 64, hei*wid*sizeof(float));
            retval = posix_memalign((void **)&mean_lImg[i], 64, hei*wid*sizeof(float));
50      retval = posix_memalign((void **)&mean_rImg[i], 64, hei*wid*sizeof(float));
        }
        retval = posix_memalign((void **)&var_lImg[i], 64, hei*wid*sizeof(float));
        retval = posix_memalign((void **)&var_rImg[i], 64, hei*wid*sizeof(float));
    }

55  lDisMap = (uch*)malloc(hei*wid);
    rDisMap = (uch*)malloc(hei*wid);

    constructor = new CVC();
60  filter = new CVF(&lImg, maxDis);
    selector = new DispSel();
    postProcessor = new PP();
}

65 DispEst::~DispEst(void)
{
    for (int i = 0; i < maxDis; i++) {
        free(lcostVol[i]);
        free(rcostVol[i]);
70    }
    free(lcostVol);
    free(rcostVol);

    free(lGray);
75    free(rGray);
    free(lGrdX);
    free(rGrdX);

    for (int i = 0; i < 6; i++) {
80        if(i < 3)
        {
            free(lImg_rgb[i]);
            free(rImg_rgb[i]);
            free(mean_lImg[i]);
85        free(mean_rImg[i]);
        }
        free(var_lImg[i]);
        free(var_rImg[i]);
    }

90    free(lImg_rgb);
    free(rImg_rgb);
    free(mean_lImg);
    free(mean_rImg);
    free(var_lImg);
95    free(var_rImg);

    free(lDisMap);
    free(rDisMap);

100    delete constructor;
    delete filter;
    delete selector;
    delete postProcessor;
}

105 void DispEst::CostConst()
{

```

```

//Set up threads and thread attributes
void *status;
110 pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
pthread_t BCV_threads[maxDis];
buildCV_TD buildCV_TD_Array[maxDis];

115 //printf("Cost Construction Underway...\n");
constructor->buildGS(&lImg, &rImg, lGray, rGray);
constructor->buildGrdX(lGray, rGray, wid, hei, lGrdX, rGrdX);

120 //Thread in blocks of threads. Join all threads in each block before continuing.
for(int level = 0; level <= maxDis/threads; level++)
{
    //Handle remainder if threads is not power of 2.
    int block_size = (level < maxDis/threads) ? threads : (maxDis%threads);

125 for(int iter=0; iter < block_size; iter++)
    {
        int d = level*threads + iter;
        buildCV_TD_Array[d] = {lImg, rImg, lGrdX, rGrdX, d, lcostVol[d]};
130 pthread_create(&BCV_threads[d], &attr, CVC::buildCV_left_thread, (void *)&
        buildCV_TD_Array[d]);
    }
    for(int iter=0; iter < block_size; iter++)
    {
        int d = level*threads + iter;
135 pthread_join(BCV_threads[d], &status);
        //printf("Joining BCV thread %d\n", d);
    }
    for(int iter=0; iter < block_size; iter++)
    {
        int d = level*threads + iter;
140 buildCV_TD_Array[d] = {rImg, lImg, rGrdX, lGrdX, d, rcostVol[d]};
        pthread_create(&BCV_threads[d], &attr, CVC::buildCV_right_thread, (void *)&
        buildCV_TD_Array[d]);
    }
    for(int iter=0; iter < block_size; iter++)
145 {
        int d = level*threads + iter;
        pthread_join(BCV_threads[d], &status);
        //printf("Joining BCV thread %d\n", d);
    }
150 }
}

void DispEst::CostFilter()
{
155 //Set up threads and thread attributes
void *status;
pthread_attr_t attr;
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
160 pthread_t FCV_threads[maxDis];
filterCV_TD filterCV_TD_Array[maxDis];

printf("Cost Filtering Underway...\n");
filter->preprocess(&lImg, lImg_rgb, mean_lImg, var_lImg);
165 filter->preprocess(&rImg, rImg_rgb, mean_rImg, var_rImg);

//printf("Left Filtering...\n");
//Thread in blocks of threads. Join all threads in each block before continuing.
for(int level = 0; level <= maxDis/threads; level++)
170 {
    int block_size = (level < maxDis/threads) ? threads : (maxDis%threads);

    for(int iter=0; iter < block_size; iter++)

```

```

    {
175         int d = level*threads + iter;
        filterCV_TD_Array[d] = {&lImg, lImg_rgb, mean_lImg, var_lImg, lcostVol[d]};
        pthread_create(&FCV_threads[d], &attr, CVF::filterCV, (void *)&filterCV_TD_Array[d]);
        //printf("Filtering Left CV @ Disparity %d\n", d);
    }
180     for(int iter=0; iter < block_size; iter++)
    {
        int d = level*threads + iter;
        pthread_join(FCV_threads[d], &status);
        //printf("Joining Left CV @ Disparity %d\n", d);
185     }
    }

    //printf("Right Filtering...\n");
    //Thread in blocks of threads. Join all threads in each block before continuing.
190     for(int level = 0; level <= maxDis/threads; level++)
    {
        int block_size = (level < maxDis/threads) ? threads : (maxDis%threads);

        for(int iter=0; iter < block_size; iter++)
195     {
        int d = level*threads + iter;
        filterCV_TD_Array[d] = {&rImg, rImg_rgb, mean_rImg, var_rImg, rcostVol[d]};
        pthread_create(&FCV_threads[d], &attr, CVF::filterCV, (void *)&filterCV_TD_Array[d]);
        //printf("Filtering Right CV @ Disparity %d\n", d);
200     }
        for(int iter=0; iter < block_size; iter++)
        {
            int d = level*threads + iter;
            pthread_join(FCV_threads[d], &status);
205            //printf("Joining Right CV @ Disparity %d\n", d);
        }
    }
    //printf("Filtering Complete\n");
    return;
210 }

void DispEst::DispSelect()
{
    //printf("Disparity Selection Underway...\n");
215    //printf("Left Selection...\n");
    selector->CVSelect_thread(&lImg, lcostVol, maxDis, lDisMap, threads);
    //printf("Right Selection...\n");
    selector->CVSelect_thread(&rImg, rcostVol, maxDis, rDisMap, threads);
    //printf("Selection Complete\n");
220 }

void DispEst::PostProcess()
{
    //printf("Post Processing Underway...\n");
225    postProcessor->processDM(&lImg, &rImg, lDisMap, rDisMap, maxDis, threads);
    //printf("Post Processing Complete\n");
}

```

LISTING C.2: Power measurement and energy recording code for the Xeon Phi platform.

```

1 /*-----
    micEnergy.cpp - Energy, Power and Temperature Monitoring Code for Xeon Phi
    -----
    Author: Charles Leech
5    Email: cl19g10 [at] ecs.soton.ac.uk
    -----*/

#define MICPOWER_MAX_COUNTERS 16
#define MICTEMP_MAX_COUNTERS 8

10 typedef struct MICPOWER_control_state {
    long long counts[MICPOWER_MAX_COUNTERS];    // used for caching

```

```

        long long lastupdate;
    } MICPOWER_control_state_t;

15 struct mic_pwr_info
    {
        long total_pwr[2], pcie_pwr, inst_pwr, inst_pwr_max;
        long con_pwr23, con_pwr24;
        long core_pwr, core_cur, core_vol;
20     long uncore_pwr, uncore_cur, uncore_vol;
        long mem_pwr, mem_cur, mem_vol;
    };

    volatile bool keepAlive = true;
25 double passEnergy = 0.0;
    pthread_t micPthread;

    /* cat /sys/class/micras/power
    110000000 // Total power, win 0 uW
30 109000000 // Total power, win 1 uW
    111000000 // PCI-E connector power
    218000000 // Instantaneous power uW
    200000000 // Max Instantaneous power
    290000000 // 2x3 connector power
35 62000000 // 2x4 connector power
    28000000 0 1043000 // Core rail; Power(uW), Current(uA), Voltage(uV)
    32000000 0 1000000 // Uncore rail; Power(uW), Current(uA), Voltage(uV)
    30000000 0 1501000 // Memory subsystem rail; Power(uW), Current(uA), Voltage(uV)
    */
40 static int read_sysfs_power(mic_pwr_info* mpi)
    {
        FILE* fp = NULL;
        int i;
        int retval = 1;
45 fp = fopen("/sys/class/micras/power", "r");
        if (!fp){
            printf("Could not open Sysfs node: \"/sys/class/micras/power\"\n");
            return 0;
        }
50 retval = fscanf(fp, "%ld %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld",
                    &mpi->total_pwr[0], &mpi->total_pwr[1],
                    &mpi->pcie_pwr,
                    &mpi->inst_pwr, &mpi->inst_pwr_max,
                    &mpi->con_pwr23, &mpi->con_pwr24,
55     &mpi->core_pwr, &mpi->core_cur, &mpi->core_vol,
                    &mpi->uncore_pwr, &mpi->uncore_cur, &mpi->uncore_vol,
                    &mpi->mem_pwr, &mpi->mem_cur, &mpi->mem_vol);
        fclose(fp);
        return retval;
60 }

    /* cat /sys/class/micras/temp
    74 0 // CPU Total temp
    0 0
65 49 0 // Fan-in temp
    60 0 // Fan-out temp
    58 0 // Memory Temp
    54 0 // Core Rail temp
    52 0 // Uncore Rail temp
70 52 0 // Memory Rail Temp
    */
    static int read_sysfs_temp(long long* temp_counts)
    {
        FILE* fp = NULL;
75     int i;
        int retval = 1;
        fp = fopen("/sys/class/micras/temp", "r");
        if (!fp)
            return 0;

```

```

80  for (i=0; i < MICTEMP_MAX_COUNTERS; i++)
    {
        retval &= fscanf(fp, "%lld %lld", &temp_counts[i], temp_counts[i+1]);
    }
    fclose(fp);
85  return retval;
}

double diff_in_ms(struct timespec start, struct timespec end)
{
90  struct timespec temp;
    double milli_seconds;
    if ((end.tv_nsec-start.tv_nsec)<0) {
        temp.tv_sec = end.tv_sec-start.tv_sec-1;
        temp.tv_nsec = 1000000000+end.tv_nsec-start.tv_nsec;
95  } else {
        temp.tv_sec = end.tv_sec-start.tv_sec;
        temp.tv_nsec = end.tv_nsec-start.tv_nsec;
    }
    milli_seconds = (double) ((double) temp.tv_sec * 1000) + ((double) temp.tv_nsec / 1000000
    );
100 return milli_seconds;
}

void* recordEnergy(void *arg) {
    int retval = 0, total_count = 0;
105 mic_pwr_info mic_pwr;
    struct timespec ts, time1, time2;
    ts.tv_sec = 0;
    ts.tv_nsec = 100000000L; // 100 milliseconds
110 float energy = 0.0f;
    while (keepAlive)
    {
        clock_gettime(CLOCK_MONOTONIC, &time1);
        usleep(100000);
115 retval = read_sysfs_power(&mic_pwr);
        clock_gettime(CLOCK_MONOTONIC, &time2);
        energy += (mic_pwr.core_pwr * diff_in_ms(time1,time2));
    }
    keepAlive = true;
120 passEnergy = energy;
    energy = 0.0f;
}

void micpower_start()
125 {
    pthread_create(&micPthread, NULL, recordEnergy, (void*)NULL);
    return;
}

130 float micpower_finalize()
{
    keepAlive = false;
    pthread_join(micPthread, NULL);
    return passEnergy / 1000 / 1000 / 1000;
135 }

```

LISTING C.3: RTM code for modelling and optimisation of the stereo matching application and Xeon Phi platform.

```

1  /*-----
    Author: Charles Leech
    Email: cl19g10 [at] ecs.soton.ac.uk
    -----*/
5  #include "rtm.hpp"

```



```

RTM::RTM(){
    //Set up application controls
    de_ctrl_rtm = new DE_Controls{
10     .cores = NUM_CPU_CORES};
    printf("RTM: App Controls Initialised\n");

    //Initialise application monitors
    de_mon_rtm = new DE_Monitors;
15    de_mon_rtm->perf_req = 1; //should be set by the application
    printf("RTM: App Monitors Initialised\n");

    rtm_props = new RTM_Properties{
        .opt_mode = OPT_MODE_ENG_MIN, .prev_opt_mode = OPT_MODE_PWR_MIN,
20        .pwr_max = 2.2, .prev_pwr_max = 5};
    printf("RTM: RTM Properties Set\n");

    req_cpu_pwr_pred = 4;
    req_cpu_lat_pred = 3;
25    freq_samples = sizeof(cpu_train_freqs)/sizeof(cpu_train_freqs[0]);
    core_samples = sizeof(cpu_train_cores)/sizeof(cpu_train_cores[0]);
    de_phi_model = new cpu_model(req_cpu_pwr_pred, req_cpu_lat_pred, freq_samples*
        core_samples);
    printf("RTM: Model Initialised\n");

30    //Launch system manager - monitors and controls system on behalf of the RTM
    sysm_rtm = new SysM(de_ctrl_rtm, de_mon_rtm, rtm_props);
    //Set up system controls
    sysm_rtm->phi_ctrl = new System_Controls(cpu_freqs[NUM_CPU_FREQS-1], GOV_USERSPACE);
    printf("RTM: System Controls Initialised\n");
35    printf("RTM: RTM Initialised\n");
}

RTM::~~RTM(){
40    printf("RTM: Shutting down System RTM\n");
    delete sysm_rtm;
    printf("RTM: RTM Shut down\n");
}

45 void RTM::train_model(StereoMatch* sm)
{
    printf("RTM: Training...\n");
    float app_perf, app_lat;

50    for(int nf=0; nf < freq_samples; nf++) //Test all Freqs
    {
        int test_freq = cpu_train_freqs[nf]; //Set freq
        //Set the system controls
        sysm_rtm->SetCPUFreq(test_freq);
55        //Test all core and frequency combinations
        for(int nc=0; nc < core_samples; nc++) //Test all # of CPUs
        {
            //Set the app controls
60            int test_cores = cpu_train_cores[nc];
            de_ctrl_rtm->cores = test_cores;

            printf("RTM: Testing DE: %d Cores at %d MHz\n", test_cores, test_freq/1000);

65            micpower_start(); //start logger
            app_lat = sm->Compute()/1000; //run the app under the new controls and record
            app_perf = 1/app_lat;
            float frame_eng = micpower_finalize(); //stop logger
            de_mon_rtm->perf_meas = app_perf;
70            de_phi_model->train(frame_eng/app_lat, app_lat, test_freq, test_cores);
            printf("RTM: Testing DE: Perf = %.4f, Energy = %.4f\n\n", app_perf, frame_eng);
        }
    }
}

```

```

    }
75     return;
    }

    void RTM::opt_de_ctrl(void)
80 {
    // If no changes required
    if(de_mon_rtm->prev_perf_req == de_mon_rtm->perf_req &&
        rtm_props->prev_pwr_max == rtm_props->pwr_max &&
        rtm_props->prev_opt_mode == rtm_props->opt_mode)
85 {
        printf("RTM: No change to constraints, continue running with current settings\n");
    }
    else
    {
90         float est_min_eng = 100, est_min_pwr = 100, est_max_fps=0;
        float est_eng, est_pwr, est_fps;
        int req_cpu_freq = cpu_freqs[NUM_CPU_FREQS-1];
        int req_cpu_cores = NUM_CPU_CORES;

95         //#####
        // Cycle through optimisation modes to determine system and app controls
        //#####
        if(rtm_props->opt_mode == OPT_MODE_ENG_MIN){
            //
            printf("RTM: Running energy optimisation of LR models...\n");
100            de_phi_model->optimise_eng(de_mon_rtm->perf_req, &est_eng, &est_fps, &req_cpu_freq, &
                req_cpu_cores);

            //Set the control parameters based on the results of optimisation in each model
            if(est_fps >= de_mon_rtm->perf_req){
                est_min_eng = est_eng;
105            }
            if(est_min_eng==100){
                printf("RTM: Error: Could not meet performance requirement!\n");
            }
        }
110    else if(rtm_props->opt_mode == OPT_MODE_PWR_MIN){
        //
        printf("RTM: Running power optimisation of LR model...\n");
        de_phi_model->optimise_pwr(de_mon_rtm->perf_req, &est_pwr, &est_fps, &req_cpu_freq, &
            req_cpu_cores);

115        //Set the control parameters based on the results of optimisation in each model
        if(est_fps >= de_mon_rtm->perf_req){
            est_min_pwr = est_pwr;
        }
        if(est_min_pwr==100){
120            printf("RTM: Error: Could not meet performance requirement!\n");
        }
    }

    else if(rtm_props->opt_mode == OPT_MODE_PERF_MAX){
125 //
        printf("RTM: Running performance optimisation of LR model...\n");
        de_phi_model->optimise_fps(rtm_props->pwr_max, &est_pwr, &est_fps, &req_cpu_freq, &
            req_cpu_cores);

        //Set the control parameters based on the results of optimisation from each model
        if(est_pwr <= rtm_props->pwr_max){
130            est_max_fps = est_fps;
        }
        if(est_max_fps==0){
            printf("RTM: Error: Could not meet power constraint!\n");
            float min_pwr=100;
135        }
    }
    else{

```

```

    printf("RTM: Error: Unknown optimisation strategy set, please include this
functionality in the RTM.\n Exiting.\n");
    exit(1);
140 }
    //#####
    //# Set system and app controls into sysm_rtm->odroid_ctrl & de_ctrl_rtm
    //#####
    sysm_rtm->phi_ctrl->cpu_gov = GOV_USERSPACE;
145 sysm_rtm->phi_ctrl->cpu_freq = req_cpu_freq;
    de_ctrl_rtm->cores = req_cpu_cores;
    de_mon_rtm->prev_perf_req = de_mon_rtm->perf_req;
    rtm_props->prev_pwr_max = rtm_props->pwr_max;
    rtm_props->prev_opt_mode = rtm_props->opt_mode;
150 printf("RTM: # Cores set to %d, Frequency set to %d MHz\n", de_ctrl_rtm->cores,
    sysm_rtm->phi_ctrl->cpu_freq);
    sysm_rtm->SetCPUFreq(req_cpu_freq);
    }
    return;
}

```

Appendix D

Stereo Matching Algorithm Data Analysis Scripts

This appendix contains the scripts used to analyse the data collected during experimentation with the stereo matching algorithm. This includes scripts required to create figures shown in Chapter 4.

LISTING D.1: Python script to generate power, performance and energy operating points from experimental data for Figure 4.13 and 4.14.

```
1 import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from matplotlib.lines import Line2D
5 import math

NUM_FREQS = 9
NUM_CORES = 15
NUM_SAMPLES = 5 #length of thread_idx_set
10
freq_series = np.zeros([NUM_FREQS, NUM_CORES], dtype=[('F', int), ('C', int), ('perf', float), ('E', float), ('S', float)])
freq_series_sample = np.zeros([NUM_FREQS, NUM_SAMPLES], dtype=[('F', int), ('C', int), ('perf', float), ('E', float), ('S', float)])

filename="de_phi_profile_upto60_v2.csv"
15 fx_ex = np.genfromtxt(filename, delimiter=' ', names=True) #Read in csv data from the file
freq_set=[619, 667, 714, 762, 857, 952, 1048, 1143, 1238]
thread_idx_set = [0,1,3,7,14] #to pick out profile entries at 4,8,16,32,60
markers = Line2D.filled_markers

20 # Create a figure, specific size & dots per inch
fig1 = plt.figure(figsize=(8, 8), dpi=128)
scatter_pp = fig1.add_subplot(111)
scatter_pp.set_xlabel('Average Power per Frame (W)', fontsize=16)
scatter_pp.set_ylabel('Average Performance (FPS)', fontsize=16)
25 scatter_pp.tick_params(axis='y', labelsize=16)
scatter_pp.tick_params(axis='x', labelsize=16)
scatter_pp.grid(True)

max_threads = 60
30 #Data table creation
j=0
for freq_idx in range(0, NUM_FREQS, 1):
    i=0
```

```

    for thread_idx in range (0,NUM_CORES,1):
35
        p1 = fx_ex[j]['perf']
        e1 = fx_ex[j]['energy']
        j+=1
        p2 = fx_ex[j]['perf']
40
        e2 = fx_ex[j]['energy']
        j+=1
        p3 = fx_ex[j]['perf']
        e3 = fx_ex[j]['energy']
        j+=1
45
        p4 = fx_ex[j]['perf']
        e4 = fx_ex[j]['energy']
        j+=1

        perf_av = (p1 + p2 + p3 + p4) / 4
50
        e_av     = (e1 + e2 + e3 + e4) / 4

        freq_series[NUM_FREQS-1-freq_idx][NUM_CORES-1-i] = (freq_set[NUM_FREQS-1-freq_idx],
max_threads-(4*thread_idx), perf_av, e_av, 0.0)
        i+=1

55 for freq_idx in range (0,NUM_FREQS):
    freq_series_sample[freq_idx] = freq_series[freq_idx][thread_idx_set]

    #Plot Graph
    #scatter_pp
60 for f in range(0,NUM_FREQS):
    scatter_pp.plot(freq_series[f]['E']*freq_series[f]['perf'],freq_series[f]['perf'],
        label=str(freq_set[f]), marker=markers[f], linestyle='None',markersize=10, color=plt.cm
        .jet(f/10.))

    #Drawing optimisation arrow on figure
    #Start point
65 f1=8
    k1=1
    px1=freq_series[f1]['E'][k1]*freq_series[f1]['perf'][k1]
    py1=freq_series[f1]['perf'][k1]
    p1text = "    P1:\n" + str(freq_series[f1]['C'][k1]) + " cores \n" + str(freq_series[f1]['F
    '][k1]) + " MHz"

70 scatter_pp.plot(px1,py1, marker="o", markersize=28, mec='k', mfc="None", mew=3)
    scatter_pp.annotate(p1text, xy=(px1,py1), xytext=(25,-20), textcoords='offset points',
        fontsize=16)

    #End point
    f2=0
75 k2=5
    px2=freq_series[f2]['E'][k2]*freq_series[f2]['perf'][k2]
    py2=freq_series[f2]['perf'][k2]
    p2text = "    P2:\n" + str(freq_series[f2]['C'][k2]) + " cores \n" + str(freq_series[f2]['F
    '][k2]) + " MHz"

    scatter_pp.plot(px2,py2, marker="o", markersize=28, mec='k', mfc="None", mew=3)
80 scatter_pp.annotate(p2text, xy=(px2,py2), xytext=(-60,20), textcoords='offset points',
        fontsize=16)

    #Arrow
    scatter_pp.annotate("",xy=(px2,py2), xytext=(px1, py1), textcoords='data', arrowprops=dict(
        color='r', width=12, headlength=20, headwidth=25))

    #Red text
    scatter_pp.text(12, 0.95, "Same Performance\nLower Power", color='r', ha="left", va="top",
        size=22, bbox=dict(fc="None",ec="k",lw=1))

85
    legend = scatter_pp.legend(loc= 'lower right', title='Freq (MHz)', numpoints=1, fontsize
        =16)
    plt.setp(legend.get_title(),fontsize=16)
    fig1.savefig("pp_power_opt.pdf",bbox_inches='tight')

90 #####
    # Create a figure, specific size & dots per inch

```

```

fig1 = plt.figure(figsize=(8, 8), dpi=128)
scatter_pp = fig1.add_subplot(111)
scatter_pp.set_xlabel('Average Power per Frame (W)', fontsize=16)
95 scatter_pp.set_ylabel('Average Performance (FPS)', fontsize=16)
scatter_pp.tick_params(axis='y', labels=16)
scatter_pp.tick_params(axis='x', labels=16)
scatter_pp.grid(True)

100 #Plot Graph
#scatter_pp
for f in range(0, NUM_FREQS):
    scatter_pp.plot(freq_series[f]['E']*freq_series[f]['perf'], freq_series[f]['perf'],
                    label=str(freq_set[f]), marker=markers[f], linestyle='None', markersize=10, color=plt.cm
                    .jet(f/10.))

105 #Drawing optimisation arrow on figure
#Start point
f1=8
k1=0
px1=freq_series[f1]['E'][k1]*freq_series[f1]['perf'][k1]
110 py1=freq_series[f1]['perf'][k1]
p1text = "      P3:\n" + str(freq_series[f1]['C'][k1]) + " cores \n" + str(freq_series[f1]['F
    '][k1]) + " MHz"
scatter_pp.plot(px1, py1, marker="o", markersize=28, mec='k', mfc="None", mew=3)
scatter_pp.annotate(p1text, xy=(px1, py1), xytext=(25, -20), textcoords='offset points',
                    fontsize=16)

115 #End point
f2=1
k2=7
px2=freq_series[f2]['E'][k2]*freq_series[f2]['perf'][k2]
py2=freq_series[f2]['perf'][k2]
120 p2text = "      P4:\n" + str(freq_series[f2]['C'][k2]) + " cores \n" + str(freq_series[f2]['F
    '][k2]) + " MHz"
scatter_pp.plot(px2, py2, marker="o", markersize=28, mec='k', mfc="None", mew=3)
scatter_pp.annotate(p2text, xy=(px2, py2), xytext=(-80, 20), textcoords='offset points',
                    fontsize=16)

#Arrow
scatter_pp.annotate("", xy=(px2, py2), xytext=(px1, py1), textcoords='data', arrowprops=dict(
    color='r', width=12, headlength=20, headwidth=25))
125 #Red text
scatter_pp.text(12, 0.95, "Same Power\nHigher Performance", color='r', ha="left", va="top",
               size=22, bbox=dict(fc="None", ec="k", lw=1))

legend = scatter_pp.legend(loc= 'lower right', title='Freq (MHz)', numpoints=1, fontsize
=16)
plt.setp(legend.get_title(), fontsize=16)
130 fig1.savefig("pp_perf_opt.pdf", bbox_inches='tight')

#####
# Create a figure, specific size & dots per inch
fig1 = plt.figure(figsize=(8, 8), dpi=128)
135 scatter_pp = fig1.add_subplot(111)
scatter_pp.set_xlabel('Average Power per Frame (W)', fontsize=16)
scatter_pp.set_ylabel('Average Performance (FPS)', fontsize=16)
scatter_pp.tick_params(axis='y', labels=16)
scatter_pp.tick_params(axis='x', labels=16)
140 scatter_pp.grid(True)

#Plot Graph
#scatter_pp
for f in range(0, NUM_FREQS):
145 scatter_pp.plot(freq_series[f]['E']*freq_series[f]['perf'], freq_series[f]['perf'],
                    label=str(freq_set[f]), marker=markers[f], linestyle='None', markersize=10, color=plt.
                    cm.jet(f/10.))

#Drawing optimisation arrow on figure
#Start point

```

```

f1=5
150 k1=7
    px1=freq_series[f1]['E'][k1]*freq_series[f1]['perf'][k1]
    py1=freq_series[f1]['perf'][k1]
    p1text = "      P5:\n" + str(freq_series[f1]['C'][k1]) + " cores \n" + str(freq_series[f1]['F
        '][k1]) + " MHz"
    scatter_pp.plot(px1,py1, marker="o", markersize=28, mec='k', mfc="None", mew=3)
155 scatter_pp.annotate(p1text, xy=(px1,py1), xytext=(-50,30), textcoords='offset points',
        fontsize=16)

#End point
f2=1
k2=3
160 px2=freq_series[f2]['E'][k2]*freq_series[f2]['perf'][k2]
    py2=freq_series[f2]['perf'][k2]
    p2text = "      P6:\n" + str(freq_series[f2]['C'][k2]) + " cores \n" + str(freq_series[f2]['F
        '][k2]) + " MHz"
    scatter_pp.plot(px2,py2, marker="o", markersize=28, mec='k', mfc="None", mew=3)
    scatter_pp.annotate(p2text, xy=(px2,py2), xytext=(-60,40), textcoords='offset points',
        fontsize=16)

165 #Arrow
    scatter_pp.annotate("",xy=(px2,py2), xytext=(px1, py1), textcoords='data', arrowprops=dict(
        color='r', width=12, headlength=20, headwidth=25))
    #Red text
    scatter_pp.text(12, 0.95, "Power and \nPerformance \nScaling", color='r', ha="left", va="
        top", size=22, bbox=dict(fc="None",ec="k",lw=1))

170 legend = scatter_pp.legend(loc= 'lower right', title='Freq (MHz)', numpoints=1, fontsize
    =16)
    plt.setp(legend.get_title(),fontsize=16)
    fig1.savefig("pp_scaling.pdf",bbox_inches='tight')

```

LISTING D.2: Python script to plot experimental data for the RTM optimisation in response to changing performance targets in Figure 4.18.

```

1 import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import math

5
#Read in csv data from the file
rtm_data = np.genfromtxt("XeonPhiRTMData_60cores.txt", delimiter=',', names=True)
app_data = np.genfromtxt("XeonPhiAppData_60cores.txt", delimiter=',', names=True)
rtm_data_ond = np.genfromtxt("XeonPhiRTMData_OND.txt", delimiter=',', names=True)
10 app_data_ond = np.genfromtxt("XeonPhiAppData_OND.txt", delimiter=',', names=True)

# Create a figure, specific size & dots per inch
fig1 = plt.figure(figsize=(8, 12), dpi=128)

15 ### Subplot 1
plot_perf = fig1.add_subplot(311)
plot_perf.set_title('Tracking of measured to target performance', fontsize=12)
#plot_perf.set_xlabel('Time (s)', fontsize=12)
plot_perf.xaxis.set_ticks(np.arange(0,210,10))
20 plot_perf.set_ylabel('Performance (FPS)', fontsize=12)
plot_perf.grid(True)
#Plot Graph
plot_perf.plot(app_data['Time']/1000000, app_data['PerfMeas'], 'b-', label='RTM Measured')
plot_perf.plot(app_data_ond['Time']/1000000, app_data_ond['PerfMeas'], 'g--', label='Linux
    Measured')
25 plot_perf.plot(app_data_ond['Time']/1000000, app_data_ond['PerfReq'], 'r-', label='Target')

plot_perf.legend(bbox_to_anchor=(1.02, 1), loc=2,
    borderaxespad=0., title='Performance Type')

30 abs_err = sum(abs(app_data['PerfMeas'] - app_data['PerfReq']))/app_data['PerfReq']/len(
    app_data) * 100

```



```

list1 = app_data['PerfReq']
list2 = app_data['PerfMeas']
neg_abs_err = sum([max(list1 - list2, 0) for list1, list2 in zip(list1, list2)]/app_data['
    PerfReq']/len(app_data) * 100
35
#### Subplot 2
plot_cont = fig1.add_subplot(312, sharex=plot_perf)
plot_cont.set_title('Adaptation of System Controls', fontsize=12)
plot_cont.set_xlabel('Time (s)', fontsize=12)
#plot_cont.set_xlabel('Time (s)', fontsize=12)
40 #plot_cont.xaxis.set_ticks(np.arange(0,22,1))
plot_cont.set_ylabel('Frequency (MHz)', fontsize=12, color='blue')
plot_cont.tick_params(axis='y', colors='blue')
plot_cont.grid(True)
#Plot Graph
45 plot_cont.plot(rtm_data['Time']/1000000, rtm_data['Freq']/1000, 'b-', label='RTM')
plot_cont.scatter(rtm_data_ond['Time']/1000000, rtm_data_ond['Freq']/1000, s=8, c='g',
    edgecolors='none', label='Linux')
ax2 = plot_cont.twinx()
ax2.set_ylabel('Number of Cores Enabled', fontsize=12, color='red')
50 ax2.plot(rtm_data['Time']/1000000, rtm_data['Cores'], 'r-', label='RTM')

leg2 = plot_cont.legend(bbox_to_anchor=(1.08, 1), loc=2,
    borderaxespad=0., title='Frequency\nController')
leg2.legendHandles[0].set_color('black')
55
#### Subplot 3
plot_mon = fig1.add_subplot(313, sharex=plot_perf)
plot_mon.set_title('Energy and Average Power Consumed per Frame', fontsize=12)
plot_mon.set_xlabel('Time (s)', fontsize=12)
60 #plot_mon.xaxis.set_ticks(np.arange(0,22,1))
plot_mon.set_ylabel('Energy (J)', fontsize=12, color='blue')
plot_mon.grid(True)
#Plot Graph
plot_mon.plot(app_data['Time']/1000000, app_data['Energy'], 'b-', label='RTM')
65 plot_mon.plot(app_data_ond['Time']/1000000, app_data_ond['Energy'], 'b--', label='Linux')
ax2 = plot_mon.twinx()
ax2.set_ylabel('Average Power (W)', fontsize=12, color='red')
ax2.plot(app_data['Time']/1000000, app_data['Power'], 'r-')
ax2.plot(app_data_ond['Time']/1000000, app_data_ond['Power'], 'r--')
70
leg3 = plot_mon.legend(bbox_to_anchor=(1.08, 1), loc=2,
    borderaxespad=0., title='Frequency\nController')
leg3.legendHandles[0].set_color('black')
leg3.legendHandles[1].set_color('black')
75
app_data_ond = np.genfromtxt("XeonPhiAppData_OND_edit.txt", delimiter=',', names=True)
avg_pwr_sav = sum((app_data_ond['Power'] - app_data['Power'])/app_data_ond['Power'])/len(
    app_data) * 100
energy_sav = sum((app_data_ond['Energy'] - app_data['Energy'])/app_data_ond['Energy'])/len(
    app_data) * 100

80 plt.tight_layout()
#Print Graphs
fig1.savefig("time_series_rtm_vs_ond.pdf", bbox_inches='tight')
plt.show()

```


Appendix E

Framework Code

This appendix contains sample code for the PRiME Framework. This was used to conduct the experiments described in Chapter 5. Additional code for the framework, including example classes and API interfaces, can be found at: https://github.com/PRiME-project/PRiME_Framework.

LISTING E.1: API header for common types in the PRiME Framework.

```
1 #ifndef PRIME_API_T_H
   #define PRIME_API_T_H
   #include <limits.h>
   #include <math.h>
5
   namespace prime { namespace api
   {
       // Numeric types
       typedef signed int disc_t;
10      typedef float cont_t;

       // Min and max limits
       static const disc_t PRIME_DISC_MIN = INT_MIN;
       static const disc_t PRIME_DISC_MAX = INT_MAX;
15      static const cont_t PRIME_CONT_MIN = -INFINITY;
       static const cont_t PRIME_CONT_MAX = INFINITY;
   } }
   #endif
```

LISTING E.2: Common application types API header.

```
1 #ifndef PRIME_API_APP_T_H
   #define PRIME_API_APP_T_H

   #include "prime_api_t.h"
5
   namespace prime { namespace api { namespace app
   {
       // These are application-level type definitions

10      // Available types of application-level knob
       enum knob_type_t {PRIME_PAR, // Parallelism
                        PRIME_PREC, // Precision
                        PRIME_DEV_SEL, // Device selection
                        PRIME_ITR // Iterations
15      };

       // Available types of application-level monitor
       enum mon_type_t {PRIME_PERF, // Performance
```

```

    PRIME_ACC, // Accuracy
20    PRIME_ERR, // Error
    PRIME_POW // for KOCL...
};

// Application-level discrete knob container
25 typedef struct knob_disc_t {
    pid_t proc_id;
    unsigned int id;
    knob_type_t type;
    prime::api::disc_t min;
30    prime::api::disc_t max;
    prime::api::disc_t val;
} knob_disc_t;

// Application-level continuous knob container
35 typedef struct knob_cont_t {
    pid_t proc_id;
    unsigned int id;
    knob_type_t type;
    prime::api::cont_t min;
40    prime::api::cont_t max;
    prime::api::cont_t val;
} knob_cont_t;

// Application-level discrete monitor container
45 typedef struct mon_disc_t {
    pid_t proc_id;
    unsigned int id;
    mon_type_t type;
    prime::api::disc_t min;
50    prime::api::disc_t max;
    prime::api::disc_t val;
    prime::api::cont_t weight;
} mon_disc_t;

55 // Application-level continuous monitor container
typedef struct mon_cont_t {
    pid_t proc_id;
    unsigned int id;
    mon_type_t type;
60    prime::api::cont_t min;
    prime::api::cont_t max;
    prime::api::cont_t val;
    prime::api::cont_t weight;
} mon_cont_t;
65 } } }

#endif

```

LISTING E.3: Common device types API header.

```

1 #ifndef PRIME_API_DEV_T_H
#define PRIME_API_DEV_T_H

#include "prime_api_t.h"
5
namespace prime { namespace api { namespace dev
{
    // These are device-level type definitions

10 // Available types of device-level knob
enum knob_type_t {PRIME_VOLT,
    PRIME_FREQ,
    PRIME_EN,
    PRIME_PMC_CNT,
15    PRIME_GOVERNOR,
    PRIME_FREQ_EN

```

```

        };

    // Available types of device-level monitor
20    enum mon_type_t {PRIME_POW,
        PRIME_TEMP,
        PRIME_CYCLES,
        PRIME_PMC
    };

25    // Device-level discrete knob container
    typedef struct knob_disc_t {
        unsigned int id;
        knob_type_t type;
30    prime::api::disc_t min;
        prime::api::disc_t max;
        prime::api::disc_t val;
        prime::api::disc_t init;
    } knob_disc_t;

35    // Device-level continuous knob container
    typedef struct knob_cont_t {
        unsigned int id;
        knob_type_t type;
40    prime::api::cont_t min;
        prime::api::cont_t max;
        prime::api::cont_t val;
        prime::api::cont_t init;
    } knob_cont_t;

45    // Device-level discrete monitor container
    typedef struct mon_disc_t {
        unsigned int id;
        mon_type_t type;
50    prime::api::disc_t val;
    } mon_disc_t;

    // Device-level continuous monitor container
    typedef struct mon_cont_t {
        unsigned int id;
        mon_type_t type;
        prime::api::cont_t val;
    } mon_cont_t;
    } } }

60 #endif

```

LISTING E.4: Application API header, containing the `rtm_interface`.

```

1 #ifndef PRIME_API_APP_H
    #define PRIME_API_APP_H
    #include <string>
    #include <vector>
5    #include <mutex>
    #include <condition_variable>
    #include "prime_api_t.h"
    #include "prime_api_app_t.h"
    #include "uds.h"

10    namespace prime { namespace api { namespace app
    {
        class rtm_interface
        {
15    public:
            rtm_interface(pid_t proc_id);
            ~rtm_interface();

            void message_handler(std::vector<char> message);

20

```

```

void reg(pid_t proc_id, unsigned long int ur_id);
void dereg(pid_t proc_id);

knob_disc_t knob_disc_reg(knob_type_t type, prime::api::disc_t min, prime::api::disc_t
max, prime::api::disc_t val);
25 knob_cont_t knob_cont_reg(knob_type_t type, prime::api::cont_t min, prime::api::cont_t
max, prime::api::cont_t val);

void knob_disc_min(knob_disc_t knob, prime::api::disc_t min);
void knob_disc_max(knob_disc_t knob, prime::api::disc_t max);
void knob_cont_min(knob_cont_t knob, prime::api::cont_t min);
30 void knob_cont_max(knob_cont_t knob, prime::api::cont_t max);

prime::api::disc_t knob_disc_get(knob_disc_t knob);
prime::api::cont_t knob_cont_get(knob_cont_t knob);

35 void knob_disc_dereg(knob_disc_t knob);
void knob_cont_dereg(knob_cont_t knob);

mon_disc_t mon_disc_reg(mon_type_t type, prime::api::disc_t min, prime::api::disc_t max
, prime::api::cont_t weight);
mon_cont_t mon_cont_reg(mon_type_t type, prime::api::cont_t min, prime::api::cont_t max
, prime::api::cont_t weight);
40 void mon_disc_min(mon_disc_t mon, prime::api::disc_t min);
void mon_disc_max(mon_disc_t mon, prime::api::disc_t max);
void mon_disc_weight(mon_disc_t mon, prime::api::cont_t weight);
void mon_cont_min(mon_cont_t mon, prime::api::cont_t min);
void mon_cont_max(mon_cont_t mon, prime::api::cont_t max);
45 void mon_cont_weight(mon_cont_t mon, prime::api::cont_t weight);

void mon_disc_set(mon_disc_t mon, prime::api::disc_t val);
void mon_cont_set(mon_cont_t mon, prime::api::cont_t val);

50 void mon_disc_dereg(mon_disc_t mon);
void mon_cont_dereg(mon_cont_t mon);

private:
pid_t proc_id;
55 prime::uds socket, ui_socket, logger_socket;

std::mutex knobs_disc_m, knobs_cont_m, mons_disc_m, mons_cont_m;
std::vector<knob_disc_t> knobs_disc;
std::vector<knob_cont_t> knobs_cont;
60 std::vector<mon_disc_t> mons_disc;
std::vector<mon_cont_t> mons_cont;

std::mutex disc_return_m, cont_return_m;
prime::api::disc_t disc_return;
65 prime::api::cont_t cont_return;

std::mutex knob_disc_return_m, knob_cont_return_m, mon_disc_return_m, mon_cont_return_m
;
knob_disc_t knob_disc_return;
knob_cont_t knob_cont_return;
70 mon_disc_t mon_disc_return;
mon_cont_t mon_cont_return;

std::mutex app_reg_m, app_dereg_m;
std::condition_variable app_reg_cv;
75 std::condition_variable app_dereg_cv;

std::mutex knob_disc_reg_m, knob_cont_reg_m, knob_disc_get_m, knob_cont_get_m;
std::condition_variable knob_disc_reg_cv, knob_cont_reg_cv, knob_disc_get_cv,
knob_cont_get_cv;
bool knob_disc_get_flag, knob_cont_get_flag;
80 std::mutex mon_disc_reg_m, mon_cont_reg_m, mon_disc_get_m, mon_cont_get_m;
std::condition_variable mon_disc_reg_cv, mon_cont_reg_cv, mon_disc_get_cv,
mon_cont_get_cv;

```

```

};

class ui_interface
85 {
public:
    ui_interface(pid_t proc_id,
        boost::function<void(pid_t)> app_reg_handler,
        boost::function<void(pid_t)> app_dereg_handler,
90    boost::function<void(unsigned int, prime::api::disc_t)> mon_disc_min_handler,
        boost::function<void(unsigned int, prime::api::disc_t)> mon_disc_max_handler,
        boost::function<void(unsigned int, prime::api::disc_t)> mon_disc_weight_handler,
        boost::function<void(unsigned int, prime::api::cont_t)> mon_cont_min_handler,
        boost::function<void(unsigned int, prime::api::cont_t)> mon_cont_max_handler,
95    boost::function<void(unsigned int, prime::api::cont_t)> mon_cont_weight_handler,
        boost::function<void(pid_t)> app_stop_handler
    );
    ~ui_interface();

100    void message_handler(std::vector<char> message);

    void return_ui_app_start(void);
    void return_ui_app_stop(void);
    void ui_app_error(std::string message);

105 private:
    pid_t proc_id;

    boost::function<void(pid_t)> app_reg_handler, app_dereg_handler;
110    boost::function<void(unsigned int, prime::api::disc_t)> mon_disc_min_handler;
    boost::function<void(unsigned int, prime::api::disc_t)> mon_disc_max_handler;
    boost::function<void(unsigned int, prime::api::disc_t)> mon_disc_weight_handler;
    boost::function<void(unsigned int, prime::api::cont_t)> mon_cont_min_handler;
    boost::function<void(unsigned int, prime::api::cont_t)> mon_cont_max_handler;
115    boost::function<void(unsigned int, prime::api::cont_t)> mon_cont_weight_handler;
    boost::function<void(pid_t)> app_stop_handler;

    prime::uds socket, logger_socket;
};
120 } } }
#endif

```

Appendix F

Framework Data Analysis Scripts

This appendix contains data analysis scripts used to process data from the framework. This was used to analyse data in the experiments described in Chapter 5.

LISTING F.1: Time series plotting and processing script for standard logger output. Used to plot Figure 5.11.

```
1  #!/usr/bin/python3
   #####
   # analysis_time_series.py
   # Created by Charles Leech on 2017-07-12.
5  #####
   import numpy as np
   import matplotlib
   matplotlib.use("pdf")
   import matplotlib.pyplot as plt
10 import time
   import sys
   from csv import reader

   style_name = "seaborn-paper"
15 if style_name in plt.style.available:
   plt.style.use(style_name)
   print("Using %s style" % style_name)

   MILLION = 1000000
20 #####
   ### Defined Functions
   #####
   def new_subplot(figure):
       n = len(fig.axes)
25   for i in range(n):
       fig.axes[i].change_geometry(n+1, 1, i+1)
       return fig.add_subplot(n+1, 1, n+1)

   #####
30 ### Read in messages from logger file
   #####
   try:
       log_file = str(sys.argv[1])
   except:
35   print("Specify a filename as the first argument")
       exit(0)

   try:
       with open(log_file, "r") as f:
40   messages = list(reader(f))
```

```

except OSError:
    print("Invalid filename specified")
    exit(0)

45 try:
    num_lines = int(sys.argv[2])
    print("Parsing first %s lines of %s" %(num_lines, log_file))
except:
    num_lines = len(messages)
50 print("Parsing all lines of %s" %(log_file))

#####
### Set up data tables and timescales
#####
55 table_api_types = dict()
ts_only_types = ["PRIME_API_DEV_RETURN_KNOB_DISC_REG", "PRIME_API_DEV_RETURN_KNOB_CONT_REG",
    \
        "PRIME_API_DEV_RETURN_MON_DISC_REG", "PRIME_API_DEV_RETURN_MON_CONT_REG", \
        "PRIME_API_DEV_RETURN_ARCH_GET"]

60 start_time = float()
end_time = float()

for field in messages[0]:
    name, val = field.split(":")
65 if name in ["ts"]:
    start_time = float(val)
    break

for field in messages[num_lines-1]:
70 name, val = field.split(":")
    if name in ["ts"]:
        end_time = float(val)
        break

75 duration = (end_time - start_time)/MILLION

#####
### Parse message one by one
#####
80 for msg in messages[:num_lines]:
    name, api_type = msg.pop(0).split(":")

    msg_end = msg.pop(-1)
    msg_names = []
85 msg_vals = []

    for field in msg:
        name, val = field.split(":")

90 if api_type not in ts_only_types:
    if name in ["ts"]:
        msg_names.append((str(name), np.dtype("f8")))
        msg_vals.append((float(val) - start_time)/MILLION)
    elif name in ["id", "proc_id"]:
95 msg_names.append((str(name), np.dtype("i8")))
        msg_vals.append(val)
    elif name in ["type", "name"]:
        msg_names.append((str(name), np.dtype("U25")))
        msg_vals.append(val)
100 else:
        msg_names.append((str(name), np.dtype("f8")))
        msg_vals.append(val)
    else:
        if name == "ts":
105 msg_names = [(str(name), np.dtype("f8"))]
            msg_vals = [(float(val) - start_time)/MILLION]
            break

```

```

#####
110  ### Split messages by api type
#####
    if api_type not in table_api_types:
        table_api_types[api_type] = np.array(tuple(msg_vals), dtype=np.dtype(msg_names))
    else:
115     table_api_types[api_type] = np.append(table_api_types[api_type], np.array(tuple(
        msg_vals), dtype=np.dtype(msg_names)))

#####
    ### Start Plotting figures
#####
120  NUM_SUBPLOTS = 6
    #fig, subplt = plt.subplots(3, sharex=True, figsize=(10, 12))
    #fig, subplt = plt.subplots(NUM_SUBPLOTS, figsize=(10, NUM_SUBPLOTS*2.5))
    plot_num = 0
125  fig = plt.figure(figsize=(6, 12))
    ax = fig.add_subplot(111)

#####
    ### Plot Device Temperature monitors
#####
130  try:
        dev_mon_cont_updates = table_api_types["PRIME_API_DEV_RETURN_MON_CONT_GET"]
    except:
        print("Warning: No device monitor values read by RTM (PRIME_API_DEV_RETURN_MON_CONT_GET).
            Some device trade-offs may not be plotted.")
135  else:
        dev_mon_temp_updates = dev_mon_cont_updates[dev_mon_cont_updates["type"] == "PRIME_TEMP"]
        if(len(dev_mon_temp_updates)):
            dev_mon_temp = dict()
            if plot_num:
140             ax = new_subplot(fig)
            for idx in np.unique(dev_mon_temp_updates["id"]):
                dev_mon_temp[idx] = dev_mon_temp_updates[dev_mon_temp_updates["id"] == idx]

                #remove outliers - replace with previous value
145             if(len(dev_mon_temp[idx][dev_mon_temp[idx]['val'] < 20])):
                 print("Removed Temp Outlier(s)")
                 np.copyto(dev_mon_temp[idx], np.roll(dev_mon_temp[idx], -1), where=dev_mon_temp[idx]
                    ['val'] < 20)

                ax.plot(dev_mon_temp[idx]["ts"], dev_mon_temp[idx]["val"], label=idx)
150             ax.set_xlim(0, duration)

            ax.legend(bbox_to_anchor=(1.02, 1), loc=2, borderaxespad=0., title="Monitor IDs")
            ax.set_ylabel("Temperature ($^\circ$C)")
            print("Plotted temperature monitors")
155             plot_num+=1

        dev_mon_pow_updates = dev_mon_cont_updates[dev_mon_cont_updates["type"] == "PRIME_POW"]
        if(len(dev_mon_pow_updates)):
            dev_mon_pow = dict()
            if plot_num:
160             ax = new_subplot(fig)
            for idx in np.unique(dev_mon_pow_updates["id"]):
                dev_mon_pow[idx] = dev_mon_pow_updates[dev_mon_pow_updates["id"] == idx]

                #remove outliers - replace with previous value
165             if(len(dev_mon_pow[idx][dev_mon_pow[idx]['val'] > 20])):
                 print("Removed Power Outlier(s)")
                 np.copyto(dev_mon_pow[idx], np.roll(dev_mon_pow[idx], -1), where=dev_mon_pow[idx]['
                    val'] < 20)

            ax.plot(dev_mon_pow[idx]["ts"], dev_mon_pow[idx]["val"], label=idx)
170             ax.set_xlim(0, duration)

```

```

    ax.legend(bbox_to_anchor=(1.02, 1), loc=2, borderaxespad=0.,title="Monitor IDs")
    ax.set_ylabel("Power (W)")
175     print("Plotted power monitors")
        plot_num+=1

#####
### Plot Device Power monitors
180 #####
    else:
        try:
            dev_log_mon_cont = table_api_types["PRIME_LOG_DEV_MON_CONT"]
        except:
185             print("Warning: No device monitor values logged directly from the device (
                PRIME_LOG_DEV_MON_CONT). Some device trade-offs may not be plotted.")
        else:
            dev_log_mon_pow = dev_log_mon_cont[dev_log_mon_cont["type"] == "PRIME_POW"]
            dev_mon_pow = dict()

190             if plot_num:
                ax = new_subplot(fig)
                for idx in np.unique(dev_log_mon_pow["id"]):
                    dev_mon_pow[idx] = dev_log_mon_pow[dev_log_mon_pow["id"] == idx]

195             #remove outliers - replace with previous value
            if(len(dev_mon_pow[idx][dev_mon_pow[idx]['val'] > 20])):
                print("Removed Power Outlier(s)")
                np.copyto(dev_mon_pow[idx], np.roll(dev_mon_pow[idx], -1), where=dev_mon_pow[idx
                ]['val'] > 20)

200             ax.plot(dev_mon_pow[idx]["ts"], dev_mon_pow[idx]["val"], label=idx)
            ax.set_xlim(0, duration)

            ax.legend(bbox_to_anchor=(1.02, 1), loc=2, borderaxespad=0.,title="Monitor IDs")
            ax.set_ylabel("Power (W)")
205             print("Plotted power monitors")
                plot_num+=1

#####
### Plot Device Frequency Knobs
210 #####
    try:
        dev_knob_disc_updates = table_api_types["PRIME_API_DEV_KNOB_DISC_SET"]
    except KeyError:
        print("No Device Knobs Set")
215     else:
        try:
            dev_knob_freq_updates = dev_knob_disc_updates[dev_knob_disc_updates["type"] == "
                PRIME_FREQ"]
            dev_knob_freq = dict()

220             if plot_num:
                ax = new_subplot(fig)
                for idx in np.unique(dev_knob_freq_updates["id"]):
                    dev_knob_freq[idx] = dev_knob_freq_updates[dev_knob_freq_updates["id"] == idx]

225             ax.step(dev_knob_freq[idx]["ts"], (dev_knob_freq[idx]["val"]+2)*100, where="post",
                label=idx)
            ax.set_xlim(0, duration)

            ax.legend(bbox_to_anchor=(1.02, 1), loc=2, borderaxespad=0.,title="Knob IDs")
            ax.set_ylabel("Frequency (MHz)")
230             print("Plotted frequency knobs")
                plot_num+=1
        except KeyError:
            print("No Frequency Knobs Set")

235 #####

```

```

#### Plot App Monitors
#####
app_mon_cont_updates = table_api_types["PRIME_API_APP_MON_CONT_SET"]

240 app_mon_cont = dict()
for idx in np.unique(app_mon_cont_updates["id"]):
    app_mon_cont[idx] = app_mon_cont_updates[app_mon_cont_updates["id"] == idx]

    if plot_num:
245         ax = new_subplot(fig)
         ax.plot(app_mon_cont[idx]["ts"], app_mon_cont[idx]["val"])
         ax.set_xlim(0, duration)
         if app_mon_cont[idx][0]["type"] == "PRIME_ERR":
             ax.set_yscale('log')
250             ax.set_ylabel("Error Monitor " + str(idx))
             for label in ax.yaxis.get_ticklabels()[::2]:
                 label.set_visible(False)
             app_mon_err = app_mon_cont[idx]
         elif app_mon_cont[idx][0]["type"] == "PRIME_PERF":
255             ax.set_ylabel("Performance Monitor " + str(idx))
             app_mon_perf = app_mon_cont[idx]
         print("Plotted app monitors")

#####
260 #### Plot App Knobs
#####
try:
    #val is contained in return message from KNOB_DISC_GET
    app_knob_disc_updates = table_api_types["PRIME_API_APP_RETURN_KNOB_DISC_GET"]
265    #app_knob_disc_updates = app_knob_disc_updates[app_knob_disc_updates["type"] != "
        PRIME_DEV_SEL"]

    app_knob_disc = dict()
    for idx in np.unique(app_knob_disc_updates["id"]):
        app_knob_disc[idx] = app_knob_disc_updates[app_knob_disc_updates["id"] == idx]

270
        ax = new_subplot(fig)
        ax.plot(app_knob_disc[idx]["ts"], app_knob_disc[idx]["val"])
        ax.set_xlim(0, duration)
        ax.set_ylabel("App Knob " + str(idx))
275        ax.margins(0, 0.02)
        print("Plotted app knobs")
    except KeyError:
        pass

280 ax.set_xlabel("Time (s)")
figure_filename = log_file[:-4] + "_time_series" + ".pdf"
fig.savefig(figure_filename, bbox_inches="tight")

```

LISTING F.2: Python class `mon_ops` used to store and calculate monitor operating points.

```

1 #!/usr/bin/python3
#####
# ui.py
# Created by Charles Leech on 2017-09-02.
5 #####
import numpy as np

class mon_ops:
    def __init__(self, mon, knobs):
10
        self.mon = mon
        #self.knobs = knobs
        self.op_knobs = dict()

15
        #Init arrays using a line from each of the input knob arrays
        for idx in knobs:
            self.op_knobs[idx] = knobs[idx][0]

```

```

        self.op_knobs[idx] = np.delete(self.op_knobs[idx], 0)

20     for entry in mon:
        for idx in knobs:
            try:
                self.op_knobs[idx] = np.append(self.op_knobs[idx], knobs[idx][knobs[idx]["ts"] <
entry["ts"]][-1])
            except IndexError:
25                self.op_knobs[idx] = np.append(self.op_knobs[idx], knobs[idx][-1])

        #Remove the first lines (added to create arrays at the start)
        for idx in knobs:
30            self.op_knobs[idx] = np.delete(self.op_knobs[idx], 0)

    def avg_repeats(self):
        #Average repeat entries for the monitor
        knobs_same = True
35        iter_idx = 0
        while(iter_idx < len(self.mon)):
            curr_sum = 1
            while(knobs_same & (iter_idx + 1 < len(self.mon))):
                #Check all knobs are the same
40                for k_idx in self.op_knobs:
                    #print(self.op_knobs[k_idx][iter_idx]["val"])
                    #print(self.op_knobs[k_idx][iter_idx+1]["val"])
                    if(self.op_knobs[k_idx][iter_idx]["val"] != self.op_knobs[k_idx][iter_idx+1]["val
"]):
                        knobs_same = False

45                if(knobs_same):
                    #Only get here if knobs are all the same. Therefore can sum 2 mon entries (
average at end)
                    curr_sum += 1
                    self.mon[iter_idx]["val"] += self.mon[iter_idx+1]["val"]
50                    #print("sum: " + str(self.mon[iter_idx]["val"]))

                    #Remove repeat entry and corresponding knob entries
                    self.mon = np.delete(self.mon, iter_idx+1)
                    for k_idx in self.op_knobs:
55                        self.op_knobs[k_idx] = np.delete(self.op_knobs[k_idx], iter_idx+1)

                #Get here when while loop breaks
                #First do average of mon entry
                self.mon[iter_idx]["val"] = self.mon[iter_idx]["val"]/curr_sum
60                #print("avg: " + str(self.mon[iter_idx]["val"]))
                #move to next mon op entry
                iter_idx += 1
                knobs_same = True

65    def pick_last(self):
        #For temperature - remove repeat entries until temp has stabilised
        knobs_same = True
        iter_idx = 0
        while(iter_idx < len(self.mon)):
70            while(knobs_same & (iter_idx + 1 < len(self.mon))):
                #Check all knobs are the same
                for k_idx in self.op_knobs:
                    #print(self.op_knobs[k_idx][iter_idx]["val"])
                    #print(self.op_knobs[k_idx][iter_idx+1]["val"])
75                    if(self.op_knobs[k_idx][iter_idx]["val"] != self.op_knobs[k_idx][iter_idx+1]["val
"]):
                        knobs_same = False

                if(knobs_same):
                    #Only get here if knobs are all the same.
                    #Remove repeat entry and corresponding knob entries
80                    self.mon = np.delete(self.mon, iter_idx)

```

```

        for k_idx in self.op_knobs:
            self.op_knobs[k_idx] = np.delete(self.op_knobs[k_idx], iter_idx+1)

85     #Get here when while loop breaks
        #move to next mon op entry
        iter_idx += 1
        knobs_same = True

90     def multi_array_init(self, app_mon_cont, app_knob_disc, dev_knob_freq):

        self.op_app_knob_disc = dict()
        self.op_dev_knob_freq = dict()

95     #Init arrays using a line from each of the input knob arrays
        for adk_idx in app_knob_disc:
            self.op_app_knob_disc[adk_idx] = app_knob_disc[adk_idx][0]
            self.op_app_knob_disc[adk_idx] = np.delete(self.op_app_knob_disc[adk_idx], 0)

100    for dfk_idx in dev_knob_freq:
            self.op_dev_knob_freq[dfk_idx] = dev_knob_freq[dfk_idx][0]
            self.op_dev_knob_freq[dfk_idx] = np.delete(self.op_dev_knob_freq[dfk_idx], 0)

        for entry in app_mon_cont:
105            #print(entry)

            #App Knobs
            for adk_idx in app_knob_disc:
                #print(app_knob_disc[adk_idx][app_knob_disc[adk_idx]["ts"] < entry["ts"]][-1])
110                self.op_app_knob_disc[adk_idx] = np.append(self.op_app_knob_disc[adk_idx],
                    app_knob_disc[adk_idx][app_knob_disc[adk_idx]["ts"] < entry["ts"]][-1])

            #Dev Knobs
            for dfk_idx in dev_knob_freq:
                #print(dev_knob_freq[dfk_idx][dev_knob_freq[dfk_idx]["ts"] < entry["ts"]][-1])
115                self.op_dev_knob_freq[dfk_idx] = np.append(self.op_dev_knob_freq[dfk_idx],
                    dev_knob_freq[dfk_idx][dev_knob_freq[dfk_idx]["ts"] < entry["ts"]][-1])

            #Remove the first lines (added to create arrays at the start)
            for adk_idx in app_knob_disc:
                self.op_app_knob_disc[adk_idx] = np.delete(self.op_app_knob_disc[adk_idx], 0)

120        for dfk_idx in dev_knob_freq:
            self.op_dev_knob_freq[dfk_idx] = np.delete(self.op_dev_knob_freq[dfk_idx], 0)

```

Appendix G

Papers Published and Submitted

This appendix contains the full text for papers submitted and published during the PhD candidature. Papers are included in the following order:

- Leech, Charles, Vala, Charan Kumar, Acharyya, Amit, Yang, Sheng, Merrett, Geoffrey and Al-Hashimi, Bashir (2017) Run-time performance and power optimization of parallel disparity estimation on many-core platforms *ACM Transactions on Embedded Computing Systems* [25]
- Tenentes, Vasileios, Leech, Charles, Bragg, Graeme, Merrett, Geoffrey, Al-Hashimi, Bashir, Amrouch, Hussam, Henkel, Jörg and Das, Shidhartha (2017) Hardware and software innovations in energy-efficient system-reliability monitoring In *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*. IEEE. 5 pp. [26]
- Singh, Amit, Leech, Charles, Basireddy, Karunakar Reddy, Al-Hashimi, Bashir and Merrett, Geoffrey (2017) Learning-based run-time power and energy management of multi/many-core systems: current and future trends *Journal of Low Power Electronics* [27]
- Vala, Charan Kumar, Immadisetty, Koushik, Acharyya, Amit, Leech, Charles, Balagopal, Vibishna, Merrett, Geoff V. and Al-Hashimi, Bashir (2017) High-speed low-complexity guided image filtering-based disparity estimation *IEEE Transactions on Circuits and Systems I: Regular Papers*. [28]
- Leech, Charles and Kazmierski, T J (2014) Energy Efficient Multi-Core Processing *ELECTRONICS*, 18, (1), pp. 3-10. [29]
- Leech, Charles, Bragg, Graeme, Davis, James, Constantinides, George, Merrett, Geoffrey and Al-Hashimi, Bashir (2018) The PRiME Framework: Application- and Platform-agnostic Runtime Management *Transactions on Computer-Aided Design of Integrated Circuits and Systems* (Under Review)
- Leech, Charles, Raykov, Yordan P., Ozer, Emre and Merrett, Geoff V. (2017) Real-time room occupancy estimation with Bayesian machine learning using a single PIR sensor and microcontroller At *IEEE Sensors Applications Symposium (SAS)*, Glassboro, NJ, 2017, pp. [30]

Run-time Performance and Power Optimization of Parallel Disparity Estimation on Many-Core Platforms

Charles Leech, University of Southampton, UK
 Charan Kumar, IIT Hyderabad, India
 Amit Acharyya, IIT Hyderabad, India
 Sheng Yang, University of Southampton, UK
 Geoff V. Merrett, University of Southampton, UK
 Bashir M. Al-Hashimi, University of Southampton, UK

This paper investigates the use of many-core systems to execute the disparity estimation algorithm, used in stereo vision applications, as these systems can provide flexibility between performance scaling and power consumption. We present a learning-based runtime management approach which achieves a required performance threshold whilst minimizing power consumption through dynamic control of frequency and core allocation. Experimental results are obtained from a 61-core Intel Xeon-Phi platform for the above investigation. The same performance can be achieved with an average reduction in power consumption of 27.8% and increased energy efficiency by 30.04% when compared to DVFS control alone without runtime management.

CCS Concepts: •**Theory of computation** → *Online learning algorithms*; •**Computing methodologies** → *Scene understanding*; •**Computer systems organization** → *Parallel architectures*; *Embedded systems*; •**Hardware** → *Power estimation and optimization*; •**Software and its engineering** → *Multithreading*; *Power management*;

ACM Reference Format:

Charles Leech, Charan Kumar, Amit Acharyya, Sheng Yang, Bashir M. Al-Hashimi and Geoff V. Merrett, 2016. Run-time Performance and Power Optimization of a Parallel Disparity Estimation Algorithm on Many-Core Platforms. *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (YYYY), 20 pages.
 DOI: <http://dx.doi.org/10.1145/0000000.0000000>

1. INTRODUCTION

Stereo vision has become more pervasive in embedded and physically-constrained systems. Disparity estimation (DE) algorithms are used in stereo vision to calculate the depth of objects in a scene. They can be used in such applications as video surveillance, autonomous vehicles and mobile robots [Cyganek and Siebert 2009]. Algorithms need to satisfy real-time performance demands, while meeting high matching precision and low power consumption constraints. The choice of estimation algorithm and implementation platform are both important factors to meet these constraints and produce a viable embedded stereo matching system.

Achieving runtime power scalability without sacrificing real-time performance has emerged as the next challenge in this domain. We predict that software implementations on many-core architectures will allow dynamic scaling of algorithms and platforms to balance performance and power constraints. An adaptive runtime manage-

This work was supported in parts by the EPSRC Grant EP/L000563/1 and the PRiME Programme Grant EP/K034448/1 (www.prime-project.org). Experimental data used in this paper will be made available to download following the paper's acceptance.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© YYYY ACM. 1539-9087/YYYY-ARTA \$15.00
 DOI: <http://dx.doi.org/10.1145/0000000.0000000>

ment approach has been developed which employs a regression-based learning method to find power/performance trade-offs between frequency and core scaling, with the aim of increasing energy efficiency and extending device battery life [Shafik et al. 2015]. To the best of our knowledge, this is the first study that investigates a many-core implementation of the DE algorithm, employing runtime management to achieve a trade-off between power and performance. This paper provides the following contributions:

- Evaluation of the performance and power characteristics of a parallel implementation of a leading DE algorithm within a many-core operating space.
- An adaptive runtime management approach for power and performance modelling and optimisation of dynamic applications on many-core systems.
- Experimental validation of the runtime management approach for the DE algorithm using power and performance trade-offs on a many-core platform.

The rest of the paper is organized as follows: Section 2 discusses related work into disparity estimation and runtime management. Section 3 details the underlining algorithms for DE and their implementation. Section 4 describes the first motivational experiment for this work, where the algorithm is profiled on a many-core platform. Section 5 introduces the adaptive run-time manager for the many-core platform running the DE algorithm. Its validation and results follow in Section 6. Finally, Section 7 concludes the paper.

2. RELATED WORKS

2.1. Disparity Estimation

Stereo Vision for depth estimation and 3D sensing has been used across many embedded applications including person counting and tracking [Burbano et al. 2015], autonomous navigation and obstacle avoidance [Mendes and Wolf 2013; Li et al. 2016; Oleynikova et al. 2015] and mobile robotics [Karakaya et al. 2014; Solak and Bolat 2015].

At the highest level, DE can be categorised into global and local algorithms [Scharstein and Szeliski 2002]. Global algorithms are formulated as an optimisation across parts of the entire image. They produce precise results, with low average error rates in the calculation of disparity values [Scharstein and Szeliski 2002]. However, they typically have complex implementations. As a result, investigations have been made to implement dedicated hardware architectures of more precise algorithms, such as Semi Global Matching (SGM) [Gehrig et al. 2009; Banz et al. 2010] and Adaptive Support Weight (ADSW) [Chang et al. 2010; Ding et al. 2011; Perri et al. 2013]. Further improvements have been made to enable real-time processing, with a reduction in execution times of over 60% [Chang et al. 2010]. However, these have resulted in increased errors rates of up to 15% compared to software implementations [Banz et al. 2010]. In addition, high memory and hardware demands have the potential to limit scalability to higher resolution images.

In contrast, local stereo matching algorithms have reduced computational complexity and more localised memory requirements, relying on simpler aggregation strategies [Ttofis et al. 2016; L. Nalpantidis and Gasteratos 2008]. However, these algorithms are prone to disparity errors at depth discontinuity regions due to the use of a fixed local window shape and size [Yoon and Kweon 2006]. To improve matching accuracy, a few attempts have been made by combining or modifying existing algorithms and transforms [Ambrosch and Kubinger 2010; Baha and Larabi 2012; Zhang et al. 2009], the most recent Adaptive Support Weight (ADSW) methods are currently the most accurate [Gehrig et al. 2009; Ding et al. 2011; Perri et al. 2013]. They work by assigning different weights to the pixels in the support window based on their colour or prox-

imity to the central pixel. In this way, they aggregate only those pixels that lie at the same disparity, leading to improved quality at depth borders [Yoon and Kweon 2006].

Recently, the use of a Guided Image Filter (GIF) [He et al. 2013] in local ADSW algorithms has been proposed to reduce the complexity of cost aggregation, leading to a high-quality, fast and simple local DE algorithm [Hosni et al. 2011]. Due to the reduced complexity of this type of filter, the algorithm can operate at real-time frame-rates for HD images when implemented in a parallel structure [Ttofis et al. 2016]. This has resulted in the migration of software implementations entirely into the hardware domain on FPGAs [Gehrig et al. 2009; Banz et al. 2010; Ttofis et al. 2016].

We highlight the fact that these fixed hardware designs lack the ability to perform adaptations at run-time and that power-performance scalability is a key attribute for any application operating on an embedded system. We chose a local algorithm for our experimentation because it has scalability when implemented in software due to implicit parallelism and low data dependence properties. ADSW and GIF enhancements ensure a high quality disparity map in terms of bad pixel errors without sacrificing the algorithm's parallelism. Scalability enables operation across a range of power-performance points, depending on system constraints. Furthermore, the memory and computational resource requirements of embedded systems prevent the implementation of Global and SGM algorithms due to their irregular data access patterns and high complexity algorithms [Banz et al. 2010].

2.2. Runtime Power and Performance Optimization

Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Concurrency Throttling (DCT) are both runtime power optimisation approaches that have received significant attention over recent years [Shafik et al. 2015; Cochran et al. 2011]. DVFS is used to reduce energy consumption by lowering the operating voltage and frequency whilst causing acceptable performance degradation [Etinski et al. 2012]. DCT selects the number of concurrent processing cores and threads during runtime to manage application parallelism and exchange performance for energy [Porterfield et al. 2013; Shafik et al. 2015]. Both DVFS and DCT control have been used in conjunction as runtime control approaches to achieve minimized energy consumption and a required performance target [Curtis-Maury et al. 2008; Hwang and Chung 2013]. These approaches are based on offline training to learn the system architecture followed by online performance prediction to guide runtime optimization and adaptation.

Existing energy minimization approaches for parallel applications have the following limitations. Firstly, existing approaches [Porterfield et al. 2013] and [Curtis-Maury et al. 2008] ignore energy minimization in the sequential part of the application, which can be significant. Secondly, these approaches [Curtis-Maury et al. 2008; Hwang and Chung 2013] use offline training processes to learn the system architecture and control DVFS and/or DCT. As a result, their models are limited to single use-cases and their scalability is poor for different many-core architectural allocations of the same application. To address these limitations, our approach encompasses the entire application execution period and uses scalable adaptation based on online model training and an iterative control process to achieve optimised frequency and core settings.

3. DISPARITY ESTIMATION: ALGORITHM, IMPLEMENTATION AND VERIFICATION

Disparity estimation is a stereo matching process which can extract depth information from a pair of rectified, disparate images in a stereoscopic configuration. The correspondence of a pixel at coordinate (x, y) of the reference image, can be found at the same vertical coordinate y , within a maximum horizontal bound called the disparity range $[0, D)$ in the target image [Son et al. 2012]. The location difference of corresponding pixels in both images is called disparity and is used to calculate the depth in

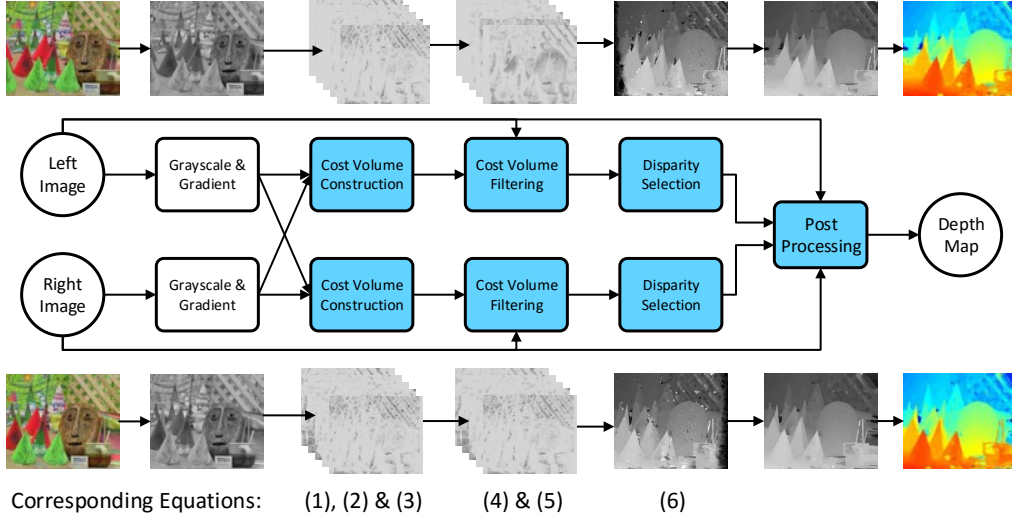


Fig. 1: Block diagram of the DE algorithm. The shaded regions are the enhanced parallel stages which offer opportunity for runtime adaptation.

meters. DE algorithms mostly follow four high-level steps: cost computation, cost aggregation, disparity computation and disparity refinement. This section describes the structure and properties of the DE algorithm, illustrated by Figure 1, highlighting the computationally intensive parts of the algorithm and how we have exploited inherent parallelism opportunities.

3.1. Algorithm

The algorithm is composed of four key stages, after pre-processing with gray-scale conversion and gradient computation from the input RGB colour images. The first stage of processing is Cost Volume Construction (CVC). This process involves the comparison of each pixel between the two images over a range of disparities d . A cost value is assigned to each pixel p in the left image based on the dissimilarity between it and a pixel in the right image. A truncated absolute difference of colours (1) and gradients (2) method is used to calculate the cost value. $M(p, d)$ and $G(p, d)$ are the cost contributions, at pixel p and disparity d , from the difference in colour and gradient respectively [Hosni et al. 2011]. I is the pixel value of each colour channel in equation (1) and the luminance in equation (2). ∇_x is the gradient in the x direction of the luminance of the image.

$$M(p, d) = \sum_{i=0}^3 |I_{left}^i(p) - I_{right}^i(p - d)| \quad (1)$$

$$G(p, d) = |\nabla_x(I_{left}(p)) - \nabla_x(I_{right}(p))| \quad (2)$$

A cost function (3) is used to balance the contribution from the colour difference or gradient difference using a weighting variable α . T_c and T_g are bounding threshold values for the colour and gradient cost contributions respectively for forming the overall cost.

$$C(p, d) = \alpha.min(T_c, M(p, d)) + (1 - \alpha).min(T_g, G(p, d)) \quad (3)$$

The second stage is Cost Volume Filtering (CVF), shown in equation (4), which is applied to the built cost volume. Filtering is performed using the Guided Image Filtering (GIF) method, using the original colour image as the guidance image, and is applied to each slice of the cost volume. The GIF is chosen because of its edge-preserving property, its non-approximate calculation, and its linear run time which is only dependent on image size and not kernel size. $q(p, d)$ is the filtered cost value at pixel p and disparity d and $C(p, d)$ is the unfiltered cost value at the same location.

$$q(p, d) = \sum W_{i,j}(I)C(p, d) \quad (4)$$

A weighting function $W_{i,j}$, equation (4), is used in the filter which favours pixels in the kernel that have similar colour to that of the central pixel. The filter operates in a square window ω_k , centred at pixel k with dimensions $r * r$. The covariance and mean of the of I in the window ω_k are given by σ_k and μ_k and the number of pixels in the filter window is $|\omega|$, with ϵ used as a smoothing parameter. One key advantage of GIF is that the filter weights can be computed from a set of linear equations (see [He et al. 2013]) which can be decomposed into a series of mean filter operations with radius r . These can be computed in $O(N)$ time where N is the number of pixel in the colour image.

$$W_{i,j} = \frac{1}{|\omega|^2} \sum_{k:(i,j) \in \omega_k} \left(1 + \frac{(I_i - \mu_k)(I_j - \mu_k)}{\sigma_k^2 + \epsilon} \right) \quad (5)$$

The third stage is disparity selection (equation 6) which is performed to generate the first instance of the disparity map which contains the initial depth information. Selection involves the condensation of the cost volume back into a single image and is performed through a winner-takes-all strategy to find the best disparity d_p value for each pixel p . D represents the upper bound of the disparity range ($0 \leq d < D$), within which the best disparity value must lie.

$$d_p = \underset{d \in D}{\operatorname{argmin}} q(p, d) \quad (6)$$

The lowest cost value for each pixel is identified from across all disparities in the cost volume. This represents the most likely distance of the same point in space between the two images. The corresponding disparity value is encoded in the disparity map.

Finally, post processing is applied to the disparity map. This involves occlusion detection, invalid pixel filling and bilateral filtering stages. A left-right consistency check can be performed because a disparity map is computed from both the left and right images' perspectives. This can be used to identify and fill mismatched pixels between the two maps with the closest consistent pixel. Following this, a bilateral filter is used to remove any remaining artefacts in the output disparity map by operating selectively only on the corrected pixel locations.

3.2. Software Implementation

The CVC and CVF stages (Equations 1 to 4) are the most computationally intensive parts of the algorithm and present opportunities to introduce high levels of parallelism. The shaded regions in Figure 1 illustrate where we have exploited this parallelism through multi-threading in order to create a large number of independent threads. The algorithm has been implemented in C++ and parallelism has been introduced to the CVC and CVF stages using the POSIX threads library.

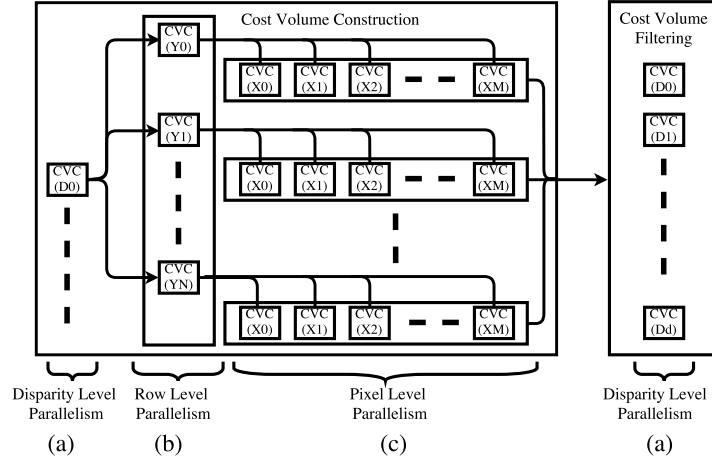


Fig. 2: The dimensions of parallelism introduced to cost volume construction and filtering.

Parallelism is present across three dimensions in the CVC stage and one dimension in the CVF stage. For both stages, the first level of parallelism is at the disparity level, partition (a) in Figure 2, where independent threads are created in the program for each disparity value. In CVC, additional parallelism is created within each disparity level to create per-row threads, $(Y0)$ to (YN) , shown vertically in partition (b) of Figure 2. From each of these, per-pixel threads are created to introduce the final level of parallelism, which is represented horizontally in Figure 2 from $(X0)$ to (XM) . N and M relate to the height and width of the input image respectively. Control over the maximum number of disparity levels that can be simultaneously computed provides the basis for core allocation.

3.3. Verification and Accuracy

Correct algorithmic function must be observed in order to verify that a demanding workload is being presented to the system. In addition, functional correctness was used to prove the successful introduction of parallelism and show that it does not adversely affect data accuracy. Table I shows that our algorithm is comparable, in terms of pixel errors per frame, to other works. Pixel error numbers are calculated for standard measures (nonocc, all, disc) across the four different image pairs of Tsukuba, Venus, Teddy and Cones, the later two of which are shown in Figure 3. The four input images (Figure 3a) and ground truth references (Figure 3c) were chosen from the widely-used Middlebury Stereo Vision dataset which provides a collection of stereo image resources for experimental purposes [Scharstein and Szeliski 2002]. The depth maps output from our algorithm are shown in Figure 3b for two test images for visual analysis of correct algorithmic function.

To analyse the effects of multi-threading and core scaling on the platform, experimentation with the algorithm described above is presented in the following section.

4. EXPERIMENTAL CHARACTERISATION OF MANY-CORE PERFORMANCE AND POWER CONSUMPTION

State-of-the-art commercial embedded platforms do not feature high core counts, especially in architectures where cores are individually configurable, therefore in emulation of future embedded many-core systems, the 61-core Intel Xeon Phi coprocessor

Table I: Comparison of the accuracy of related stereo matching algorithms using standard image pairs from the Middlebury database.

	Platform	Tsukuba			Venus			Teddy			Cones			Avg % bad pixel
		nonocc	all2	disc3	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	
[Mei et al. 2011]	GPU	1.07	1.48	5.73	0.09	0.25	1.15	4.10	6.22	10.9	2.42	7.25	6.95	3.97
[Bleyer et al. 2011]	CPU	2.09	2.33	9.31	0.21	0.39	2.62	2.99	8.16	9.62	2.47	7.80	7.11	4.59
[Wang et al. 2013]	FPGA	2.39	3.27	8.87	0.38	0.89	1.92	6.08	12.1	15.4	2.12	7.74	6.19	5.61
[Jin and Maruyama 2014]	FPGA	1.66	2.17	7.64	0.40	0.60	1.95	6.79	12.4	17.1	3.34	8.97	9.62	6.05
[Ttofis et al. 2016]	FPGA	2.38	3.01	9.38	0.40	0.7	3.62	7.23	12.7	17.2	2.87	8.59	8.27	6.36
Proposed	Xeon Phi	3	4.48	9.1	1.5	2.54	6.41	6	9.8	12.7	4.2	8.5	8.72	6.41
[Banz et al. 2010]	FPGA	4.1	-	-	2.7	-	-	11.4	-	-	8.4	-	-	6.7
[Hirschmuller 2008]	CPU	3.26	3.96	12.8	1.00	1.57	11.3	6.02	12.2	16.3	3.06	9.75	8.90	7.50
[Zhang et al. 2009]	CPU	1.99	2.65	6.77	0.62	0.96	3.20	9.75	15.1	18.2	6.28	12.7	12.9	7.60
[Shan et al. 2014]	FPGA	3.62	4.15	14.0	0.48	0.87	2.79	7.54	14.7	19.4	3.51	11.1	9.64	7.65
[Zhang et al. 2011]	FPGA	3.84	4.34	14.2	1.20	1.68	5.62	7.17	12.6	17.4	5.41	11.0	13.9	8.20
[Jin and Maruyama 2012]	FPGA	1.43	2.51	6.60	2.37	2.97	13.1	8.11	13.6	15.5	8.12	13.8	16.4	8.71
[Jin et al. 2010]	FPGA	9.79	11.6	20.3	3.59	5.27	36.8	12.5	21.5	30.6	7.34	17.6	21.0	17.2
[Shan et al. 2012]	FPGA	-	24.5	-	-	15.7	-	-	15.1	-	-	14.1	-	17.3
[Chang et al. 2010]	ASIC	20.4	20.6	47.9	15.3	16.6	29.5	25.1	32.4	34.1	22.9	31.1	30.6	27.2

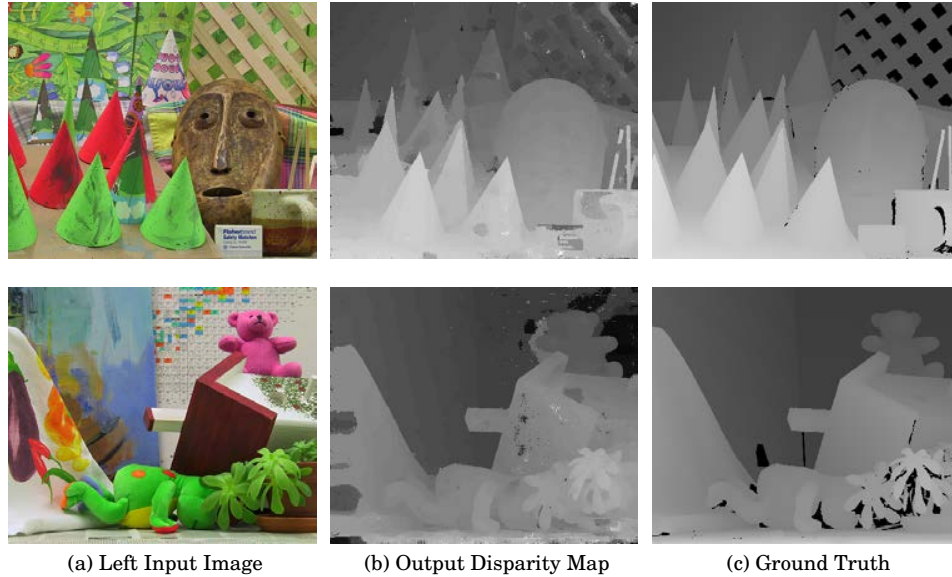


Fig. 3: Comparison of the depth map output from our algorithm and the ground truth depth map provided with the dataset.

is used as a demonstrative platform for core scaling [Intel 2015]. Although it is not strictly an embedded platform from an energy efficiency perspective, it is characteristic of the many-core parallel architectures that may feature in future embedded platforms which is why it is used in this experimentation.

To highlight the importance of system optimization for highly (but not embarrassingly) parallel algorithms, Figure 4 shows the power and performance trade-offs for running the DE algorithm on the Xeon Phi. Performance is defined as the Frames-Per-Second (fps) computed by the algorithm. Each point shows the performance and

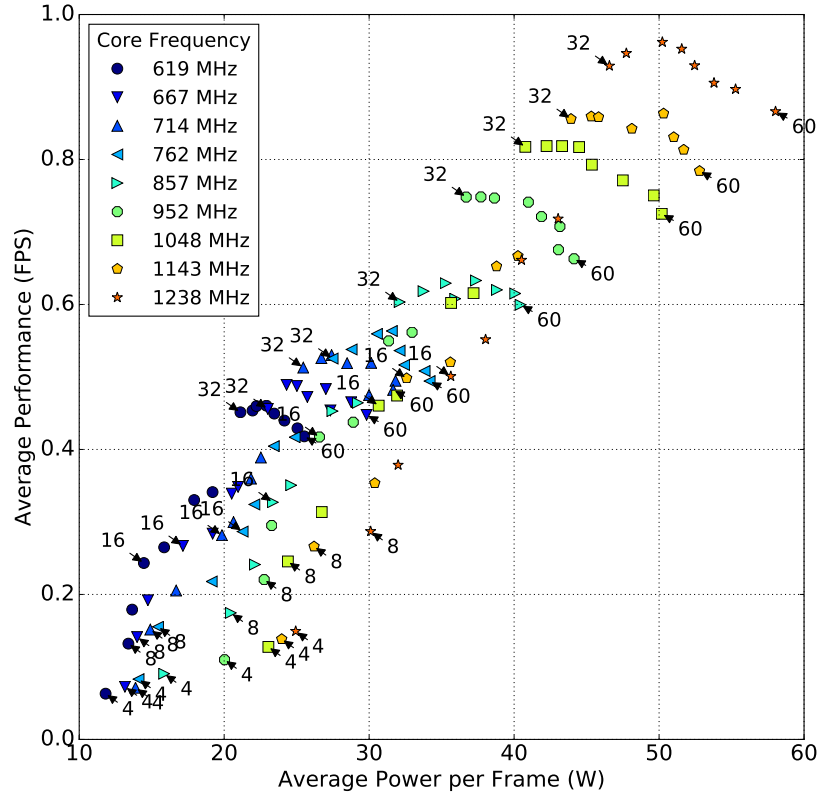


Fig. 4: Power and performance trade-offs for the possible range of cores and operating frequencies when executing the DE algorithm.

power consumption when executing the algorithm at each core count and frequency, sweeping active core number in intervals of 4 and frequency from 619 MHz through nine intervals to 1238 MHz. The labels attached to points show the number of active cores used at those operating points. Figure 4 shows that there is a 46W range in power consumption and over 15x range in performance that is attainable by operating at different frequency or core allocation points.

Four example operating points are shown in Table II. Going from p1 to p2 can be achieved similar performance (0.287 to 0.327) but with a 22% reduction in power consumption. Similarly, moving from p3 to p4 can yield a 3.1 times improvement in performance at approximately the same power consumption. For this application, using the maximum number of cores at maximum frequency (60 cores at 1238 MHz) does not yield the highest performance, yet it does consume the highest power. Therefore, DVFS alone is not enough to optimize power and performance. The next section introduces, adaptive runtime management which controls both frequency and core count to meet a target performance set by the DE algorithm.

5. PROPOSED ADAPTIVE RUNTIME MANAGEMENT

Figure 5 shows the block diagram for the runtime optimisation approach, highlighting the interactions between the application, runtime and hardware. The performance

Table II: Normalized power and performance trade-offs

Operating Point	p1	p2	p3	p4
Frequency (MHz)	1238	857	1238	667
Cores	8	16	4	32
Performance (fps)	0.287	0.327	0.149	0.456
Power (W)	30.10	23.35	24.93	23.01

target of the application and power constraint of the system are communicated to the runtime layer through the application monitors and system monitors framework. The dynamic adaptation of application parallelism, active core number and frequency is provided through the system and application controls framework. The runtime manager layer consists of two components: a learning-based power/performance model and a runtime controller. The model is derived and learned through runtime measurements of the system and application under different operating conditions (threads, voltage and frequency). The model then guides the runtime controller in optimizing the application's parallelism and the system DVFS controls to meet the application-specified performance targets. These two components are further detailed below.

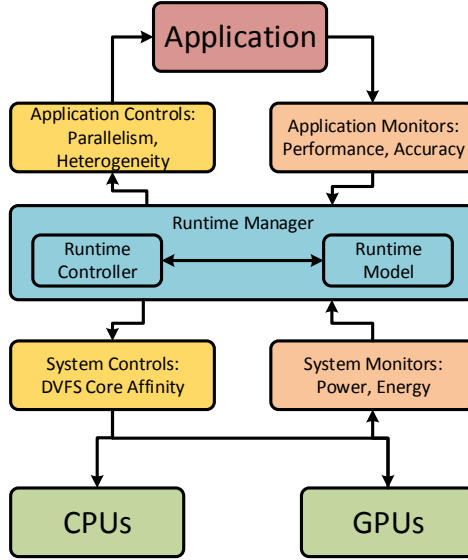


Fig. 5: Block diagram of the proposed run-time optimisation approach

5.1. Runtime Power and Performance Model

The runtime model simulates the current application and hardware configuration of the system (i.e. the DE algorithm on the Xeon Phi platform). Statically generated models can provide better accuracy, but involve extensive offline profiling of individual applications. A runtime model enables flexibility in terms of the application-system configuration, which can be learnt without the overhead associated with offline profiling. Through careful design of the runtime models, high accuracy can be achieved with little overhead (Section 5.2 and Section 6.2). Hence, our approach uses a runtime model as a critical component for energy-efficient adaptation. Such a runtime model enables

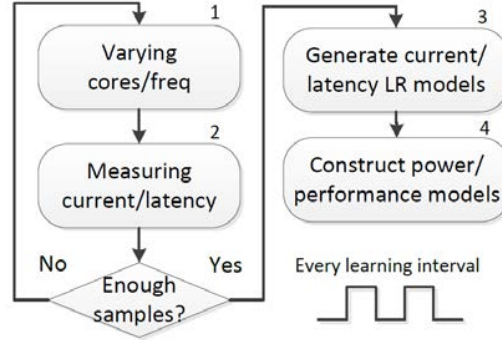


Fig. 6: Flowchart of the runtime energy/performance model generation

the prediction of power-performance trade-offs under different operating conditions. The model is learnt using runtime measurements from power sensors and application performance measurements.

Figure 6 shows how the model is learnt using linear regression in 4 steps. The modelling starts by varying the operating frequencies and number of active cores (step 1). For every frame, current and latency measurements are captured from the power sensors and the application (step 2). The measurements are used to test the hypotheses of the regression process until the learning interval is complete (Section 5.2 justifies the choice of the learning interval). After this interval, current and latency models are generated for the given application running on the platform (step 3). These models are combined to derive the power and performance models (step 4).

Linear regression is used to establish a relationship between the dependent variables of the system (power and performance) and their associated independent predictor variables (i.e. number of cores, VF levels, etc.) [Cohen et al. 2013]. The relationship is defined by a hypothesis function as:

$$h_{\theta}(x) = \sum_{i=0}^n \theta_i x_i = \Theta^T X \quad . \quad (7)$$

where x_i is a predictor, n is the number of predictors, θ_i is a fitting coefficient, X and $\Theta^T X$ are the matrix representations of x_i and θ_i . The Θ values need to be chosen to minimize the mean-squared prediction error ($J(\theta)$) of the hypothesis in (7), which is given by:

$$\begin{aligned} J(\theta) &= \sum_{j=1}^m (h_{\theta}(x^{(j)}) - y^{(j)})^2 \\ &= (\Theta^T X - \vec{y})^T (\Theta^T X - \vec{y}), \end{aligned} \quad (8)$$

where y is the measured value, m is number of learning samples. $J(\theta)$ is minimum when its gradient becomes 0. Hence, from (8) the gradient of $J(\theta)$ can be defined as:

$$\begin{aligned} \nabla J(\theta) &= \nabla (\Theta^T X - \vec{y})^T (\Theta^T X - \vec{y}) \\ &= X^T X \Theta - X^T \vec{y}. \end{aligned} \quad (9)$$

From (9), the fitting coefficients Θ of the hypothesis in (7) can then be computed as:

$$\Theta = (X^T X)^{-1} X^T \vec{y} \quad . \quad (10)$$

Table III: Modelling Hypotheses

Model	Hypothesis $h_\theta(x)$
Latency (τ)	$\theta_0 + \theta_1 \cdot \frac{1}{f} + \theta_2 \cdot \frac{1}{f \cdot c} + \theta_3 \cdot c$
Current (i)	$\theta_0 + \theta_1 \cdot v + \theta_2 \cdot v \cdot f \cdot c$
Performance	$F_{perf}(f) = 1/\tau$
Power	$F_{pow}(f) = v \cdot i$

From (10), the computation complexity of the regression-based modelling is $O(n^2 \times m)$, where n is the number of predictors and m is the number of learning samples. Hence, to achieve a fast runtime model both n and m need to be small. In this work, two predictors are used: number of cores (c) and frequency (f), together with the intercept; hence, $n=3$.

Performance and power are not linear functions of frequency and the number of cores, therefore we first generate models for output current (i) and latency (τ), then build performance and power models from these. Table III shows the different hypotheses used to generate the models. Column 1 shows the target model and column 2 shows the hypothesis used. These models and their hypotheses are explained further as follows:

(1) Latency (τ) is expressed as a sum of four terms: the first term is a constant (θ_0) meaning delay contributed by factors independent of multi-threading and frequency (such as memory contention, I/O setup, etc); the second term ($\theta_1 \cdot \frac{1}{f}$) is proportional to the CPU clock period representing the time spend by the sequential part of the application; the third term ($\theta_2 \cdot \frac{1}{f \cdot c}$) is proportional to both the clock period and number of cores, representing the time spent by the parallel part of the application; the last term ($\theta_3 \cdot c$) shows the latency related to the effects of multi-threading.

(2) Current (i) is expressed as a sum of three terms: the first two terms ($\theta_0 + \theta_1 \cdot v$) approximate the leakage current, while the last term ($\theta_2 \cdot v \cdot f \cdot c$) signifies the dynamic current.

(3) Performance in units of Frames-Per-Second (fps) is expressed as inversely proportional to the latency.

(4) Power consumption is expressed as a product of the instantaneous current (i) and supply voltage (v).

The supply voltage used in these hypotheses (Table III) is derived as a direct function of the operating frequency as it is fixed by the frequency controller based on the selected frequency. The regression-based learning of the power/performance trade-offs and their validations are further detailed in Section 5.2 where we demonstrate the impact that the number of learning samples has on the model prediction accuracy and associated runtime overheads.

5.2. Runtime Model Validation

Validation of the runtime model is carried out in two stages. In the first stage, the hypotheses of Table III are established as linear models. The models are then used at runtime in the second stage.

Figure 7a and 7b show scatter plots of the measured power and the performance data of the DE algorithm executing on the platform under different operating conditions (frequency and number of cores). A subset of this data is used as training samples to generate the power and performance models of Figure 7c and 7d through the hypotheses of Table III. The models show predicted power and performance values across the full range of operating conditions. During application execution, the runtime models

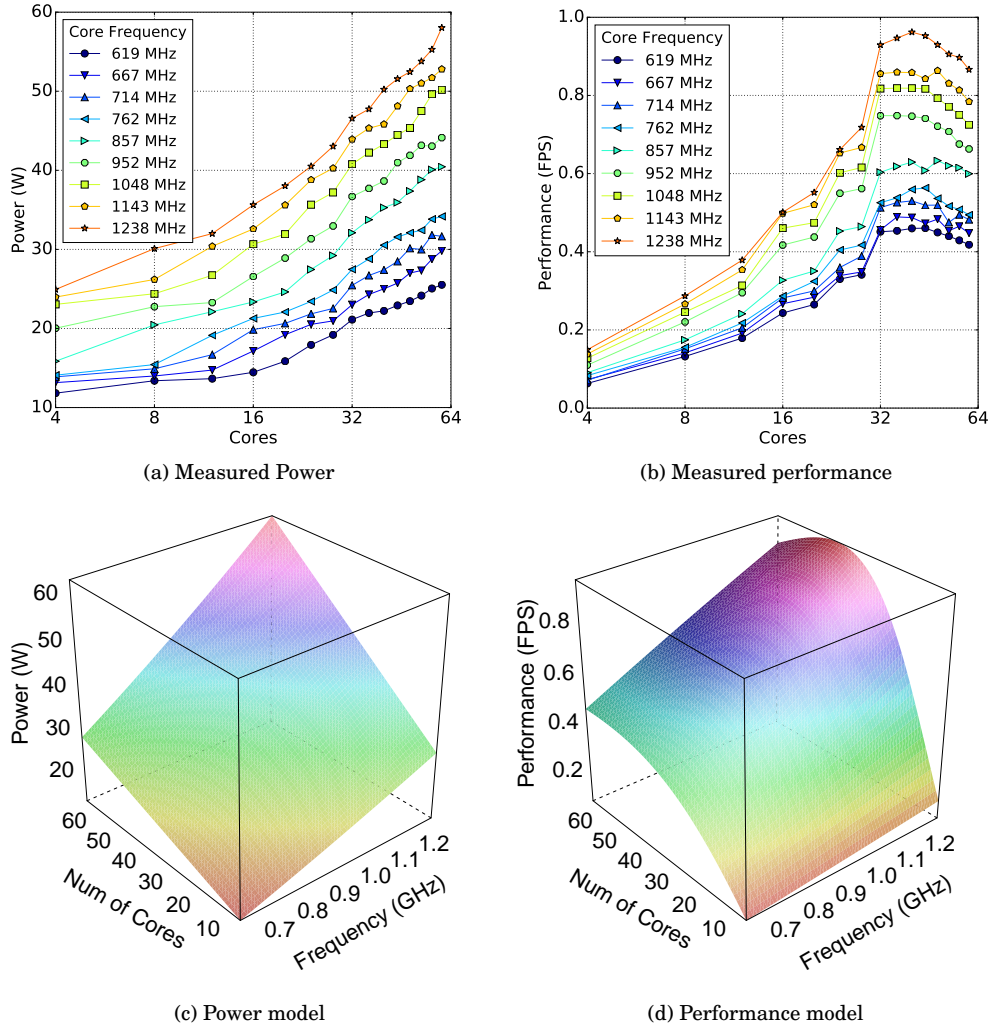


Fig. 7: Plots of measured power (a) and performance (b) under different operating conditions, and runtime models for power (c) and performance (d) generated from training data.

are interrogated every time the application or system constraints change to determine predicted power and performance for new optimal operating points.

The modelled values exhibit a high degree of correlation with the measured values. This is because the runtime model (Section 5.1) is generated using realistic component models of current (I) and latency (τ) from measured data. The accuracy of this model depends on a number of factors; the number of samples acquired, the number of predictors, and the underlying relationships between current, latency, performance and power. Figure 8 shows how many training points (frequency and core number) is required to achieve error convergence and low absolute power (Figure 8(a)) and performance (Figure 8(b)) modelling errors. The x -axis shows the number of core training

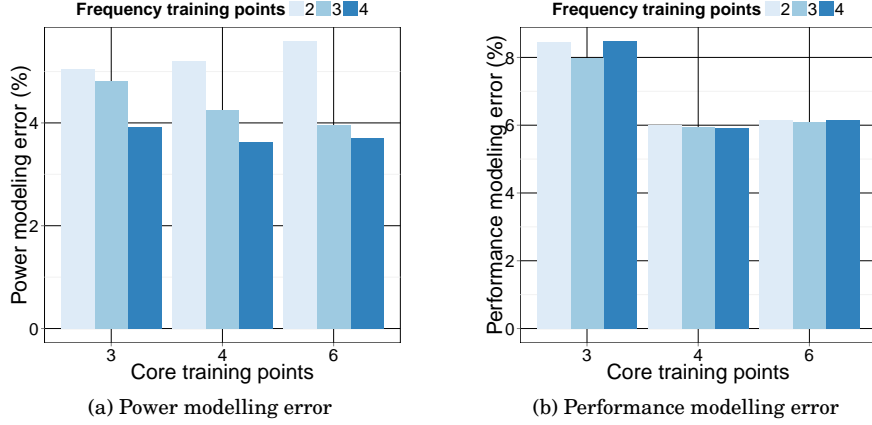


Fig. 8: Evaluation of how increasing the number of training samples for both core count and frequency reduces the power and performance modelling error.

points and the bars show the different number of frequency training points. The total number of training points is the product of the core and frequency training points. Modelling error is calculated using a comparison between the predicted power and performance values and measured data under the same conditions. The validation data set is exclusive of the training samples used to generate the models. The modelling error in terms of miss-prediction reduces with an increase in the number of training points. Convergence of the modelling error is significant after four core and three frequency training points. This is deemed as when the reduction in power and performance modelling error is less than 0.5% between training samples. The absolute performance and power model errors at this point are 5.95% and 4.25% respectively. This result shows that the runtime models require a low convergence interval of only 12 training points. This is because regression with a small number of predictors is rigid and the variance in the model is small [Draper and Smith 1998]. The high modelling accuracy helps the runtime manager to achieve near-optimal operational conditions. A demonstration of this is presented in Section 6.

5.3. Runtime Optimization

The runtime model enables runtime optimization of power and performance through DVFS and core controls. Using the model coefficients, the runtime controller uses a gradient-descent based search in the optimization space to quickly predict the optimal number of cores and frequency controls for each power/performance constraint. Algorithm 1 shows example pseudo-code for the gradient-descent process when a performance target is used as the input. The objective is to find the minimum power point, frequency and core number as the output. The operating frequencies (f_n) and number of cores (c_n) are initialized (line 1). These are updated by a gradient descent (line 3-4). The learning rate (α) in the algorithm is also initialized to a high value to ensure a fast convergence. While the updated F_{perf} exceeds the specified performance target (line 5), the learning rate is reduced (line 6) and f_n and c_n are further updated by another gradient descent (line 7-8). Predictions and updates are continued until the minimum performance target is met. The operating frequency and number of cores that provides

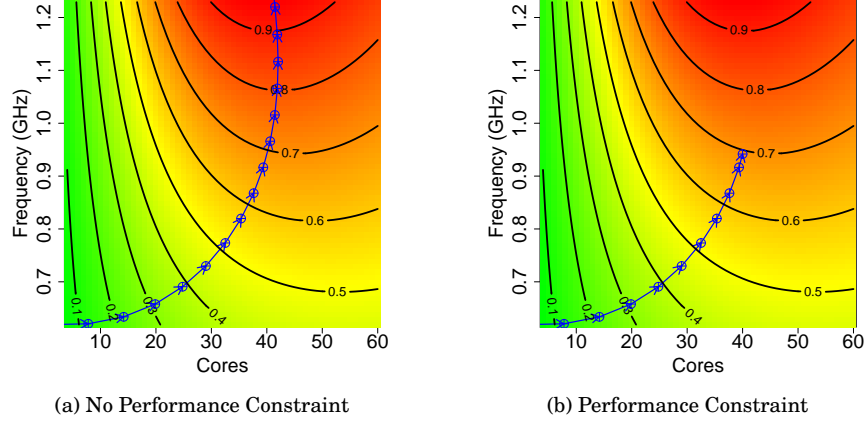


Fig. 9: Runtime optimization examples (a) without performance constraint and (b) with performance constraint.

the minimum power consumption while meeting the specified performance target are returned (f_n, c_n).

ALGORITHM 1: Gradient descent based performance constrained power optimization through DVFS and core control

Input: Performance Requirement: $Perf_{req}$

Output: Minimum power point: Pwr_{min} , operating freq: f and number of cores: c

```

1 Initialize:  $f_n, c_n$  and learning rate:  $\alpha$ 
2 repeat
3    $f_{n+1} := f_n - \alpha \frac{\partial}{\partial f} F_{perf}(f_n, c_n)$ 
4    $c_{n+1} := c_n - \alpha \frac{\partial}{\partial c} F_{perf}(f_n, c_n)$ 
5   while  $F_{perf}(f_{n+1}, c_{n+1}) > Perf_{req}$  do
6      $\alpha := 0.5 \times \alpha$ 
7      $f_{n+1} := f_n - \alpha \frac{\partial}{\partial f} F_{perf}(f_n, c_n)$ 
8      $c_{n+1} := c_n - \alpha \frac{\partial}{\partial c} F_{perf}(f_n, c_n)$ 
9   end
10   $f_{n-1} := f_n, f_n := f_{n+1}$ 
11   $c_{n-1} := c_n, c_n := c_{n+1}$ 
12 until  $f_n = f_{n+1} \ \& \ c_n = c_{n+1}$ ;
13 return  $F_{perf}(f_n, c_n)$  as  $Perf_{max}$ 

```

An example of DVFS and core controls through Algorithm 1 is shown in Figure 9a and illustrates how the gradient descent algorithm finds the optimum operating point. Power consumption is shown using an overlaid colour map and the performance level is shown with contour lines. To determine the optimal DVFS controls and core allocation, the power and performance are predicted using the lowest operating frequencies (f_i) and core allocation (c_i) initially. For each next operating frequency during the search, the step size is reduced with the gradient. The gradient-descent heuristic search starts at one core with an operating frequency of 619 MHz, giving 0.017 predicted fps and 11

Table IV: Optimized operating points, sampled from Figure 10, that minimized power consumption under each performance target.

Time (s)	% of Target Perf	Cores	Frequency (MHz)	Energy (J)	Avg Power (W)
6.82	112	16	1238	60.63	33.62
31.4	98.5	52	952	61.28	40.95
51.3	98.7	56	1048	65.25	48.10
74.7	100	56	952	64.14	43.06
94.8	98.0	40	667	53.39	26.06
111.9	106	12	1048	78.38	25.86
130.8	116	20	619	59.25	16.90
154.4	115	28	667	54.40	21.38
172.2	104	48	762	58.29	32.29
195.3	95.7	52	952	63.02	42.30

W, followed by 8 cores at 619 MHz giving 0.12 fps and 13 W. The process continues until it converges at 41 cores with a closest operating frequency of 1238 MHz giving the highest performance of 0.99 fps and an average power consumption of 57 W. Figure 9b demonstrates another example of the same algorithm applied with a performance target. This time the search only continues until the performance constraint is met, at which point the corresponding operating frequency and core allocation is selected (40 cores at 952 MHz). The effectiveness of runtime optimization of core number and DVFS controls is further validated in Section 6.

6. EXPERIMENTAL RESULTS - ADAPTIVE RUNTIME MANAGEMENT

This section examines the effectiveness of the proposed runtime management approach, its overheads and verifies the accuracy of the DE algorithm's output. The proposed runtime approach is engineered into a prototype runtime manager and framework as part of a complete system; implementing application, runtime and hardware, on the Xeon Phi platform. The platform uses the same voltage-frequency island for all the cores. The operating frequency can be varied from 619 MHz to 1238 MHz in 9 steps, and the corresponding voltage varies from 0.995 V to 1.060 V. The relationship between the supply voltage v and operation frequency f has been found empirically and can be approximated as:

$$v = 0.93 + 0.11f \quad (11)$$

where frequency is in GHz. Core allocation and DVFS is controlled through the runtime manager using the runtime power/performance model.

6.1. Online Adaptation of the Disparity Estimation Algorithm

In section 4, the algorithm's power-performance operating space was characterised; this experiment demonstrates the online adaptation process that the RTM operates as part of a complete system. After runtime modelling of the algorithm's power and performance trade-offs is completed, the proposed approach can accurately adapt to changes in the performance target through selection of the most energy-efficient core allocation and DVFS control settings. These properties are demonstrated in the three time series graphs of Figure 10. Furthermore, a comparison is made to the default Linux frequency governor and scheduler. These experiments are shown as dashed lines. Direct comparison between the proposed runtime management approach and existing work is not possible because of the multi-dimensional optimisation space created in our implementation of the algorithm.

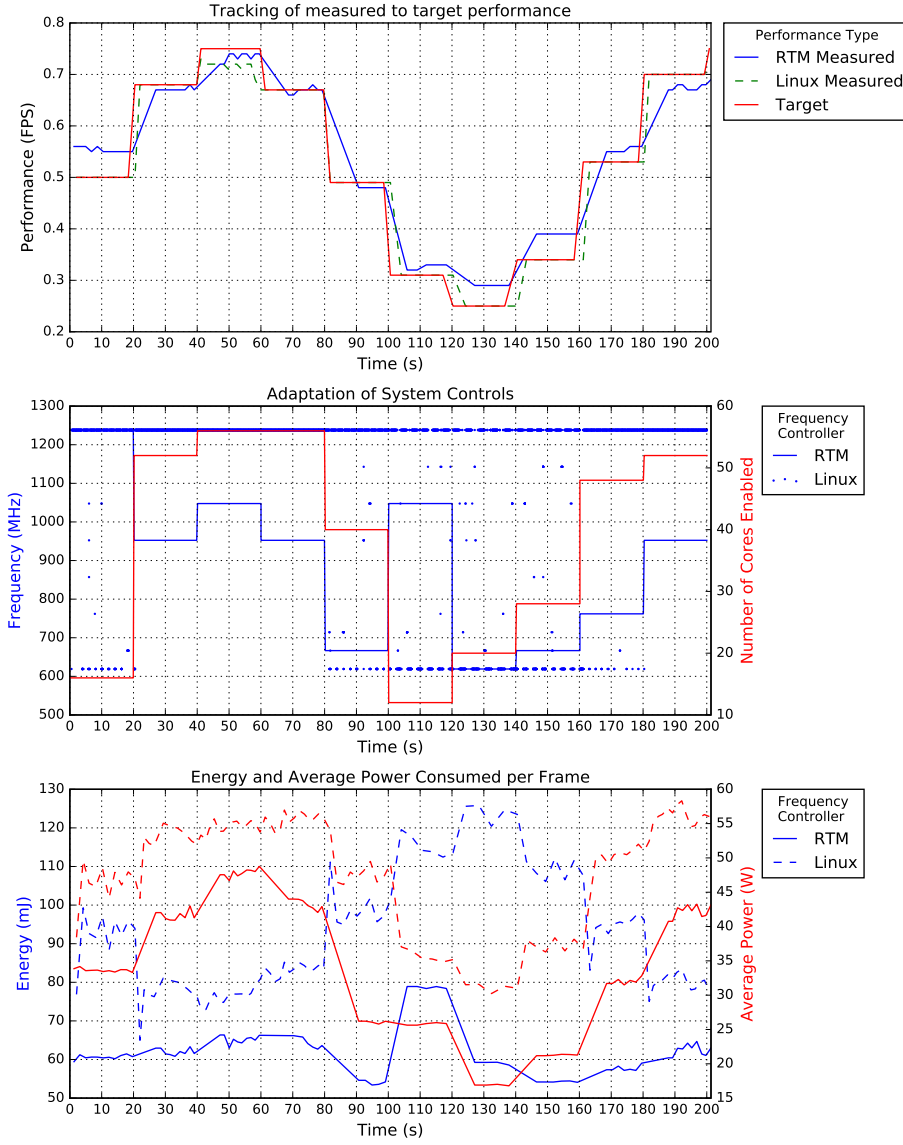


Fig. 10: Time series analysis of the RTM performing online adaptations of core number and frequency to optimise power and energy whilst meeting a target performance.

Changes in target performance could be due to a multitude of factors depending on the application, in our case a higher performance is required to enable the system to calculate the depth of objects in the scene moving at a higher speed. Adaptation to changes in the performance target can be seen in the top series where the measured performance tracks the target as it changes over time. The average absolute error in measured performance is 5.56%. Excluding occasions when the measured performance exceeds the target performance, which is not considered a penalty, this error drops to

1.16%. Performance of the application under Linux is manually matched to the target to allow the governor to make DVFS changes once frame processing is complete, as a result, the measured performance tracks the target very tightly.

The online adaptation of the number of active cores and their frequency is shown in the second series of Figure 10. The RTM interrogates the power/performance model in order to predict which operating point will give the target performance with the lowest power consumption. It can be observed that changes in core number and frequency occur independently of each other and are only loosely correlated. This is due to the overlapping of operating points, in terms of power and performance, which use a high number of cores at a low frequency with those which use a lower number of cores at a higher frequency. For example, using data from Table IV, at 112 seconds a target of 0.31 fps leads to 12 cores at 1048 MHz whereas at 154 seconds, the similar target of 0.34 select a very different operating point at 28 cores at 667 MHz. This shows how the RTM model captures the power and performance trade-offs between DVFS and core scaling in an application. Moreover, there is a stronger trend in the hypothesis that an increased performance target will result in both a higher frequency and number of cores, as can be seen from 120 seconds onwards.

The final series shows the measured average power and energy consumed per frame as the performance target fluctuates, for both the proposed approach and the default Linux. To achieve the same performance, the proposed approach can reduce average power consumption by 27.8% and increase energy efficiency by 30.04%. The proposed approach consistently gives lower power consumption and increased energy efficiency across the entire range of performance targets. This is primarily due to the core scaling ability of the RTM and its rigorous control of frequency.

6.2. Runtime Manager Overheads

The proposed approach incurs runtime overheads due to various adaptation steps, including optimization, control and monitoring operations. Learning the runtime model incurs a small overhead of 2 ms in calculating the linear regression coefficients. Runtime adaptation exhibits the small overhead of 2.4 ms due to the gradient-descent based search operation performed on the runtime model. Adaptation only takes place when the performance target or power constraint is changed. Collecting training data takes a period of 12 frames, which must be completed before optimization begins. Training time is application-dependent, for disparity estimation, it takes approximately 10 seconds depending on the operating points which training frames are based on.

7. CONCLUSIONS

This work shows that many-core systems present a viable approach to the implementation of the disparity estimation algorithm and other parallel embedded applications. This paper has investigated the implementation of a parallel disparity estimation algorithm on a many-core platform. An adaptive runtime power and performance optimization approach has been presented to reduce the power consumption of the system. We have reported a performance and power trade-off, demonstrating that it is possible to achieve the same performance with lower power consumption and higher energy efficiency by optimizing frequency and core allocation.

REFERENCES

- Kristian Ambrosch and Wilfried Kubinger. 2010. Accurate Hardware-based Stereo Vision. *Comput. Vis. Image Underst.* 114, 11 (Nov 2010), 1303–1316. DOI: <http://dx.doi.org/10.1016/j.cviu.2010.07.008>

- Nadia Baha and Slimane Larabi. 2012. Accurate Real-time Neural Disparity MAP Estimation with FPGA. *Pattern Recogn.* 45, 3 (March 2012), 1195–1204. DOI: <http://dx.doi.org/10.1016/j.patcog.2011.08.005>
- C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch. 2010. Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation. In *Embedded Computer Systems (SAMOS), 2010 International Conference on.* 93–101. DOI: <http://dx.doi.org/10.1109/ICSAMOS.2010.5642077>
- Michael Bleyer, Christoph Rhemann, and Carsten Rother. 2011. PatchMatch Stereo - Stereo Matching with Slanted Support Windows. In *British Machine Vision Conference 2011*. 1–11. http://publik.tuwien.ac.at/files/PubDat_201949.pdf Vortrag; British Machine Vision Conference BMVC 2011, Dundee; 2011-08-29 – 2011-09-02.
- A. Burbano, S. Bouaziz, and M. Vasiliu. 2015. 3D-sensing Distributed Embedded System for People Tracking and Counting. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*. 470–475. DOI: <http://dx.doi.org/10.1109/CSCI.2015.76>
- N. Y. C. Chang, T. H. Tsai, B. H. Hsu, Y. C. Chen, and T. S. Chang. 2010. Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight. *IEEE Transactions on Circuits and Systems for Video Technology* 20, 6 (June 2010), 792–805. DOI: <http://dx.doi.org/10.1109/TCSVT.2010.2045814>
- Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. 2011. Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*. ACM, New York, NY, USA, 175–185. DOI: <http://dx.doi.org/10.1145/2155620.2155641>
- J. Cohen, P. Cohen, S.G. West, and L.S. Aiken. 2013. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. Taylor & Francis. <https://books.google.co.uk/books?id=fAnSOgbdFXIC>
- Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S. Nikolopoulos, Bronis R. de Supinski, and Martin Schulz. 2008. Prediction Models for Multi-dimensional Power-performance Optimization on Many Cores. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT '08)*. ACM, New York, NY, USA, 250–259. DOI: <http://dx.doi.org/10.1145/1454115.1454151>
- B. Cyganek and J. P. Siebert. 2009. *Introduction to 3D Computer Vision Techniques and Algorithms*. Wiley-Blackwell.
- Jingting Ding, Jilin Liu, Wenhui Zhou, Haibin Yu, Yanchang Wang, and Xiaojin Gong. 2011. Real-time stereo vision system using adaptive weight cost aggregation approach. *EURASIP Journal on Image and Video Processing* 2011, 1 (2011), 1–19. DOI: <http://dx.doi.org/10.1186/1687-5281-2011-20>
- N. R. Draper and H. Smith. 1998. *Applied Regression Analysis* (3rd ed.). Wiley-Blackwell.
- M. Etinski, J. Corbalan, J. Labarta, and M. Valero. 2012. Understanding the Future of Energy-performance Trade-off via DVFS in HPC Environments. *J. Parallel Distrib. Comput.* 72, 4 (April 2012), 579–590. DOI: <http://dx.doi.org/10.1016/j.jpdc.2012.01.006>
- Stefan K. Gehrig, Felix Eberli, and Thomas Meyer. 2009. *A Real-Time Low-Power Stereo Vision Engine Using Semi-Global Matching*. Springer Berlin Heidelberg, Berlin, Heidelberg, 134–143. DOI: http://dx.doi.org/10.1007/978-3-642-04667-4_14
- K. He, J. Sun, and X. Tang. 2013. Guided Image Filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 6 (June 2013), 1397–1409. DOI: <http://dx.doi.org/10.1109/TPAMI.2012.213>

- H. Hirschmuller. 2008. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 2 (Feb 2008), 328–341. DOI: <http://dx.doi.org/10.1109/TPAMI.2007.1166>
- A. Hosni, M. Bleyer, C. Rhemann, M. Gelautz, and C. Rother. 2011. REal-time local stereo matching using guided image filtering. In *2011 IEEE International Conference on Multimedia and Expo*. 1–6. DOI: <http://dx.doi.org/10.1109/ICME.2011.6012131>
- Y. S. Hwang and K. S. Chung. 2013. Dynamic Power Management Technique for Multi-core Based Embedded Mobile Devices. *IEEE Transactions on Industrial Informatics* 9, 3 (Aug 2013), 1601–1612. DOI: <http://dx.doi.org/10.1109/TII.2012.2232299>
- Intel. 2015. Intel Xeon Phi Product Family. Online. (2015).
- Minxi Jin and Tsutomu Maruyama. 2012. A Real-time Stereo Vision System Using a Tree-structured Dynamic Programming on FPGA. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '12)*. ACM, New York, NY, USA, 21–24. DOI: <http://dx.doi.org/10.1145/2145694.2145698>
- Minxi Jin and Tsutomu Maruyama. 2014. Fast and Accurate Stereo Vision System on FPGA. *ACM Trans. Reconfigurable Technol. Syst.* 7, 1, Article 3 (Feb. 2014), 24 pages. DOI: <http://dx.doi.org/10.1145/2567659>
- S. Jin, J. Cho, X. D. Pham, K. M. Lee, S. K. Park, M. Kim, and J. W. Jeon. 2010. FPGA Design and Implementation of a Real-Time Stereo Vision System. *IEEE Transactions on Circuits and Systems for Video Technology* 20, 1 (Jan 2010), 15–26. DOI: <http://dx.doi.org/10.1109/TCSVT.2009.2026831>
- S. Karakaya, G. Kkyildiz, C. Toprak, and H. Ocak. 2014. Development of a human tracking indoor mobile robot platform. In *Proceedings of the 16th International Conference on Mechatronics - Mechatronika 2014*. 683–687. DOI: <http://dx.doi.org/10.1109/MECHATRONIKA.2014.7018343>
- G. C. Sirakoulis L. Nalpantidis and A. Gasteratos. 2008. Review of stereo vision algorithms: From software to hardware. *International Journal of Optomechatronics* 2, 4 (Jan 2008), 435–462.
- J. Li, Y. Liu, S. Du, P. Wu, and Z. Xu. 2016. Hierarchical and Adaptive Phase Correlation for Precise Disparity Estimation of UAV Images. *IEEE Transactions on Geoscience and Remote Sensing* 54, 12 (Dec 2016), 7092–7104. DOI: <http://dx.doi.org/10.1109/TGRS.2016.2595861>
- X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and Xiaopeng Zhang. 2011. On building an accurate stereo matching system on graphics hardware. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. 467–474. DOI: <http://dx.doi.org/10.1109/ICCVW.2011.6130280>
- Caio César Teodoro Mendes and Denis Fernando Wolf. 2013. Real Time Autonomous Navigation and Obstacle Avoidance Using a Semi-global Stereo Method. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13)*. ACM, New York, NY, USA, 235–236. DOI: <http://dx.doi.org/10.1145/2480362.2480413>
- H. Oleynikova, D. Honegger, and M. Pollefeys. 2015. Reactive avoidance using embedded stereo vision for MAV flight. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 50–56. DOI: <http://dx.doi.org/10.1109/ICRA.2015.7138979>
- Stefania Perri, Pasquale Corsonello, and Giuseppe Cocorullo. 2013. Adaptive Census Transform: A Novel Hardware-oriented Stereo vision Algorithm. *Comput. Vis. Image Underst.* 117, 1 (Jan. 2013), 29–41. DOI: <http://dx.doi.org/10.1016/j.cviu.2012.10.003>
- Allan K. Porterfield, Stephen L. Olivier, Sridutt Bhalachandra, and Jan F. Prins. 2013. Power Measurement and Concurrency Throttling for Energy Reduction in OpenMP Programs. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD*

- Forum (IPDPSW '13)*. IEEE Computer Society, Washington, DC, USA, 884–891. DOI: <http://dx.doi.org/10.1109/IPDPSW.2013.15>
- Daniel Scharstein and Richard Szeliski. 2002. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *Int. J. Comput. Vision* 47, 1-3 (April 2002), 7–42. DOI: <http://dx.doi.org/10.1023/A:1014573219977>
- Rishad A. Shafik, Anup Das, Sheng Yang, Geoff Merrett, and Bashir M. Al-Hashimi. 2015. Adaptive Energy Minimization of OpenMP Parallel Applications on Many-Core Systems. In *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures (PARMA-DITAM '15)*. ACM, New York, NY, USA, 19–24. DOI: <http://dx.doi.org/10.1145/2701310.2701311>
- Yi Shan, Yuchen Hao, Wenqiang Wang, Yu Wang, Xu Chen, Huazhong Yang, and Wayne Luk. 2014. Hardware Acceleration for an Accurate Stereo Vision System Using Mini-Census Adaptive Support Region. *ACM Trans. Embed. Comput. Syst.* 13, 4s, Article 132 (April 2014), 24 pages. DOI: <http://dx.doi.org/10.1145/2584659>
- Y. Shan, Z. Wang, W. Wang, Y. Hao, Y. Wang, K. Tsoi, W. Luk, and H. Yang. 2012. FPGA based memory efficient high resolution stereo vision system for video tolling. In *Field-Programmable Technology (FPT), 2012 International Conference on*. 29–32. DOI: <http://dx.doi.org/10.1109/FPT.2012.6412106>
- S. Solak and E. D. Bolat. 2015. Distance estimation using stereo vision for indoor mobile robot applications. In *2015 9th International Conference on Electrical and Electronics Engineering (ELECO)*. 685–688. DOI: <http://dx.doi.org/10.1109/ELECO.2015.7394442>
- Hyeon-Sik Son, Kyeong-ryeol Bae, Seung-Ho Ok, Yong-Hwan Lee, and Byungin Moon. 2012. *A Rectification Hardware Architecture for an Adaptive Multiple-Baseline Stereo Vision System*. Springer Berlin Heidelberg, Berlin, Heidelberg, 147–155. DOI: http://dx.doi.org/10.1007/978-3-642-27192-2_19
- C. Ttofis, C. Kyrkou, and T. Theodoridis. 2016. A Low-Cost Real-Time Embedded Stereo Vision System for Accurate Disparity Estimation Based on Guided Image Filtering. *IEEE Trans. Comput.* 65, 9 (Sept 2016), 2678–2693. DOI: <http://dx.doi.org/10.1109/TC.2015.2506567>
- W. Wang, J. Yan, N. Xu, Y. Wang, and F. H. Hsu. 2013. Real-time high-quality stereo vision system in FPGA. In *Field-Programmable Technology (FPT), 2013 International Conference on*. 358–361. DOI: <http://dx.doi.org/10.1109/FPT.2013.6718387>
- Kuk-Jin Yoon and In So Kweon. 2006. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 4 (April 2006), 650–656. DOI: <http://dx.doi.org/10.1109/TPAMI.2006.70>
- K. Zhang, J. Lu, and G. Lafruit. 2009. Cross-Based Local Stereo Matching Using Orthogonal Integral Images. *IEEE Transactions on Circuits and Systems for Video Technology* 19, 7 (July 2009), 1073–1079. DOI: <http://dx.doi.org/10.1109/TCSVT.2009.2020478>
- Lu Zhang, Ke Zhang, Tian Sheuan Chang, Gauthier Lafruit, Georgi Krasimirov Kuzmanov, and Diederik Verkest. 2011. Real-time High-definition Stereo Matching on FPGA. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '11)*. ACM, New York, NY, USA, 55–64. DOI: <http://dx.doi.org/10.1145/1950413.1950428>

Hardware and Software Innovations in Energy-Efficient System-Reliability Monitoring

Vasileios Tenentes*, Charles Leech*, Graeme M. Bragg*, Geoff Merrett*, Bashir M. Al-Hashimi*,
Hussam Amrouch[†], Jörg Henkel[†], Shidhartha Das[‡]

*ECS, University of Southampton, UK. Email: {V.Tenentes, C.Leech, G.Bragg, G.Merrett, bmah}@ecs.soton.ac.uk

[†]Karlsruhe Institute of Technology, Karlsruhe, Germany. Email: {amrouch; henkel}@kit.edu

[‡]ARM, Cambridge, UK. Email: sdas@arm.com

Abstract—Many threats that can undermine the reliability of a system can be realized at design, while others only during its online operation. As the availability of system monitoring sensors and run-time software increases in heterogeneous platforms, there is a demand for a novel platform-independent framework that can capture and deliver, in a holistic way, system level self-assessment and adaptation capabilities at run-time. In this paper, two groups from academia and one from industry present the following three contributions.

First, system reliability is considered from the perspective of novel timing guardband designs for aging mitigation. Effective timing guardband models are presented from the physical to the system level, while targeting multiple wear-out mechanisms.

Second, a technique for correlating complex software and micro-architectural events with power integrity loss is presented. The presented technique uses an embedded voltage noise sensor, a power-network model and a genetic algorithm for identifying workload that triggers power-network resonances which can ultimately lead to system failures.

Third, the ‘PRiME’ cross-layer programming framework is presented that unites available sensors and dynamic-voltage and frequency scaling actuators with learning-based run-time process mapping and scheduling algorithms. Scenarios on exploring the energy efficiency and reliability of heterogeneous platforms using run-time software derived from the developed framework are also reviewed.

I. INTRODUCTION

Energy efficient and reliable computing has become a major requirement of many applications nowadays. Mobile computing, IoT applications, smart cities and self-driving vehicles demand mechanisms for self-assessment and adaptation in order to perform reliably and energy-efficiently, but within very low cost constraints and platform volume restrictions [1]. At the same time, on-chip sensing and acting mechanisms are increasingly being embedded in an attempt to realize and mitigate reliability threats of energy-efficient system designs [1], [2], [3], [4]. They are based on monitoring in real time operational characteristics such as power consumption, power noise, temperature, hardware interrupts and performance monitoring events etc., and then processing the data for effectively exploring trade-offs in order to identify opportunities for energy efficient computing through acting e.g. tuning of Dynamic Voltage and Frequency Scaling (DVFS) policies.

Alternatively, they provide policies to avoid threats that can undermine system reliability [1], [2], [5]. As the complexity of sensors and software inevitably persists in heterogeneous systems, there is a demand for a novel platform-independent cross-layer framework that can capture and deliver, in a holistic way, system level self-assessment and adaptation capabilities for energy efficient computing using machine learning.

This paper discusses recent advances on hardware and software for system-reliability monitoring. In Section II, we present how timing guardbanding models for energy efficient yet reliable computing systems against multiple aging mechanisms can be derived at the design-time from the physical to the system level. In Section III, we present a technique for correlating complex software and micro-architectural events with power integrity loss of mobile-computing systems. In Section IV, we present the PRiME generic framework that unites the available sensors and the dynamic-voltage and frequency scaling actuators with learning-based run-time process mapping and scheduling algorithms. Multiple scenarios for exploring energy efficiency and reliability trade-offs of heterogeneous platforms that use the cross-layer programming framework are presented. Conclusions are drawn in Section V.

II. RELIABLE AND ENERGY-EFFICIENT GUARDBANDS FOR MITIGATING AGING

To ensure the correct functionality of circuits, *guardbands* (i.e. safety margins) need to be included in order to keep the deleterious impact of aging mechanisms, like Bias Temperature Instability (BTI) and Hot Carrier Injection (HCI), at bay. This holds even more when technology scaling approaches atomic levels in which wider and wider guardbands become indispensable. Hence, circuits’ designers need not only to *accurately estimate the required guardband* but to also answer the critical question of “*what is a reliable, yet energy-efficient guardband?*” At runtime, aging guardbands can be implemented by means of either timing or voltage as follows: (a) *Timing Guardbands*: To guarantee that a circuit will be always clocked with a sustainable frequency, a timing guardband, which is an additional time slack on top of the critical path delay, is typically added [6]. This ensures that no timing violations will take place during the lifetime despite

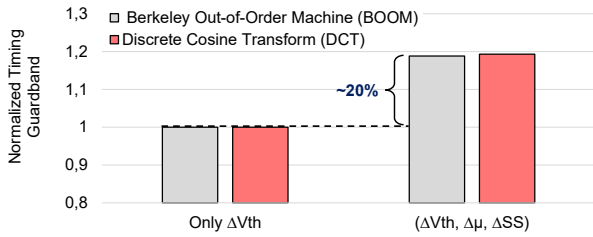


Fig. 1. The necessity of considering threshold voltage (V_{th}), carrier mobility (μ) and sub-threshold slope (SS) degradations in estimating aging guardbands.

degradation effects that will occur. However, including a timing guardband leads to efficiency loss since the design will be clocked with a lower frequency than its potential [7], [8].

(b) *Voltage Guardbands*: To avoid any performance loss caused by timing guardbands while sustaining reliability, designers alternatively may employ a guardband by means of voltage instead of timing [9]. Increasing the voltage with a specific guardband allows operating circuits with a faster speed in which any delay increase later due to aging effects will be always compensated. However, a voltage guardband comes with power/energy overheads and thus efficiency loss [10].

A. Considering multiple aging mechanisms

During the operation of transistors, aging mechanisms lead to generating different kinds of defects such as interface and oxide traps. Overtime, accumulated defects interact with the applied electric fields and hence they *manifest themselves as degradations* in the key electrical characteristics of MOSFETs. The primary observations are changes in threshold voltage (V_{th}), carrier mobility (μ) and sub-threshold slope (SS). To accurately estimate the required guardbands, different aging-induced degradations need to be *jointly* considered along with the existing interdependencies between them. In Fig. 1, we show how considering the impact of aging on V_{th} alone results in underestimating the required timing guardband by around 20% which, in turn, leads to unreliable operation due to including insufficient guardbands. The presented results for both BOOM processor [11] and DCT circuit have been estimated through creating “aging-aware cell libraries” under different degradation effects and then employing them within the Static Timing Analysis from Synopsys to accurately estimate the impact of aging on delay increase (details in [8]). Recently, a paradigm shift in aging from sole long-term reliability degradation, as in the traditional view, to short-term reliability degradation has been demonstrated in [12]. This is due to the temporal violation of guardband when aging effects are combined with the ultra-fast voltage switching. Additionally, instantaneous aging effects, due to the stochastic nature of defects in the deep nano scale, have been also recently reported and measured [7], [13]. In summary, accurate estimation of guardbands necessitates considering the long-/short-term effects of aging as well as the instantaneous effects of aging towards sustaining reliability.

B. Energy efficient guardbands

In order to increase the efficiency by minimizing guardbands, we proposed approaches at different design abstraction levels. Our developed aging models, aging-aware cell libraries, reliability framework, etc. are publicly available at [14].

1) *At the physical level*: As the causes are of physical origin, the fact that aging mechanisms may magnify or cancel each other cannot be neglected. Therefore, we modeled the joint impact of BTI and HCI demonstrating that what matters to answer “what is a reliable, yet energy-efficient guardband?” is the *interdependencies* between them [15]. We also demonstrated in [10] how considering the interdependencies between aging mechanisms and process variation (i.e. modeling all of them jointly instead of individually) enables designers to considerably minimize guardbands due to the higher certainty.

2) *At the circuit level*: We showed in [8] that aging-aware cell libraries and tool flows are indispensable to *efficiently contain* guardbands. Our investigation revealed that aging-induced degradations *unevenly* influence the gates’ delay within standard cell libraries. Moreover, the same gate can be *differently* influenced by the same aging degradation due to the important role that input signal slew and output load capacitance play. Therefore, considering aging degradations during logic synthesis results in smaller guardbands (around 50%) as our analysis for different kinds of processors showed [8].

3) *At the system level*: To address the question of “*do we really need to include a guardband even in error-tolerant circuits?*”, we studied the impact of aging-induced timing errors in image processing circuits demonstrating that aging leads to an unacceptable quality loss [8]. To completely remove guardbands and hence maximize efficiency, we explored in [16], for the first time, approximate computing principles in the context of aging. We showed that reducing the precision by merely 3 bits sustains circuits’ lifetime for 10 years under worst-case aging with unnoticeable drop in images’ quality.

III. GENETIC ALGORITHMS AND SENSORS FOR EXPLORING SYSTEM POWER INTEGRITY

Power delivery is a well-known challenge for high-end microprocessor systems. While there is a large body of work on Power Delivery Network (PDN) modeling for high-end enterprise server systems [17], [18], [19], similar research on mobile platforms is sparse. Mobile computing platforms have order-of-magnitude lower current consumption. This alleviates power delivery concerns, however, cost constraints and platform volume restrictions impose fundamental limitations on PDN impedance reduction. Furthermore, such systems are rapidly evolving into complex heterogeneous clusters that integrate application processors with graphics engines and specialized hardware accelerators. Current and future generation of such systems continue to rely upon supply voltage scaling to achieve energy-efficiency gains, thereby achieving higher performance under similar power budgets as previous generations. In addition, the trend towards GHz+ operating frequencies and the ubiquity of low-power techniques such as clock-gating and power-gating, make these systems susceptible to pathological

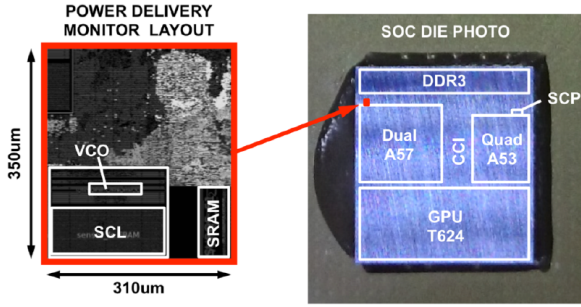


Fig. 2. On-Chip DSO Based Power Delivery Monitor samples on-die voltage on the A57 cluster: Support for waveform capture of upto 2K points [1] enables correlation of simulation analysis.

AC transients. Consequently, mobile computing systems are ultimately limited by power-delivery. However, this comes at a cost of both increasing current, and increasing current density, such that these systems effectively become constrained by power delivery.

A combination of higher PDN impedance, GHz+ operating frequencies and ubiquity of low-power techniques such as clock-gating, lead to infrequent combination of system and micro-architectural events that make these systems susceptible to pathological AC supply noise conditions. As such, sufficient voltage margin must be deployed to guarantee that such conditions do not lead to system failure. Ultimately, these voltage margins limit the energy-efficiency of battery-powered mobile platforms. Therefore, design-time optimization and post-silicon PDN characterization of such systems is crucial.

In [1], we presented the system-level PDN modeling and analysis results on a dual-core 64bit ARM Cortex-A57 platform in 28nm CMOS. We characterized the individual contributions of each constituent of the PDN, namely the Printed Circuit Board (PCB), the package and the die. We correlate our simulation analysis with measurement results based on a combination of off-chip instrumentation and on-chip circuitry. The analytical solution for the voltage droop seen at the die supply rails for such a simplified model of the PDN can be found in [1]. The voltage droop can be decomposed into a DC IR-drop term and an AC Ldi/dt term. The resistive component of the droop is addressed by increasing the metallization resources in the PDN. The inductive component is a complex trade-off between the package and the die and far exceeds the resistive droop magnitude in modern computing systems.

We conducted our analysis using a combination of on-chip and off-chip measurement. A high-bandwidth on-chip digital sampling oscilloscope (OC-DSO) snoops the supply rails of the A57 cluster [20]. The OC-DSO (Figure 2) runs continuously in real-time, logging data and capturing waveforms on trigger events. Event counter and time-mark registers track the size and frequency of voltage transients. For voltage transients of interest, threshold and gradient triggers can initiate waveform capture of up to 2K points into the internal SRAM trace buffer. A decimation block allows flexible bandwidth/sample rate to allow measurement of low frequency transients.

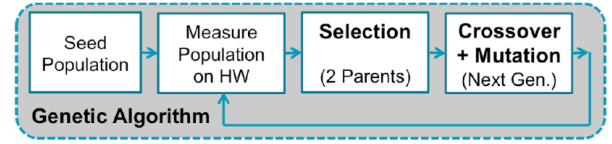


Fig. 3. The Genetic-Algorithm based framework for automatic generation of worst-case workloads.

Micro-architectural and system events often cause abrupt changes in current demand, leading to inductive transients that stress timing guardbands. Manually creating workloads that can trigger worst-case resonances in the system is difficult due to the complexity of the underlying micro-architecture, especially in out-of-order cores, such as the ARM A57. We circumvent this issue by automatically generating worst-case workloads using a genetic-algorithm based framework [21] that is agnostic to the processor micro-architecture (Figure 3).

Our results demonstrate how complex software and micro-architectural interactions can trigger PDN resonances that ultimately lead to system failure [2]. We draw several key conclusions regarding the PDN behavior in mobile systems:

- 1) *The inductive AC component of the voltage droop far exceeds (by an order-of-magnitude) the resistive DC component.*: This difference is especially acute in PDN for mobile systems, where the DC impedance is well managed through generous metallization resources. In contrast, the AC component, determined by complex interactions between the die, PCB and the package, is difficult to optimize during design-time.
- 2) *The PCB inductance assumes greater significance compared to the package*: It occurs due to constraints on the decoupling capacitors number that can be physically placed on-chip. Cost considerations limit the usage of package decaps causing the total inductance to be dominated by the PCB.
- 3) *PDN impedance exhibits a strong dependence on the power-gating state of the compute clusters*: Power gating a core reduces the total current drawn from the system, although at the expense of higher impedance magnitude at resonance. Voltage-droop mitigation techniques such as adaptive clocking [22], need to take into account the likelihood of larger voltage variation despite lower current consumption when individual components of the cluster are power-gated.

The above reasons make robust and reliable power delivery a first-class design challenge for future mobile systems. Timing guardbands budgeted for voltage transients ultimately limit the energy-efficiency of such systems. Voltage-droop mitigation techniques such as canary-based voltage tuning, error-resilient techniques [23], adaptive clocking [22] and integrated voltage regulation provide limited respite, either due to high calibration overheads or due to excessive cost and complexity. Indeed, further research is required to adequately address this challenge in future systems.

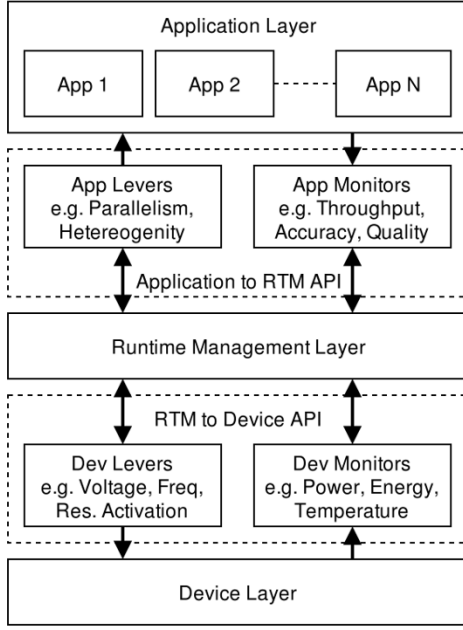


Fig. 4. PRiME framework.

IV. RUNTIME LEARNING FOR FAULT-TOLERANT AND ENERGY-EFFICIENT SYSTEMS

In this section, we present the PRiME generic framework (Figure 4), which is a cross-layer framework designed to be platform-independent and is targeting to capture and deliver, in a holistic way, system level self-adaptation capabilities for reliable and energy-efficient computing using machine learning. Next, we review scenarios on successfully exploring energy efficiency and reliability trade-offs of heterogeneous platforms using PRiME framework.

A. Power modelling using performance monitoring counters

We proposed a novel methodology for building accurate run-time power models using performance monitoring counters (PMCs) for embedded devices [24]. This methodology is based on making more efficient use of limited training data and better adapting to unseen scenarios by uniquely considering stability. We present a software implementation of it and build power models for ARM Cortex-A7 and Cortex-A15 CPUs, with 3.8% and 2.8% average error, respectively. Using this methodology as a solid foundation, we improved it by adding thermal-awareness and analytically decomposing the power into its constituting parts [25]. Here, we provide their model equations and software tools for implementing in a run-time manager or for using with an architectural simulator, such as gem5. On this purpose, the same methodology has been integrated into gem5 for estimating the power consumption of a simulated quad-core ARM Cortex-A15 [26].

B. Run-time energy management

We have explored run-time energy management by applying several principles depending upon the application domains. In [4], learning-based run-time power/energy management

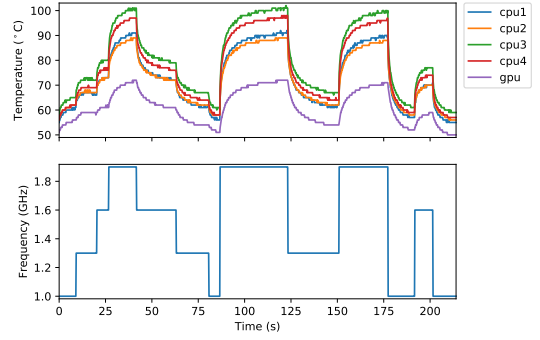


Fig. 5. Performance/temperature profiles of heterogeneous system during the dynamic transition of frequency scaling modes.

approaches for multi/many-core systems are surveyed. Specifically, we have looked model-based [27] and reinforcement [3] learning approaches. The model-based learning approaches perform offline analysis to derive the system behaviour for all the possible inputs and use the appropriate behaviour at run-time depending upon the input. In reinforcement learning, the system behaviour is learnt at run-time during execution and predictions are made based on the current system status. In addition, we have explored run-time management of concurrent multi-threaded applications on heterogeneous multi-cores [28]. The approach first selects thread-to-core mapping based on the performance requirements and resource availability. Then, it applies online adaptation by DVFS control to save energy, without trading-off application performance. To perform run-time energy management of heterogeneous multi-cores, we have devised techniques to efficiently share CPU and GPU resources [29].

C. Run-time aging mitigation

We showed that DVFS designs, together with stress-induced BTI variability, exhibit high temperature-induced BTI variability, depending on their workload and operating modes. In order to account for this variability in lifetime estimation, we proposed a simulation framework for the BTI degradation analysis of DVFS designs accounting for workload and actual temperature profiles [5]. Moreover, using ‘PRiME’ framework, we also explored the temperature variability of a heterogeneous ARM SoC (Fig. 5) at run-time and we used the models developed in [5] to identify the proper DVFS policies and the proper process mapping and scheduling that honour expected lifetime and energy efficiency constraints. We also showed that BTI comes with benefits for leakage current of logic [30] and memories [31], [32], and we proposed an accurate and energy-efficient BTI sensor design [33] that exploits the aging benefits for power-gating designs [34].

V. CONCLUSION

We presented novel timing guardband designs for aging mitigation from the physical to the system level, while targeting simultaneously multiple wear-out mechanisms. We also presented techniques for correlating complex software and micro-architectural events with power integrity loss, which is

a crucial issue for near threshold voltage computing systems. Moreover, the novel platform-independent ‘PRiME’ framework that can capture and deliver, in a holistic way, system level self-assessment and adaptation capabilities at run-time was presented, and we reviewed scenarios that use the developed framework for achieving energy efficiency and reliability trade-offs exploration for heterogeneous platforms.

ACKNOWLEDGMENTS

This work was supported in part by the German Research Foundation (DFG) as part of the priority program Dependable Embedded Systems (SPP 1500 - spp1500.itec.kit.edu) and EPSRC grant EP/K034448/1, the PRiME Programme, www.prime-project.org.

REFERENCES

- [1] S. Das, P. Whatmough, and D. Bull, “Modeling and characterization of the system-level power delivery network for a dual-core arm cortex-a57 cluster in 28nm cmos,” in *2015 IEEE/ACM International Symp. on Low Power Electronics and Design (ISLPED)*, July 2015, pp. 146–151.
- [2] P. N. Whatmough, S. Das, Z. Hadjilambrou, and D. M. Bull, “Power integrity analysis of a 28 nm dual-core arm cortex-a57 cluster using an all-digital power delivery monitor,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 6, pp. 1643–1654, June 2017.
- [3] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, “Learning transfer-based adaptive energy minimization in embedded systems,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 6, pp. 877–890, June 2016.
- [4] A. K. Singh, C. Leech, B. K. Reddy, B. M. Al-Hashimi, and G. V. Merrett, “Learning-based run-time power and energy management of multi/many-core systems,” *Current and Future Trends, Journal of Low Power Electronics (JOLPE) in Special Section on “New and Future Trends in Low Power Electronics”*, 2017.
- [5] H. Chahal, V. Tenentes, D. Rossi, and B. M. Al-Hashimi, “Bti aware thermal management for reliable dvfs designs,” in *2016 IEEE International Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Sept 2016, pp. 1–6.
- [6] S. Arasu, M. Nourani, J. M. Carulli, and V. K. Reddy, “Controlling aging in timing-critical paths,” *IEEE Design and Test Magazine*, vol. 33, pp. 82–91, 2016.
- [7] V. Santen, J. Martinez, H. Amrouch, M. Nafria, and J. Henkel, “Reliability in Super- and Near-Threshold Computing: A Unified Model of RTN, BTI and PV,” *IEEE Trans. on Circuits and Systems-I: Regular Paper (TCAS-I)*, 2017.
- [8] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, “Reliability-aware design to suppress aging,” in *53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.
- [9] T.-B. Chan, W.-T. J. Chan, and A. B. Kahng, “Impact of adaptive voltage scaling on aging-aware signoff,” in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2013, pp. 1683–1688.
- [10] H. Amrouch, V. M. van Santen, and J. Henkel, “Interdependencies of degradation effects and their impact on computing,” *IEEE Design and Test Magazine*, vol. 34, no. 3, pp. 59–67, 2017.
- [11] C. Celio, D. A. Patterson, and K. Asanovi, “The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor,” Berkeley, Tech. Rep., 2015.
- [12] V. M. van Santen, H. Amrouch, N. Parihar, S. Mahapatra, and J. Henkel, “Aging-aware voltage scaling,” in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2016, pp. 576–581.
- [13] V. M. van Santen, H. Amrouch, J. Martin-Martinez, M. Nafria, and J. Henkel, “Designing guardbands for instantaneous aging effects,” in *53rd ACM/EDAC/IEEE Des. Autom. Conf. (DAC)*, 2016, pp. 1–6.
- [14] “Our released models, tools and degradation-aware cell libraries,” <http://ces.itec.kit.edu/dependable-hardware.php>.
- [15] H. Amrouch, V. M. van Santen, T. Ebi, V. Wenzel, and J. Henkel, “Towards interdependencies of aging mechanisms,” in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2014, pp. 478–485.
- [16] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, “Towards aging-induced approximations,” in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, pp. 41:1–41:6.
- [17] R. Bertran, A. Buyuktosunoglu, P. Bose, T. J. Slegel, G. Salem, S. Carey, R. F. Rizzolo, and T. Strach, “Voltage noise in multi-core processors: Empirical characterization and optimization opportunities,” in *2014 47th Annual IEEE/ACM International Symp. on Microarchitecture*, Dec 2014, pp. 368–380.
- [18] K. Wilcox, R. Cole, H. R. F. III, K. Gillespie, A. Grenat, C. Henrion, R. Jotwani, S. Kosonocky, B. Munger, S. Naffziger, R. S. Orefice, S. Pant, D. A. Priore, R. Rachala, and J. White, “Steamroller module and adaptive clocking system in 28 nm cmos,” *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 24–34, Jan 2015.
- [19] A. Yeung, H. Partovi, Q. Harvard, L. Ravezzi, J. Ngai, R. Homer, M. Ashcraft, and G. Favor, “5.8 a 3ghz 64b arm v8 processor in 40nm bulk cmos technology,” in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 110–111.
- [20] P. N. Whatmough, S. Das, Z. Hadjilambrou, and D. M. Bull, “14.6 an all-digital power-delivery monitor for analysis of a 28nm dual-core arm cortex-a57 cluster,” in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, Feb 2015, pp. 1–3.
- [21] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. S. Govindan, “Audit: Stress testing the automatic way,” in *45th Annual IEEE/ACM MICRO*, Dec 2012, pp. 212–223.
- [22] K. A. Bowman, C. Tokunaga, T. Karnik, V. K. De, and J. W. Tschanz, “A 22nm dynamically adaptive clock distribution for voltage droop tolerance,” in *2012 Symp. on VLSI Circ. (VLSIC)*, June 2012, pp. 94–95.
- [23] S. Das, D. M. Bull, and P. N. Whatmough, “Error-resilient design techniques for reliable and dependable computing,” *IEEE Trans. on Device and Materials Reliability*, vol. 15, no. 1, pp. 24–34, March 2015.
- [24] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett, “Accurate and stable run-time power modeling for mobile and embedded cpus,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 106–119, Jan 2017.
- [25] M. J. Walker, S. Diestelhorst, A. Hansson, D. Balsamo, G. V. Merrett, and B. M. Al-Hashimi, “Thermally-aware composite run-time cpu power models,” in *26th Intern. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sept 2016, pp. 17–24.
- [26] B. K. Reddy, M. J. Walker, D. Balsamo, S. Diestelhorst, B. M. Al-Hashimi, and G. V. Merrett, “Empirical cpu power modelling and estimation in the gem5 simulator,” in *27th International Workshop PATMOS*, Sept 2017.
- [27] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi, “Adaptive energy minimization of embedded heterogeneous systems using regression-based learning,” in *25th Intern. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sept 2015, pp. 103–110.
- [28] K. R. Basireddy, A. Singh, G. V. Merrett, and B. M. Al-Hashimi, “Tmd: run-time management of concurrent multi-threaded applications on heterogeneous multi-cores,” in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, Mar 2017.
- [29] A. K. Singh, A. Prakash, K. R. Basireddy, G. V. Merrett, and B. M. Al-Hashimi, “Energy efficient run-time mapping and thread partitioning of concurrent opencl applications,” *CPU-GPU MPSoCs, ACM Trans. on Embedded Computing Systems (TECS)*, 2017.
- [30] D. Rossi, V. Tenentes, S. Yang, S. Khursheed, and B. M. Al-Hashimi, “Aging benefits in nanometer cmos designs,” *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 64, no. 3, pp. 324–328, March 2017.
- [31] D. Rossi, V. Tenentes, S. Khursheed, and B. M. Al-Hashimi, “Bti and leakage aware dynamic voltage scaling for reliable low power cache memories,” in *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*, July 2015, pp. 194–199.
- [32] D. Rossi, V. Tenentes, S. M. Reddy, B. M. Al-Hashimi, and H. A. Brown, “Exploiting aging benefits for the design of reliable drowsy cache memories,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.
- [33] V. Tenentes, D. Rossi, S. Yang, S. Khursheed, B. M. Al-Hashimi, and S. R. Gunn, “Coarse-grained online monitoring of bti aging by reusing power-gating infrastructure,” *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1397–1407, April 2017.
- [34] D. Rossi, V. Tenentes, S. Khursheed, and B. M. Al-Hashimi, “Nbti and leakage aware sleep transistor design for reliable and energy efficient power gating,” in *2015 20th IEEE European Test Symposium (ETS)*, May 2015, pp. 1–6.

Learning-based Run-time Power and Energy Management of Multi/Many-core Systems: Current and Future Trends

Amit Kumar Singh, Charles Leech, Basireddy Karunakar Reddy, Bashir M. Al-Hashimi, Geoff V. Merrett

Abstract– Multi/Many-core systems are prevalent in several application domains targeting different scales of computing such as embedded and cloud computing. These systems are able to fulfil the ever-increasing performance requirements by exploiting their parallel processing capabilities. However, effective power/energy management is required during system operations due to several reasons such as to increase the operational time of battery operated systems, reduce the energy cost of datacenters, and improve thermal efficiency and reliability. This article provides an extensive survey of learning-based run-time power/energy management approaches. The survey includes a taxonomy of the learning-based approaches. These approaches perform design-time and/or run-time power/energy management by employing some learning principles such as reinforcement learning. The survey also highlights the trends followed by the learning-based run-time power management approaches, their upcoming trends and open research challenges.

Keywords– Multi/many-core systems, power/energy optimization, run-time, machine learning

1 INTRODUCTION

Multi/many-core systems are becoming prevalent for various domains such as embedded and high performance computing (HPC). These systems provide increased parallelism by performing parallel execution of tasks on various cores [1]. This leads to high performance and thus helps to achieve increased performance requirements of modern computing systems. Chip manufacturers have developed several multi/many-core processors, e.g., Samsung's Exynos 8-core processor [2], Intel's Teraflop 80-core processor [3], AMD's Opteron 16-core processor [4], Tilera's TILE-Gx family 100-core processor [5], and Kalray's MPPA 256-core processor [6]. Depending upon the number of cores in the chip, they are connected by a shared bus or on-chip interconnection network [7–9]. These many-core processors are being exploited in various application domains to realize efficient many-core systems. It is also expected that higher number of cores will be integrated within a chip with technological advancements [10].

For these systems, usually, the applications need to be partitioned (parallelized) into multiple tasks that can be executed concurrently on different cores. Such partitioning is referred to as functional partitioning and can be furnished with the help of state-of-the-art application parallelization tools, e.g., MPSoC Application Programming Studio (MAPS) [11] and MNEMEE project tool-chain [12], and/or manual analysis. This procedure requires detailed application knowledge and involves finding the tasks, adding synchronization and inter-task communication in the tasks, management of the memory hierarchy communication and checking of the parallelized code (tasks) to ensure for correct functionality [13]. In case the multi/many-core system is heterogeneous, i.e. contains different types of cores, a task *binding* process that specifies the core types on them the task can be allocated along with the cost of allocation is required [14]. To compute the allocation cost, the binding process analyses the implementation costs (e.g., performance, power and resource utilization) of each task on different supported core types such as general purpose processor (GPP), digital signal processor (DSP) and coarse grain re-configurable hardware.

Power and energy efficient execution of applications on multi/many-core systems is desired in order to enhance operational time of battery-powered systems or energy cost of HPC datacenters. From several decades, enormous efforts have been put to optimize energy at circuit, architecture and system levels. The optimization of energy during application execution concerns to system level efforts. These efforts have tried to optimize energy by employing three essential ingredients: *mapping* [15–17], *dynamic voltage and frequency scaling (DVFS)* [18–21], and *dynamic power management (DPM)* [22–26]. The mapping defines assignment and ordering of the tasks and their communications onto resources of multi/many-core system in view of some optimization criteria such as compute performance and energy consumption. In DVFS, the voltage/frequency of the cores is adjusted dynamically to save energy consumption while meeting certain level of performance. The DPM process shuts down the cores when they are inactive. Several principles have been followed for shutting the cores down, for example, greedy approach where a core enters into sleep mode as soon as processing on the core is finished and timeout approach that enters the core into sleep mode after certain time of idleness if no request is received within that time. Out of mapping, DVFS and DPM, they have been applied individually and in combinations as well, e.g., mapping in [15, 16] and both mapping and DVFS in [27, 28].

While optimizing power and energy consumption during execution, the timing requirements of applications need to be satisfied. Different types of timing requirements are imposed depending upon the kind of target system, e.g., hard real-time and soft real-time systems [29]. Examples of hard real-time systems are time critical systems such as automotive engine and flight control software. In soft real-time systems such as video processing and HPC datacenters, the deadline violations can be tolerated. There has also been energy optimization efforts of mixed criticality systems, where part of the system has hard real-time requirement and rest has soft real-time requirement. Example of such a system is aeroplane, where cockpit system part has hard real-time requirement due to its safety critical issues and the parts in the passenger area such as lighting and television screen have soft real-time requirement.

Machine learning based power and energy optimization during multi/many-core system operation (i.e. at run-time) has gain significant attention over the last decade, although it exists since 1959. It provides learning ability to systems without being explicitly programmed. With respect to power and energy optimization, the

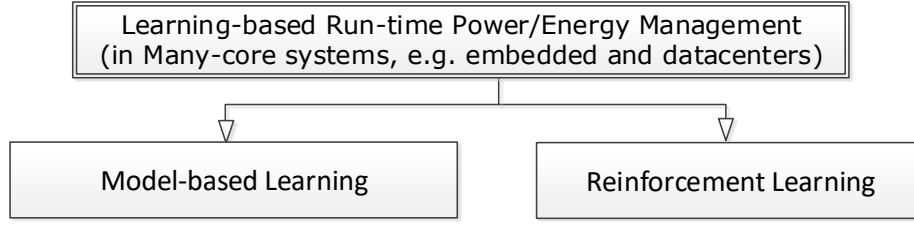


Figure 1: A taxonomy of learning based run-time power and energy management approaches.

learned information over time is used to predict appropriate mapping, DVFS or DPM policy to be applied for future execution. Such learning is helpful to make data driven predictions/decisions, where prediction models can be derived from sample inputs. Despite the fact that several articles have been published and significant progress has been made for machine learning based power and energy optimization of multi/many-core systems, there still remains many open questions and research challenges.

1.1 Learning-based Run-time Power and Energy Management Challenges

It is well known that the design space to map tasks on cores increases exponentially with the number of tasks and cores. Therefore, for large size problems, the learning based approach has the challenge to predict the most energy efficient *mapping* during application execution and adapt to the predicted mapping. Further, since modern cores possess DVFS capability, the prediction of appropriate voltage/frequency level is required during application execution in the view of optimizing energy consumption and satisfying timing requirement. However, finding the best energy efficient voltage/frequency level at run-time involves the challenge to perform accurate predictions within a short amount of time. Similar challenges exist to apply DPM. Additionally, while considering mapping and DVFS together, the prediction step needs to identify both the mappings and DVFS levels during execution, which adds further complexity as two aspects need to be considered jointly.

1.2 Classification of Learning-based Run-time Power and Energy Management Strategies

Learning-based run-time power and energy management strategies can be classified with a number of taxonomies, e.g., criticality (hard or soft real-time), optimization ingredient (mapping, DVFS, and DPM), employed learning principles, etc. Broadly, the classification can be done based on learning principle and other taxonomies can be included at some hierarchy in the learning principle based classification. For example, model-based (supervised/regression) learning can be used to find appropriate run-time mapping or DVFS level while trying to satisfy soft real-time requirements. Figure 1 shows classification of learning based power/energy management strategies based on the employed principle. The model-based learning approaches perform offline analysis to derive the system behaviour for all the possible inputs and use the appropriate behaviour at run-time depending upon the input. In reinforcement learning, the system behaviour is learnt at run-time during execution and predictions are made based on the current system status.

Paper Organization: Section 2 and Section 3 cover analysis and elaboration of model-based and reinforcement learning approaches, respectively. A comparative study of strategies falling into different categories has been performed in Section 4. Section 5 provides the upcoming trends that could be followed as the future research and open research challenges for learning based run-time power/energy management approaches. Finally, Section 6 provides some concluding remarks.

2 SUPERVISED MODEL-BASED LEARNING

The aim of model-based learning is to make predictions about future responses based on evidence in the presence of uncertainty. Supervised machine learning is a collective term for a group of algorithms that perform predictive modeling to establish a relationship between a set of predictor (independent) variables and a target (dependent) variable. In this section, each of the algorithms is introduced and discussed in the context

of learning-based run-time power or energy management strategies. Following this, the modeling approaches are grouped by the optimization and control methods that they employ, those of mapping, DVFS and DPM, which are introduced in section 1.

2.1 Model-based learning Approaches

Most of the supervised machine learning algorithms can be engineered to operate as either classification or regression techniques:

- Classification techniques predict discrete responses - for example, whether an email is genuine or spam, or whether a tumor is cancerous or benign. Classification models classify input data into categories. Typical applications include medical imaging, speech recognition, and credit scoring.
- Regression techniques predict continuous responses - for example, changes in temperature or fluctuations in power demand. Typical applications include electricity load forecasting and algorithmic trading.

2.1.1 Generalized Linear Models

The most common group of predictive algorithms used for learning-based run-time management are Generalized Linear Models (GLM). This term encompasses the majority of empirically and analytically derived models of performance or power commonly derived for prediction in management systems. Ordinary Least Squares (OLS) is an example of a GLM algorithm and is based on a linear combination of the predictor variables in the form of a hypothesis function and is defined as;

$$\hat{y}(\omega, x) = \omega_0 + \sum_{i=1}^n \omega_i x_i \quad (1)$$

The coefficients ω_i are established using a series of j training samples $(x_{i,j}, y_j)_{i=1 \dots n, j=1 \dots m}$ with i predictor variables. The OLS process minimizes the mean-squared prediction error $E(\omega)$ of the model, expressed as:

$$E(\omega) = \sum_{j=1}^m (f_{\omega}(x_j) - y_j)^2 \quad (2)$$

Future target values \hat{y} are predicted given new data x_{n+1} . Figure 2a illustrates how a OLS regression function is calculated from the training data points such that the mean-squared error is minimized.

GLM models can be built using many different algorithms besides OLS. Ridge regression addresses some of the problems of OLS by imposing a penalty on the size of coefficients and therefore the ridge coefficients minimize a penalized residual sum of squares. Further alternatives include Lasso, least angle and Bayesian regression, Orthogonal Matching Pursuit (OMP), the perceptron, passive aggressive algorithms and polynomial regression. Logistic regression is a GLM algorithm used for classification rather than regression. Also known in literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier, in this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

2.1.2 Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods that construct a hyperplane or set of hyperplanes in a high-dimensional space, which can be used for classification, regression or other tasks such as outlier detection.

In Support Vector Classification (SVC), the model is a representation of the examples points in space, mapped so that the separate categories are divided by a clear gap that is as wide as possible. A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (the

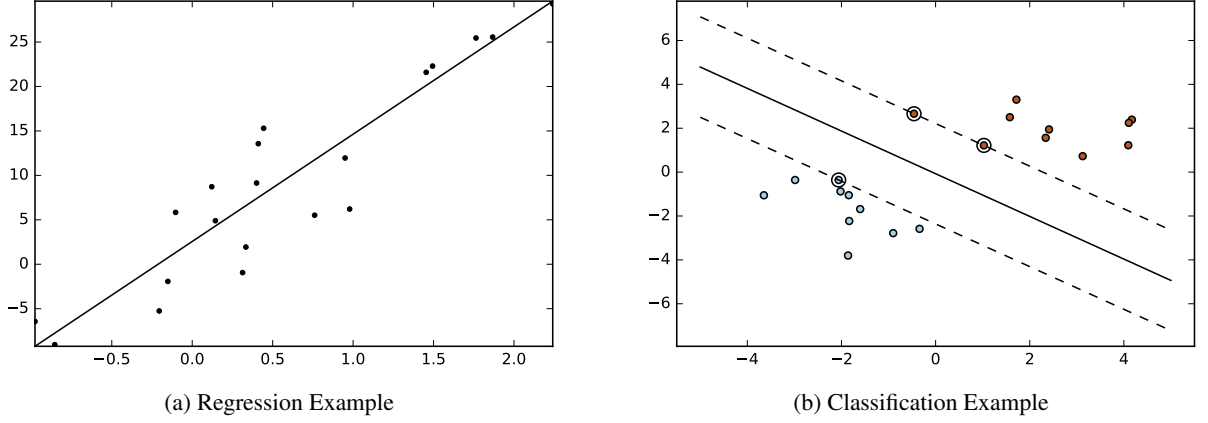


Figure 2: Graphical examples of model-based learning approaches for (a) linear regression and (b) support vector machine classification.

functional margin). The larger the margin the lower the generalization error of the classifier. New data points are mapped into that same space and predicted to belong to a category based on which side of the gap they fall. The model is trained using a dataset of n points in the form $(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)$ where y_i are either 1 or -1 , each indicating the class to which the point \vec{x}_i belongs. Each \vec{x}_i is a p -dimensional vector. The maximum-margin hyperplane that divides the group of points \vec{x}_i for which $y_i = 1$ from the group of points for which $y_i = -1$, is defined so that the distance between the hyperplane and the nearest point \vec{x}_i from either group is maximized. Any hyperplane can be written as the set of points \vec{x} satisfying $\vec{w} \cdot \vec{x} - b = 0$ where \vec{w} is the normal vector to the hyperplane. Figure 2b illustrates the output of the SVC process with the hyperplane (solid line) and the functional margins (dashed lines) shown. The training data points are shown with the bounding points highlighted.

Support Vector Regression (SVR) uses the same principles as the SVC, but in this case the intention is to develop a function and hyperplane that minimizes deviation of y_i for all training data [30]. As with classification, the algorithm takes input vectors X and y , but in this case y is expected to have floating point values instead of integer values. The model produced depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

SVMs have several advantages. They are effective in high dimensional spaces, even in cases where the number of dimensions is greater than the number of samples. They use a subset of training points in the decision function (called support vectors), so are memory efficient. They are versatile because different kernel functions can be specified for the decision function in order to classify data more appropriately. However, if the number of features is much greater than the number of samples, the method is likely to give poor performances. Also, SVMs do not directly provide probability estimates, these must be calculated using expensive cross-validation methods.

2.1.3 Naive Bayes

Naive Bayes is a simple technique for performing classification and can build models that assign class labels to instances of features where the class labels are identified from some finite set. Bayes methods are a set of supervised learning algorithms based on applying Bayes theorem with the “naive” assumption of independence between every pair of features. Given a class variable y and a dependent feature vector x_1 through x_n , Bayes’ theorem states the following relationship:

$$P(y | x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i | y)}{P(x_1, \dots, x_n)} \quad (3)$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i | y) \quad (4)$$

As a result, the classification task is essentially the assignment of the maximum a posteriori (MAP) class given the vector x_i and the prior of class assignments to y_i [31]. An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem.

2.1.4 Neural Networks

Neural networks (NNs) build a computational model based on a large collection of connected units called artificial neurons, analogous to axons in a biological brain. Connections between neurons carry an activation signal which can also be weighted to affect the strength of connections and the likelihood of activation. Neural networks must be trained from examples, rather than explicitly programmed, and excel in areas where the solution or feature detection is difficult to express in a traditional computer program. With sufficient training, NN can expose complex and hidden relationships that are difficult to characterize using rule-based programming. Typically, neurons are connected in layers, and signals travel from the input, to the output layer. Back propagation is the use of forward stimulation to modify connection weights, and is done to train the network using training data with known correct outputs. Increasing the number of hidden units in an NN leads to better representational power and the ability to model more complex functions, but increases the amount of training data and time required to arrive at accurate models.

If trained for too long, NNs can become overfitted and the model will include characteristics of outliers from the sample data, yielding an approximation with excellent accuracy on training examples, yet poor performance on further data from the same distribution. Overfitting can be prevented by reserving part of the data as a test set to such that unbiased estimate of the NN's accuracy. However, the model accuracy can be degraded as a result of holding aside training data for error estimation as it reduces the number of samples used for training which may be required. Cross validation is a mechanism to overcome this whereby the data set is divided into N equal-sized folds with N NN model is built instead of just a single. Each NN is trained on N1 folds and tested on the remaining fold, therefore the test fold for each NN differs from the other models [32].

2.2 Model-based learning for Run-time Management

In the context of run-time learning and management, modeling enables prediction of the current and future states of a system. This can include physical quantities, such as power, temperature and energy, or the specific properties of applications, such as performance, latency and accuracy. When applying specific requirements or constraints to these properties, models can be used to determine the system configuration that will minimize power consumption and maximize performance before execution, including control over parallelism, DVFS and DPM settings. The model-based learning approaches that have been used in literature are discussed, divided into three main control methods; task mapping (including parallelism/multi-threading control), DVFS and DPM. Categorization of existing literature is shown in Table 1. Many model-based approaches use multiple control methods in conjunction to achieve power/energy and performance optimization, with mapping applied first and then DVFS or DPM refinements made afterwards.

2.2.1 Task Mapping and Parallelism

Model-based learning is commonly employed to build power and performance models of applications executing on platforms to predict the optimal mapping of an application's tasks to processors and determine the level of parallelism that should be created.

Table 1: Classification of existing model-based run-time learning techniques for power and energy management

Domain	Reference	DPM	DVFS	Mapping
Embedded/Desktop	[32, 33]		✓	
	[34–36]			✓
	[31, 37]	✓	✓	
	[38]	✓		✓
	[39–42]		✓	✓
Datacenters/HPC	[39, 43]		✓	✓
	[44]	✓	✓	
	[45–47]	✓		✓

The first approaches to determine the required level of parallelism and task mapping using GLMs were empirically derived using Amdahl’s Law [48]. However, this fails to capture inter-thread communication, data dependence synchronizations and hardware contentions that lead to sub-optimal scaling. Modeling approaches have been developed to characterize these penalties with feedback-driven threading for homogeneous architectures [49] and Scale-Up/Scale-Out for heterogeneous architectures [38]. In the latter, *scale-out* refers to thread-level parallelism [50] and *scale-up* to the adaptive thread-core mapping enabled by heterogeneous cores. These processes are characterized by two orthogonal functions, which are derived from a combination of empirical modeling and fitting of additional coefficients via linear regression [38]. Similarly, composite models can be constructed, with a combination of an empirically-derived component and a GLM component. These are designed to characterize the system whilst also mitigate the modeling error that arises from unknown factors. In [34], a holistic and resource-agnostic scalability model is developed in order to determine the degree of parallelism to assign to each task. The model is based on predicting speed-up from Amdahl’s Law, with consideration given to the aforementioned parallelism penalties, however in addition it employs linear regression to analyze the speed-up properties of particular task in order to assign the correct level of parallelism.

Coarse-grain mapping of applications to computational resources can be driven by regression-based learning [42]. The approach uses OLS approaches to build a model of the energy/performance trade-offs between using different computing resources in a heterogeneous system for a particular task. The task is mapped on a computing resource at run-time based on the minimum energy consumption for a given application performance requirement. Parallelism within each resource is not considered because of the particular platform.

On the other hand, approaches that target HPC and datacenter systems consider task-level parallelism an essential component of their predictive models. These approaches considered modeling the system in order to perform Dynamic Concurrency Throttling (DCT) [39] and thread packing [40], processes which we include as part of mapping. Curtis-Maury et al. [39] consider an IPC-based linear regression solution trained from samples of the power-performance adaptation search space collected from real workloads. They derive a performance prediction model which dynamically adjusts DCT, DVFS, and thread placement at the granularity of program phases.

In order to more accurately and generically predict performance improvements for changes in mapping, GLMs can leverage performance monitoring counters (PMCs) which are built into the hardware architecture [32,35]. This approach is portable across many applications as it only relies on information from the hardware. Furthermore, metrics such as instructions-per-cycle (IPC) and processor utilization can be used to predict performance and build linear models across many platforms [39]. Pack & Cap is an example of a model-based approach to control mapping which relies on PMC data [40]. Furthermore, it is different to other approaches in that it employs a multinomial logistic regression (MLR) classifier to make optimal DVFS and thread packing control decisions in order to maximize performance within a power budget. The addition of thread packing to DVFS as a control knob increases the range of feasible power constraints by an average of

21% when compared to DVFS alone and reduces workload energy consumption by an average of 51.6%.

SVC models can also be found in run-time management scheme and used to classify tasks or programs as suitable for particular functional units in a heterogeneous architecture. Wen et al. [36] develop an OpenCL task scheduling scheme to map kernels from multiple programs on CPU/GPU heterogeneous platforms. At run-time, it determines which kernels are likely to best utilize a device from a performance model that predicts a kernel's speedup based on its static code structure. Naive Bayes has also been used in power management to build power-performance model and perform classification [31]. In the context of this work, the goal is to devise a power management policy for issuing DVFS commands on a CMP system that minimize the total energy dissipation based on the load conditions and workload characteristics [37]. The motivation for utilizing a Bayesian classifier is to reduce the overhead of the power management activities which are performed regularly to determine and assign DVFS settings for each processor core in the system.

The use of Artificial Neural Networks (ANN) for modeling and prediction in run-time management system is not common, given the extensive training time and large volume of data that is required to achieve an accurate model, as discussed in 2.1.4. However, ANNs can have a role to play in the static components of a hierarchical system such as modeling the behavior of applications as in [32]. A resource allocation framework is created composed of per-application ANN performance models and a global resource manager. Shared system resources are periodically redistributed between applications at fixed decision-making intervals. Each application model's its performance as a function of its allocated resources and recent behavior, using an ensemble of ANNs to learn an approximation of this function. Past program behavior and allocated resource amounts are presented at the input units, and performance predictions are obtained from the output units [32]. The drawback of these model is the training of the NN weights which can only be done by performing successive passes over training examples.

2.2.2 DVFS

DVFS is used to control the performance/throughput of tasks by adjusting the operating frequency of the processor. Dynamic power dissipation is reduced as a result and energy can be saved if further voltage scaling occurs. The power and performance relationship is often modeled to enable prediction of the optimal DVFS settings. A basic model can be built from understanding of the underlying physical characteristics of the static and dynamic power dissipation of components and how these are affected by frequency and voltage or empirically from experimentation using training samples. The later may be done with the aid of hardware performance statistics such as IPC [39] or PMCs [40, 41]. The former uses multivariate linear regression to estimate specific coefficients for the hardware event rates of a particular configuration in order to determine the required DCT and DVFS settings. The *Pack & Cap* [40] approach makes use of L_1 -regularization to select the most relevant PMC metrics automatically and a multinomial logistic regression (MLR) classifier to determine the DCT and DVFS settings that maximize performance under a power constraint by selecting the output with the highest probability.

Yang et al. [42] use hypotheses about the affect of frequency and voltage on the current and latency for each resource in a heterogeneous platform as the basis for their power/performance model which is used to determine the most energy efficient resource to execute on and the DVFS settings to apply given a performance requirement. This model is trained using a OLS linear regression technique at the beginning of application execution. Although it can be done a run-time, it does not change over the remainder of the application so it cannot adapt to changes in application behavior. In a similar way, Juan et al. [33] use constrained-polynomial (positive polynomial) functions to learn the relationship between performance and power and build a model based on frequency and utilization. Additional energy reduction is achieved through additional DPM techniques including turbo-mode and near-threshold operation in what they call extended-range DVFS.

A Bayesian classification approach to DVFS setting is used by Jung et al. [31, 37] in the prediction of power and performance. The predicted state is used to look up the optimal power management action from a pre-computed policy lookup table. The motivation for using this form of model is the reduced overhead of prediction in the *power manager (PM)* and as a result can provide energy savings for even rapidly and widely varying workloads.

2.2.3 DPM

DPM process are often driven by predictions from models in conjunction with mapping or DVFS actions. DPM can be achieved through migration in heterogeneous to resources with different power/performance operating points [38, 42] or by power gating CPU cores in homogeneous multi/many-core systems [32, 40]. These two processes are captured as Scale-Up and Scale-Out by Ma et al. [38] who perform DPM and mapping on a heterogeneous architecture based on prediction from GLM performance and power models with additional heuristic scheduling. Cochran et al. [40] propose a similar approach for performance optimization under a power budget with PARSEC benchmarks on a homogeneous CMP through a combination of thread packing to control parallelism and DVFS setting to reduce power.

2.3 Model-based Management in Datacenters

Model-based approaches, including supervised machine learning, have been used to increase energy efficiency in server [51] and datacenter [43] platforms. Mapping, DVFS and DPM techniques have all been employed such as workload consolidation, which aims to reduce the number of active processing resources and as result reduce the power consumption and heat dissipation [45]. Modeling the effect of these management processes is even more important as the hardware in datacenters becomes increasingly heterogeneous [43]. In this situation, power and performance must be modeled for all the settings of each resource as well as the partitioning of workloads across multiple resources. Wu et al. use linear regression to build a model of a heterogeneous CPU and FPGA platform, with support for workload partitioning [43], from both compile-time and run-time profiling, and use it to for run-time average power estimation. Lama et al. [46] apply machine learning to build fuzzy power and performance models to capture non-linear system behavior and drive a model predictive control framework. This self-adaptive modeling allows them to capture time-varying relationships between application performance and allocation of resources for dynamic and bursty workloads. Distributed controllers coordinate with each other to allocate resources and meet the service level agreements of applications.

3 REINFORCEMENT LEARNING

3.1 Introduction to Reinforcement Learning

Reinforcement learning (RL) is a machine intelligence approach that has been applied in many different areas. It mimics one of the most common learning styles in natural life. The machine learns to achieve a goal by trial-and-error interaction with a dynamic environment. RL algorithms are developed to find the optimal solution to sequential decision problem, and have been proven effective in a variety of problems from different areas [21]. RL is inspired by the trial-and-error method humans used for making decisions for millions of years. In RL, the agent interacts with the system (Fig. 3).

The general learning model consists of:

- An agent
- A finite state space S
- A set of available actions A for the agent
- A reward function $R: S \times A \rightarrow R$

The goal of RL is to find the best actions under different states such that by following those best actions, the agent can optimize the long-term reward. It is achieved by learning a policy, i.e. a mapping between the states and the actions.

Q-learning is one of the most popular algorithms that perform reinforcement learning. At each step of interaction with the environment, the agent observes the environment and issues an action based on the system state. By performing the action, the system moves from one state to another. The new state gives the agent

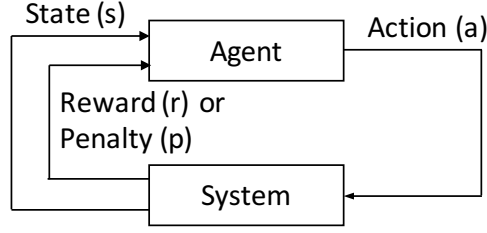


Figure 3: Agent-system interaction in RL. The systems is characterized by state s . The action α is taken by an agent to change the state, with resulting outcomes r or p and new state s'

a reward (a real or natural number) or punishment (a negative reward) which indicates the value of the state transition. The agent keeps a value function for each state-action pair, which represents the expected long-term reward if the system starts from state s , taking action a , and thereafter following a policy. Based on this value function, the agent decides which action should be taken in current state to achieve the maximum long-term rewards. The core of the Q-learning algorithm is a value iteration update of the value function. The Q-value for each state-action pair is initially chosen by the designer and later, it is updated each time an action is issued and a reward is received, based on the following expression.

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha (r_i + \gamma * \max_{a'} Q(s', a') - Q(s_i, a_i)) \quad (5)$$

In the above expression, r_i is the reward given at time i , and $0 \leq \alpha \leq 1$ is the learning rates which may be the same value for all pairs. The discount factor γ is a value between 0 and 1, and s' and a' are the next state and action after taking a_i , respectively. The next time when state s is visited again, the action with the maximum Q-value will be chosen. As a model-free learning algorithm, it is not necessary for the Q-learning agent to have any prior information about the system, such as the transition probability from one state to another. Thus, it is a highly adaptive and flexible algorithm.

Q-learning is originally designed to find the policy for a Markov Decision Process (MDP). It is proved that the Q-learning is able to find the optimal policy when the learning rate is reduced to 0 at an appropriate rate, given the condition that the environment is MDP. However, it is important to point out that a computing system for power/energy management is typically non-Markovian. Therefore, it is not guaranteed that Q-learning will find the optimal policy in case of such problems.

3.2 Reinforcement Learning for Run-time Management

System level power/energy management must consider the uncertainty and variability that comes from the environment, application and hardware. Statically optimized resource and power management are not likely to achieve the best performance when the input characteristics are changing. As a result reinforcement learning has been used for DPM [22–26], DVFS [18–21, 52], or combination of DPM, DVFS and mapping [28, 53, 54] in embedded, desktop and datacenter domains. A detailed classification of existing RL based approaches for power/energy management is given Table 2.

3.3 Mapping with DPM or DVFS

Ye et al. [54] claims that existing learning based DPM is not dedicated to power reduction in multi-core processors. Further, they address this issue by including task allocation into their Q-learning based DPM framework. As the multi-core processors offer flexibility in assigning tasks to any processor core, it helps in partially controlling idle periods for power savings. This technique judiciously allocates the tasks to cores which offer a better trade-off between power consumption and system performance. To further achieve better power savings, core temperatures are integrated into DPM framework as the temperature has profound impact on leakage power consumption. The simulation-based approach is used for evaluating the effectiveness of their approach.

Table 2: Classification of existing reinforcement learning based techniques for power/energy management

Domain	Ref.	DPM	DVFS	Mapping
Embedded/Desktop	[22–26]	✓		
	[18–21, 52]		✓	
	[53]	✓	✓	
	[54]	✓		✓
	[28]		✓	✓
Datacenters	[55–58]			✓
	[59, 60]	✓		✓

To improve the energy efficiency and thermal aspects (average and peak temperature, and thermal cycling) in multi-core systems, a Q-learning based approach for DVFS and POSIX thread allocation is proposed in [28]. It simultaneously optimizes the temperature and energy consumption with reduced learning space by employing a two-stage hierarchical approach: a heuristic-based thread allocation at a longer time interval to improve thermal cycling, followed by a learning-based DVFS at a much finer interval to improve average temperature, peak temperature and energy consumption. This approach is evaluated on nVidia's Tegra SoC, featuring four ARM Cortex-A15 cores, running Linux with set of applications from MiBench [61], PARSEC [62] and SPLASH-2 [63].

3.4 DVFS

Reinforcement learning based approaches have been widely used for energy minimization through DVFS [18–21, 52]. Shen et al. [18] proposed an approach for simultaneous temperature, performance and energy (TPE) management. It dynamically selects the processor's voltage and frequency to achieve the required balance among energy, performance and temperature of the processor. This approach models the processor's energy as convex function, which first decreases and then increases with the frequency. Further, performance and temperature are modelled as concave and convex functions increasing with the normalized frequency. The relation among temperature, performance and energy based on the above observations determines the possible trade-off space of the TPE management. Their environment state is a vector of four components, (f, T, IPS, μ) representing the clock frequency, the temperature, the instructions per second (IPS) and the CPU intensiveness, respectively. The TPE controller offers two modes: *free* and *constrained*. *Free* mode allows user to explore the trade-off by tuning the weight coefficients in the penalty function and *constrained* mode lets the user to set constraints to two out of the three parameters in T, P and E, while optimizing the third one. The approach is validated on Dell Precision T3400 workstation with Intel Core 2 Duo E8400 Processor running Linux.

Juan et al. [19] present a semi-supervised reinforcement learning (RL) based approach for performing dynamic voltage and frequency scaling (DVFS) to efficiently utilize the available on-chip power budget while maximizing the performance. Further, an evaluation and comparison among core-only, uncore-only and cooperative core/uncore DVFS control for performance boosting is discussed for the first time along with a “reverse” DVFS technique for maximizing performance under power constraints. The semi-supervised RL separates the centralized agent into several “distributed” agents, and arrange a supervisor to coordinate the actions among agents to best exploit the power budget for the performance increase. This helps in reducing the exponential growth of state complexity and maintains a linear complexity with the number of cores. The targeted architecture is a symmetric, NoC-based CMP containing 16 tiles, and each tile has a Pentium4 core, a private L1 cache, a shared L2 cache and an on-chip router. The evaluation with a wide spectrum of parallel, multi-threaded applications shows an average 10.9% improvement in program execution time while satisfying given power constraints.

The emerging multi-task/thread applications generally have complicated execution causality and task-level dependencies, and the execution states of one core would affect other cores in a multi-core system. To

globally optimize the policy of voltage/frequency selection for improving energy efficiency in such scenarios, a core-level modular RL (MLR) based online DVFS, which explicitly considers the relationship between different cores, is proposed in [21]. This approach distributively applies modular Q-learning (MQL), one of the most commonly used MRL algorithm, to each core to learn the system behaviour. MQL decomposes the state-space into several much smaller modules and integrates multiple modules in an agent (DVFS controller of each core) to select actions based on more than its own state. At each learning-epoch, DVFS controller collects necessary information required by every module from the corresponding cores in the system and passes it into associated modules to learn and update their Q-table. As each module has its own Q-table and updates independently, a mediator is used in an agent to arbitrate the solutions reported by each module. Furthermore, task-dependency information of applications, collected from static-time workload characterization or dynamic profiling, is used to help efficiently create and associate most useful modules to each core which incurs only linear cost in core count. Experiments conducted on a homogeneous system with 4, 16, 32 and 64-cores (implemented in JADE full-system simulator [64]) running realistic applications from COSMIC benchmark suite [65], show up to 28% energy savings compared to individual-learning method.

Considering the future many-core systems, an On-line Distributed RL (OD-RL) based DVFS control algorithm to improve performance under power constraints is discussed in [52]. At the finer grain, a per-core RL method is used to learn the optimal control policy of the voltage/frequency levels that maximizes the performance under the budget. At the coarser grain, an efficient global power budget reallocation algorithm is used to maximize the overall performance. Spatially, distributed RL works on each core locally and independently, while the budget re-allocator reallocates the budget among all the cores. Temporally, distributed RL operates at every control epoch, while the budget re-allocator executes every M epochs. This approach is validated using Sniper simulator with PARSEC and SPLASH-2 multi-threaded benchmark suites. Experimental results show up to 98% less budget overshoot, up to 44.3x better throughput per over-the-budget energy, up to 23% higher energy efficiency and two orders of magnitude speedup over existing techniques [66, 67].

3.5 DPM

Learning-based DPM techniques have been proved to be effective in reducing power consumption [22–26]. An on-line learning algorithm in [25] dynamically selects the best DPM policies from a set of candidate policies called experts. Each expert has a weight factor, the value of which indicates the benefit gained if the correspondent expert was chosen during the last idle period. The one with the highest value will control the device for the next idle period. Prabha et al. [26] propose a similar approach using a different learning algorithm. The expert-based machine learning algorithm is able to find an appropriate DPM policy in short time without any prior workload information. However, it cannot explore the power-performance trade-offs effectively.

The traditional Q-learning algorithm provides a model-free solution for the MDP with a provable convergence to the optimal solution. However, in a non-markovian environment or a partially observable Markovian environment [68, 69], Q-learning is capable of achieving the same performance as other reference learning algorithms at the cost of slower convergence (number of epochs for the Q-values becoming stable). To address this issue, Liu et al. [22] propose a system level power management based on enhanced Q-learning to improve the convergence speed. It adopts the method in [70] and enhances the performance of traditional Q-learning by exploiting the sub-modularity and monotonic structure in the cost function of a power management system. The enhancement restricts the search space of Q-learning algorithm to policies whose action is non-decreasing to the number of waiting requests, which helped in improving power consumption and latency by 40% and 90%, respectively, compared to traditional Q-learning. Further, it can adapt to changing performance constraint during run-time and converge to a policy that delivers just enough performance with minimum power consumption within 50 updates. Their experimental results show up to 30% and 60% reduction in power consumption for synthetic and real workloads, respectively, when compared against existing expert-based power management technique [71].

The reinforcement learning technique is applied to multi-cores in Ye and Xu [2012] and shown superior than the distributed power managers using Tan et al. [2009]. All the core states and actions are encoded and the learning function is approximated using the back-propagation neural network (BPNN) [54]. Tasks are

assumed independent, thus their policy is able to not only determine the power mode, but also assign each task to a specific core. However, task assignment (context scheduling) has other considerations, such as data locality, that greatly affect the performance in real environments. Besides, scalability is still a problem for its time complexity $O(n^2)$.

A policy called Multi-level Reinforcement Learning (MLRL) that is much more scalable in efficiency and effectiveness for multi-cores is proposed in [23]. The multilevel paradigm, having *coarsening* and *uncoarsening* phases, is first proposed in Karypis et al. [1999] for circuit partitioning and then applied to other VLSI problems. The coarsening phase does recursive clustering of elements until the problem size is small enough to be solved. Then, in the *uncoarsening* phase, the coarsened elements are de-clustered by applying the refinement algorithm. A coarse solution is produced between the two phases. Coarsening phase generates a good approximation of the original problems, so better initial solutions can be obtained and makes refinement algorithms more effective due to local problems with smaller sizes in the uncoarsening phase.

The multilevel paradigm is exploited to compress the searching space, speed-up the convergence rate, and result in $O(n \log n)$ time complexity for n cores. Target architecture is a multiprocessor system, ARM Cortex-A9 MPCore, containing n homogeneous cores and m threads per core, which can simultaneously execute $n \times m$ contexts. The workload characteristics are assumed unknown in advance; it may comprise several single- or multi-threaded programs. Simulation results, using Multi2Sim [72] and the power simulator McPAT [73], show that their policy runs 53% faster and outperforms the state-of-the-art work [54] with 13.6% energy savings and 2.7% latency penalty on average for the SPLASH-2 benchmarks while the performance penalty is close to zero.

The above approaches, including the enhanced Q-learning which is a model-free RL requiring no knowledge of state transition probability functions, needs knowledge of state action spaces and reward function. The enhanced Q-learning based DPM learns a policy online by identifying which action is the best for a certain system state, based on the reward or penalty (cost) received. In this way, the approach does not depend on any pre-designed experts, and can achieve a much wider range of power-latency trade-offs. However, as this is based on a discrete-time model of the stochastic process, it suffers from the following limitations: (i) the discrete-time controller has relatively high overhead to make frequent and regular decisions, and (ii) discrete-time controller may not make timely decisions for fast state changes.

As a solution to the above problems, Wang et al. [24] presented an online adaptive DPM technique based on the model-free RL method, which requires no prior knowledge of the state transition probability function and the reward function. Furthermore, this approach can perform policy learning and power management in a continuous-time and event-driven manner to learn a desirable timeout policy (optimal DPM policy when the service requests inter-arrival times are stationary but non-exponentially distributed [74], which derives optimal timeout value using MDP methods). It also utilizes the enhanced temporal difference ($TD(\lambda)$) learning algorithm for semi-MDP (SMDP) [75] in order to accelerate convergence and alleviate the reliance on the Markovian property. $TD(\lambda)$ is more robust in non-Markovian cases as it seamlessly combines the simple one-step TD algorithm and the Monte Carlo method and has fast learning rate. Workload prediction is incorporated in this work to provide partial information about service requester (SR) state for the RL algorithm. Specifically, an online Bayesian classifier [28] is chosen as the workload predictor because of its relatively high prediction accuracy, low implementation cost, and the fact that the information it provides comes with a certain degree of certainty due to the use of posterior probability [28]. Further, it uses only two actions “keep sleep” and “wake-up”, instead of time-out policy to reduce state-action pairs of RL and improves the convergence speed using multiple-update initialization, dynamic action sets, and locally randomized action selection techniques. The experiments are performed considering two different devices: a hard disk drive (HDD) and a wireless adapter card (WLAN card) on both synthesized and real workloads traces. Without sacrificing any latency, it achieves a maximum 18.6% saving in power dissipation compared to the reference expert-based approach proposed in [71]. Alternatively, the maximum latency saving without any power consumption increase is 73% compared to the existing best-of-bread DPM techniques. This method works only for single-core systems having single device without DVFS capability.

3.6 DVFS with DPM

As discussed above, DPM and DVFS has proven to be effective techniques for reducing power/energy consumption. Both DPM and DVFS provide a set of control knobs for run-time power management. From this perspective, both are fundamentally same. While the DVFS is usually found as the power control knob for CMOS digital ICs, such as micro-controllers or microprocessors, during the active time; the DPM is usually for the peripheral devices, such as the hard disk drives and network interface, or for microprocessors running interactive applications accompanied with long idle intervals. Shen et al. [53] considers both DPM and DVFS as power management and propose a novel approach for system level power management using enhanced Q-learning. This technique considers the peripheral device (an interactive system that processes the I/O requests generated by software applications) and the microprocessor. Experiments for power management of peripheral devices are done using a simulated HDD with synthetic and real workloads. Furthermore, microprocessor power management approach, Q-learning based DVFS controller, is evaluated on a Dell Precision T3400 workstation with Intel Core 2 Duo E8400 Processor with applications from MiBench [61] and MediaBench [76]. The result showed that, compared to existing machine learning approaches, their power management technique is more flexible in adapting to different workload and hardware, and provides a wider range of power-performance trade-off.

3.7 Reinforcement Learning in Datacenters

Reinforcement learning based power management techniques have been proved to be effective to reduce energy costs in datacenters [56–60]. These learning based approaches are used for resource/workload consolidation, which involves turning off under-utilized/sleep servers to save power, and resource allocation. Fundamentally, resource/workload consolidation can be seen as DPM with mapping.

A localized version of online RL for resource allocation is presented in [56–58]. The RL module observes the application's local state, the local number of servers allocated by the arbiter, and the reward specified by the local Service Level Agreement (SLA). Considering the scalability of look-up table based Q-learning, severe approximations have been made by representing the application state solely by current mean arrival rate of page requests, and ignoring other sensor readings (e.g. mean response times, queue lengths, number of customers, etc.). Further, to improve the initial performance of the arbiter's policy, a heuristic initialization is employed. However, these approaches work well for simple systems running simple applications with less state variables and needs certain amount of exploration of actions believed to be suboptimal. To address this issue, Tesauro et al. [58] proposed a hybrid method combining the advantages of both explicit model-based methods and model-free RL. This approach involves offline training of RL module, instead of online training, on data collected while an externally supplied initial policy (e.g., based on an approximate queuing model) makes management decisions in the system.

Lin et al. [59] discussed a power management technique which does both the job dispatching and resource consolidation. The job dispatch is performed at a finer interval than the resource consolidation based on the job arrival time and the server resource availability, while the resource consolidation decision on a fixed-length time slot basis. The proposed approach was verified by simulations using real Google cluster data traces. Further, Farahnakian et al. [60] proposed a Q-learning based dynamic Virtual Machine (VM) consolidation method to optimize the number of active hosts according to the current resources utilization. The method intelligently decides on when to switch a host into the active or sleep power mode through a learning agent which learns host power mode detection policy.

4 COMPARATIVE STUDY

This section presents a comparative study of various machine learning based approaches

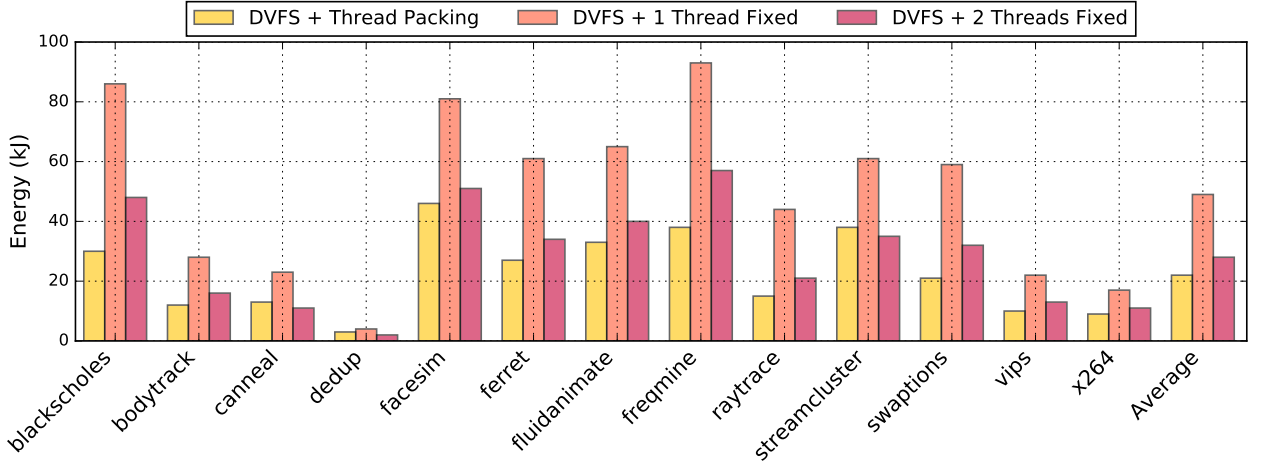


Figure 4: Comparison of energy consumption for benchmarks with and without thread-packing on a homogeneous CMP, reprinted from [40].

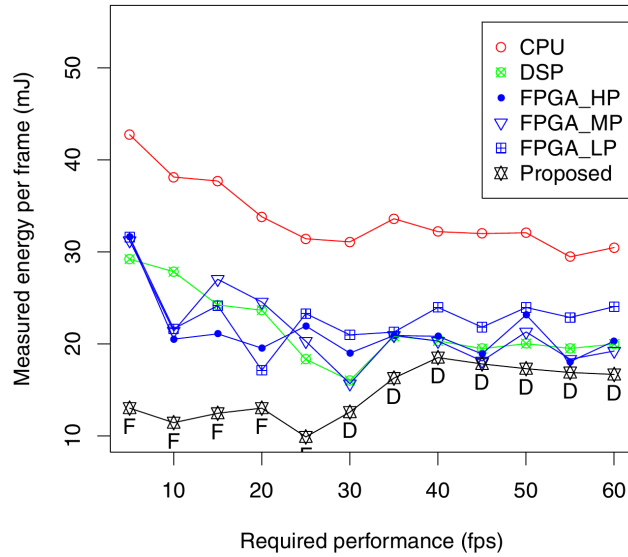


Figure 5: Energy consumption of an edge detection filter executing on the different processing elements of a heterogeneous multiprocessor over a range of performance constraints [42].

4.1 Comparison of Model-based learning Approaches

Figure 4 shows that energy savings can be achieved by combining DVFS and adaptive thread mapping on a homogeneous CMP with the aid of an MLR classifier [40]. The experiments are conducted across a series of PARSEC benchmarks. In the first experiment (DVFS + Thread Packing), the thread packing technique is used to dynamically adjust the number of threads to meet a changing power budget, with DVFS settings applied on top. The second (DVFS + 1 Thread Fixed) and third (DVFS + 2 Threads Fixed) experiments use a fixed one and two threads respectively and so must rely on predicted DVFS settings alone. For the majority of cases, thread packing increases the range of achievable power constraints over DVFS alone. Given that DPM techniques to shutdown cores has not been applied, the one thread fixed case consumes a lot more energy due to the static power consumption of the idle cores. The thread packing experiment is able to utilize all four cores when the power budget allows and the performance gain from this outweighs the increase in dynamic power.

In addition, Figure 5 shows that learning the power/performance trade-offs for each processing resource of a heterogeneous system enables prediction of the optimal mapping and DVFS settings. A linear regression

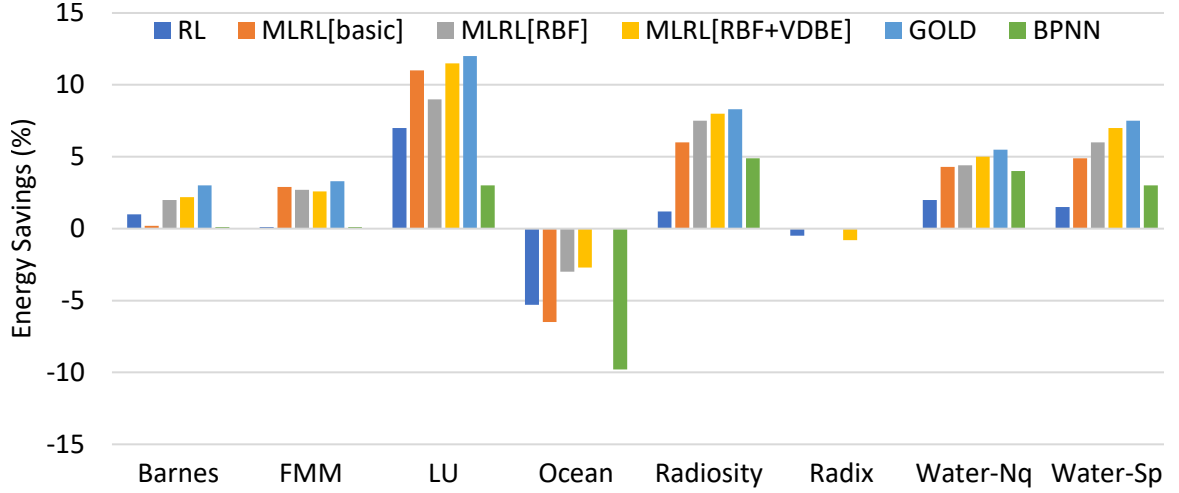


Figure 6: Comparison of energy savings achieved by various DPM techniques for SPLASH benchmark applications. Extracted from [23].

model is built from training data and used to predict the power and performance of an edge detection filter for particular mapping and DVFS settings on the heterogeneous CPU/DSP/FPGA platform [42]. The run-time management system implemented will switch the mapping of the filter between resources if the performance requirement changes to ensure optimal energy consumption per frame. The combination of mapping and adaptive DVFS setting (black line) produces a lower energy consumption for the entire range of performance levels when compared to static mapping of the filter to any of the individual resources on the platform. For the CPU and DSP experiments, the ondemand Linux governor selects the DVFS settings for each frame and the three FPGA experiments operate at fixed frequencies of 10, 50 and 100 MHz for FPGA_LP, FPGA_MP and FPGA_HP respectively.

4.2 Comparison of Reinforcement learning Approaches

This section compares the results obtained by reinforcement learning approaches for DPM and DVFS on multi-core systems. Figure 6, extracted from [23], compares six DPM techniques: the original reinforcement learning policy (RL), MLRL without enhancement technique (MLRL[basic]), MLRL with GGAP-RBF [77] without VDBE-softmax [78] (MLRL[RBF]), MLRL with all the enhancement techniques (MLRL[RBF+VDBE]), the golden reference (GOLD), and the BPNN policy (BPNN). The RL policy has least energy saving due to its large state space (at least 2^n) as it directly encodes the power states. For MLRL[basic], the state space is still large on the root node; many samples are needed for n states and n actions, but the agent seldom tries other actions using ϵ -greedy. This leads to higher energy saving than BPNN ($10.99\% > 5.71\%$), demonstrating that the proposed multilevel paradigm is more effective than directly applying function approximation with BPNN. The function approximation MLRL[RBF] achieves energy saving similar to MLRL[basic] (10.99% to 11.31%) and close to the upper bound. Finally, combining VDBE-softmax scheme further saves more energy (11.31% to 13.58%). Furthermore, it has been reported that the performance penalty of MLRL[RBF+VDBE] is close to zero and energy savings are close to GOLD (14.66%).

Figure 7 shows comparison of energy consumption of four DVFS approaches with two different switching intervals: predictive [79], learning-based [53], ondemand [80], learning transfer-based [20]. The energy values are normalized with respect to learning transfer-based approach. Among all these approaches, learning transfer-based approach achieves better energy savings (up to 38 %) and is efficient to adapt to workload and performance variations within the application (intra) and across the applications (inter). Moreover, as per the observations made in [20], both predictive and learning-based approaches fail to meet application performance requirement despite their energy savings.

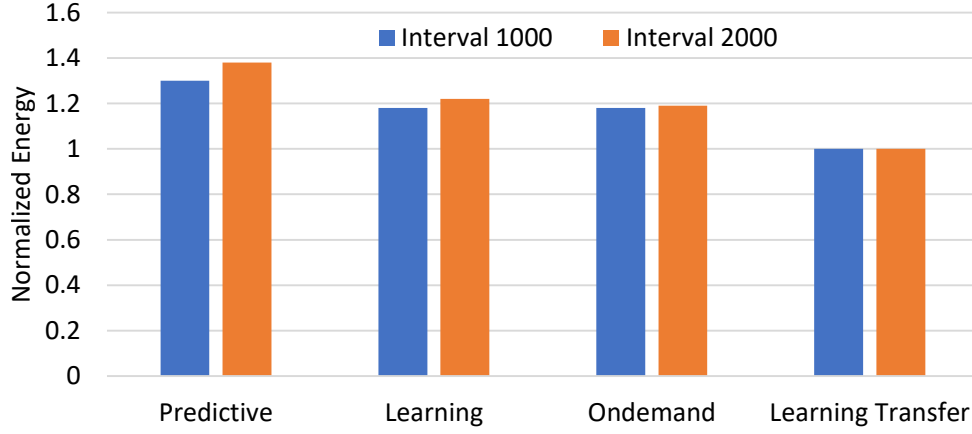


Figure 7: Comparison of normalized energy consumption of various DVFS approaches. Extracted from [20].

5 UPCOMING TRENDS FOR FUTURE RESEARCH AND OPEN CHALLENGES

This section highlights some of the upcoming trends and challenges to be faced to take the learning-based run-time power/energy management approaches for multi/many-cores into the next era.

5.1 Distributed Reinforcement Learning

The literature indicates that many reinforcement learning (RL) algorithms exist to find near-optimal solutions in polynomial time. These algorithms have been applied for single core systems or multi-core systems with small number of cores. Since the number of states increases exponentially with the number of cores, the RL method is known to be non-scalable. An explosion in the state space (number of states) takes place for large scale problems and thus they become very expensive to use at run-time. Therefore, efficient and scalable RL algorithms are desired for large scale many-core systems with hundreds of cores.

In order to address the scalability issues imposed by centralized resource management in many-core systems, distributed or hierarchical management has been employed [29, 81]. In distributed management, each core is checked independently to make management decisions, whereas the cores are grouped into multiple clusters in hierarchical management and each cluster is managed by a local manager. It is evident that hierarchical approach exploits feature of both the centralized and distributed approaches.

Similar to resource management, distributed reinforcement learning is desired to address the scalability issues [52]. In order to reduce the learning complexity, a hierarchical reinforcement learning can be explored, which can provide trade-off between solution quality and computational complexity. The parameters of the hierarchical approach (e.g. cluster size) can be adjusted to achieve several trade-off points, which can be used to optimize energy consumption.

5.2 Learning-based Multi-objective Optimization

We have shown that learning-based power/energy optimization of multi/many-core systems has been extensively focused. However, along with the power/energy, modern multi/many-core systems need to be optimized for several other metrics, e.g. temperature, reliability, fault-tolerance, and security. The temperature is optimized to improve reliability, reduce the cooling cost of many-core based HPC datacenters and mitigate its effect on performance and leakage power. Reliability is to be optimized to increase the mean time to failure of a system. However, in case a fault has happened, optimization need to be performed to achieve fault-tolerance. Optimization for security has become an important concern due to possible attack in the communication channels of connected devices and their interaction with untrusted devices. All these requirements indicate the need for multi-objective optimization.

It has also been observed that learning-based approaches have been used to optimize one metric, e.g. performance while satisfying power budget requirement [52]. In future, it is expected that the requirements to jointly optimize for several performance metrics will grow. Towards it, some progress has already been made, e.g., three metrics execution time, energy consumption and temperature are optimized in [82]. However, in [82], learning-based optimization is not employed. Since learning-based approaches have tremendous potential for optimizations, they should be explored to perform multi-objective optimization.

It is evident that optimization for multiple objectives will increase the design space and thus the learning time. Therefore, the design space needs to be efficiently pruned so that Pareto-fronts for several conflicting objectives can be derived quickly at run-time. In order to maintain a low complexity, the learning process can be bounded by an upper limit on the exploration time.

5.3 Learning-based Optimization for Diverse Application Domains

In addition to multi/many-core systems, learning-based optimization can be performed for several application domains. In fact, it has already be explored for some application domains. For example, a reinforcement learning approach for self optimizing memory controllers [83], learning-based energy management in a hybrid electric vehicles [84] and learning approaches for manufacturing [85]. Such diversity for application domains indicates that, machine learning is effective across a wide spectrum of application domains.

Based on the above, it is expected that machine learning is going to prevail for the design and optimization of systems for various application domains. Several companies have already started projecting the potential of machine learning, e.g., ARM's DynamIQ technology based processors will include dedicated processor instructions for machine learning and are expected to deliver up to a 50x boost in performance over the next 3-5 years relative to Cortex-A73-based systems today. These evidences indicate widespread adoption of machine learning approaches for future computing systems.

6 CONCLUSION

This paper provides a survey of learning-based run-time power and energy management strategies for multi/many-core systems. Specially, model-based and reinforcement learning approaches optimizing for energy consumption is embedded systems, desktop systems and HPC datacenters are surveyed and compared. Based on the analysis of the surveyed and compared learning-based strategies, upcoming trends and open challenges are identified. The research directions highlighted in this survey are expected to advance in future due to potential of learning-based approaches, which will also help to address the open challenges. These advances will need to explore efficient machine learning strategies to take the learning-based approaches for multi/many-core systems into the next era of computing.

ACKNOWLEDGEMENTS

This work was supported in parts by the EPSRC Grant EP/L000563/1 and the PRiME Programme Grant EP/K034448/1 (www.prime-project.org). Experimental data used in this paper can be found at DOI: <http://doi.org/10.5258/SOTON/D0109>.

REFERENCES

- [1] A. Jerraya, H. Tenhunen, and W. Wolf, “Guest Editors’ Introduction: Multiprocessor Systems-on-Chips,” *Computer*, no. 7, pp. 36–40, 2005.
- [2] “Exynos 5 Octa (5422).” www.samsung.com/exynos/, 2016.
- [3] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, “An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS,” in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 98–589, 2007.
- [4] AMD, “AMD Opteron 6000 series processors,” 2011. <http://www.amd.com/en-us/products/server/opteron/6000> (Last visited: 12 February, 2016).
- [5] TILE-Gx, “First 100-core Processor with the New TILE-Gx Family,” 2009. <http://www.tilera.com/> (Last visited: 12 February, 2016).
- [6] B. D. De Dinechin, D. Van Amstel, M. Poulhiès, and G. Lager, “Time-critical computing on a single-chip massively parallel processor,” in *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*, pp. 1–6, 2014.
- [7] L. Benini and G. De Micheli, “Networks on chips: a new SoC paradigm,” *Computer*, no. 1, pp. 70–78, 2002.
- [8] F. Worm, P. Ienne, P. Thiran, and G. De Micheli, “An adaptive low-power transmission scheme for on-chip networks,” in *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*, pp. 92–100, 2002.
- [9] T. Bjerregaard and S. Mahadevan, “A survey of research and practices of Network-on-chip,” *ACM Comput. Surv.*, no. 1, 2006.
- [10] S. Borkar, “Thousand core chips: a technology perspective,” in *Proceedings of ACM Design Automation Conference (DAC)*, pp. 746–749, 2007.
- [11] J. Ceng, J. Castrillon, W. Sheng, H. Scharwächter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda, “MAPS: an integrated framework for MPSoC application parallelization,” in *Proceedings of ACM Design Automation Conference (DAC)*, pp. 754–759, 2008.
- [12] A. Mallik *et al.*, “MNEMEE - An Automated Toolflow for Parallelization and Memory Management in MPSoC Platforms,” in *Proceedings of ACM Design Automation Conference (DAC)*, 2011.
- [13] G. Martin, “Overview of the mpsoc design challenge,” in *Proceedings of ACM Design Automation Conference (DAC)*, pp. 274–279, 2006.
- [14] L. Smit, G. Smit, J. Hurink, H. Broersma, D. Paulusma, and P. Wolkotte, “Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture,” in *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT)*, pp. 421–424, 2004.
- [15] J. Hu and R. Marculescu, “Energy-aware mapping for tile-based NoC architectures under performance constraints,” in *Proceedings of IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 233–239, 2003.
- [16] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, “Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms,” *J. Syst. Archit.*, pp. 242–255, 2010.

- [17] S. Kaushik, A. K. Singh, W. Jigang, and T. Srikanthan, "Run-time computation and communication aware mapping heuristic for noc-based heterogeneous mp soc platforms," in *IEEE International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, pp. 203–207, 2011.
- [18] H. Shen, J. Lu, and Q. Qiu, "Learning based dvfs for simultaneous temperature, performance and energy management," in *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, pp. 747–754, IEEE, 2012.
- [19] D.-C. Juan and D. Marculescu, "Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pp. 97–102, ACM, 2012.
- [20] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 6, pp. 877–890, 2016.
- [21] Z. Wang, Z. Tian, J. Xu, R. K. Maeda, H. Li, P. Yang, Z. Wang, L. H. Duong, Z. Wang, and X. Chen, "Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pp. 684–689, IEEE, 2017.
- [22] W. Liu, Y. Tan, and Q. Qiu, "Enhanced q-learning algorithm for dynamic power management with performance constraint," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 602–605, European Design and Automation Association, 2010.
- [23] G.-Y. Pan, J.-Y. Jou, and B.-C. Lai, "Scalable power management using multilevel reinforcement learning for multiprocessors," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 19, no. 4, p. 33, 2014.
- [24] Y. Wang and M. Pedram, "Model-free reinforcement learning and bayesian classification in system-level power management," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3713–3726, 2016.
- [25] G. Dhiman and T. S. Rosing, "Dynamic power management using machine learning," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pp. 747–754, ACM, 2006.
- [26] V. L. Prabha and E. C. Monie, "Hardware architecture of reinforcement learning scheme for dynamic power management in embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, pp. 1–1, 2007.
- [27] A. K. Singh, A. Das, and A. Kumar, "Energy Optimization by Exploiting Execution Slacks in Streaming Applications on Multiprocessor Systems," in *Proceedings of ACM Design Automation Conference (DAC)*, pp. 115:1–115:7, 2013.
- [28] A. Das, B. M. Al-Hashimi, and G. V. Merrett, "Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 2, p. 24, 2016.
- [29] A. K. Singh, P. Dziurzynski, H. R. Mendis, and L. S. Indrusiak, "A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 24, 2017.
- [30] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, pp. 199–222, Aug. 2004.
- [31] H. Jung and M. Pedram, "Improving the efficiency of power management techniques by using bayesian classification," in *9th International Symposium on Quality Electronic Design (isqed 2008)*, pp. 178–183, March 2008.

- [32] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, (Washington, DC, USA), pp. 318–329, IEEE Computer Society, 2008.
- [33] D.-C. Juan, S. Garg, J. Park, and D. Marculescu, "Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '13, (Piscataway, NJ, USA), pp. 8:1–8:10, IEEE Press, 2013.
- [34] S. Sridharan, G. Gupta, and G. S. Sohi, "Adaptive, efficient, parallel execution of parallel programs," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, (New York, NY, USA), pp. 169–180, ACM, 2014.
- [35] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, (Washington, DC, USA), pp. 213–224, IEEE Computer Society, 2012.
- [36] Y. Wen, Z. Wang, and M. F. P. O'Boyle, "Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms," in *2014 21st International Conference on High Performance Computing (HiPC)*, pp. 1–10, Dec 2014.
- [37] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, pp. 1395–1408, Sept 2010.
- [38] J. Ma, G. Yan, Y. Han, and X. Li, "An analytical framework for estimating scale-out and scale-up power efficiency of heterogeneous manycores," *IEEE Transactions on Computers*, vol. 65, pp. 367–381, Feb 2016.
- [39] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz, "Prediction models for multi-dimensional power-performance optimization on many cores," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp. 250–259, 2008.
- [40] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & cap: Adaptive dvfs and thread packing under power caps," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, (New York, NY, USA), pp. 175–185, ACM, 2011.
- [41] H. Sasaki, S. Imamura, and K. Inoue, "Coordinated power-performance optimization in manycores," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, pp. 51–61, Sept 2013.
- [42] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi, "Adaptive energy minimization of embedded heterogeneous systems using regression-based learning," in *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 103–110, Sept 2015.
- [43] Y. Wu, D. S. Nikolopoulos, and R. Woods, "Runtime support for adaptive power capping on heterogeneous socs," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pp. 71–78, July 2016.
- [44] H. Shen and Q. Qiu, "Contention aware frequency scaling on cmps with guaranteed quality of service," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, March 2014.

- [45] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "power" struggles: Coordinated multi-level power management for the data center," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XIII*, (New York, NY, USA), pp. 48–59, ACM, 2008.
- [46] P. Lama, Y. Guo, C. Jiang, and X. Zhou, "Autonomic performance and power control for co-located web applications in virtualized datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, pp. 1289–1302, May 2016.
- [47] M. A. H. Monil, R. Qasim, and R. M. Rahman, "Energy-aware vm consolidation approach using combination of heuristics and migration control," in *Ninth International Conference on Digital Information Management (ICDIM 2014)*, pp. 74–79, Sept 2014.
- [48] D. H. Woo and H.-H. S. Lee, "Extending amdahl's law for energy-efficient computing in the many-core era," *Computer*, vol. 41, pp. 24–31, Dec. 2008.
- [49] M. A. Suleman, M. K. Qureshi, and Y. N. Patt, "Feedback-driven threading: Power-efficient and high-performance execution of multi-threaded workloads on cmps," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XIII*, (New York, NY, USA), pp. 277–286, ACM, 2008.
- [50] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Id-gunji, E. Ozer, and B. Falsafi, "Scale-out processors," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 500–511, June 2012.
- [51] J. Ng, X. Wang, A. K. Singh, and T. Mak, "Defragmentation for efficient runtime resource management in noc-based many-core systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 11, pp. 3359–3372, 2016.
- [52] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1521–1526, EDA Consortium, 2015.
- [53] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, p. 24, 2013.
- [54] R. Ye and Q. Xu, "Learning-based power management for multicore processors via idle period manipulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 7, pp. 1043–1055, 2014.
- [55] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation," in *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pp. 65–73, IEEE, 2006.
- [56] J. L. Berral, Í. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavalda, and J. Torres, "Towards energy-aware scheduling in data centers using machine learning," in *Proceedings of the 1st International Conference on energy-Efficient Computing and Networking*, pp. 215–224, ACM, 2010.
- [57] G. Tesauro *et al.*, "Online resource allocation using decompositional reinforcement learning," in *AAAI*, vol. 5, pp. 886–891, 2005.
- [58] G. Tesauro, R. Das, W. E. Walsh, and J. O. Kephart, "Utility-function-driven resource allocation in autonomic systems," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pp. 342–343, IEEE, 2005.

- [59] X. Lin, Y. Wang, and M. Pedram, "A reinforcement learning-based power management framework for green computing data centers," in *Cloud Engineering (IC2E), 2016 IEEE International Conference on*, pp. 135–138, IEEE, 2016.
- [60] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pp. 500–507, IEEE, 2014.
- [61] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pp. 3–14, IEEE, 2001.
- [62] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, vol. 2011, 2009.
- [63] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *ACM SIGARCH Computer Architecture News*, vol. 23, pp. 24–36, ACM, 1995.
- [64] R. K. Maeda, P. Yang, X. Wu, Z. Wang, J. Xu, Z. Wang, H. Li, L. H. Duong, and Z. Wang, "Jade: a heterogeneous multiprocessor system simulation platform using recorded and statistical application models," in *Proceedings of the 1st International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems*, p. 8, ACM, 2016.
- [65] Z. Wang, W. Liu, J. Xu, B. Li, R. Iyer, R. Illikkal, X. Wu, W. H. Mow, and W. Ye, "A case study on the communication and computation behaviors of real applications in noc-based mpsocs," in *VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on*, pp. 480–485, IEEE, 2014.
- [66] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture*, pp. 347–358, IEEE Computer Society, 2006.
- [67] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pp. 29–40, ACM, 2010.
- [68] M. D. Pendrith and C. Sammut, *On reinforcement learning of control actions in noisy and non-Markovian domains*. Citeseer, 1994.
- [69] K. Sikorski and T. Balch, "Model-based and model-free learning in markovian and non-markovian environments," in *Proceedings of Agents-2001 Workshop on Learning Agents*, 2001.
- [70] D. V. Djonin and V. Krishnamurthy, "V-blast power and rate control under delay constraints in markovian fading channels-optimality of monotonic policies," in *Information Theory, 2006 IEEE International Symposium on*, pp. 2099–2103, IEEE, 2006.
- [71] G. Dhiman and T. S. Rosing, "Dynamic power management using machine learning," in *Proceedings of the 2006 IEEE/ACM International Conference on Computer-aided Design, ICCAD '06*, (New York, NY, USA), pp. 747–754, ACM, 2006.
- [72] R. Ubal, J. Sahuquillo, S. Petit, and P. López, "Multi2sim: A simulation framework to evaluate multicore-multithread processors," in *IEEE 19th International Symposium on Computer Architecture and High Performance computing, page (s)*, pp. 62–68, Citeseer, 2007.

- [73] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pp. 469–480, IEEE, 2009.
- [74] T. Simunic, L. Benini, P. Glynn, and G. De Micheli, "Event-driven power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 7, pp. 840–857, 2001.
- [75] S. J. Bradtke and M. O. Duff, "Reinforcement learning methods for continuous-time markov decision problems," *Advances in neural information processing systems*, pp. 393–400, 1995.
- [76] "MediaBench." <http://euler.slu.edu/?fritts/mediabench/>.
- [77] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.
- [78] M. Tokic and G. Palm, "Value-difference based exploration: Adaptive control between epsilon-greedy and softmax.," in *KI*, pp. 335–346, Springer, 2011.
- [79] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Proceedings of the 2004 international symposium on Low power electronics and design*, pp. 174–179, ACM, 2004.
- [80] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, vol. 2, pp. 215–230, sn, 2006.
- [81] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on Multi/Many-core Systems: Survey of Current and Emerging Trends," in *Proceedings of ACM Design Automation Conference (DAC)*, pp. 1:1–1:10, 2013.
- [82] H. F. Sheikh and I. Ahmad, "Efficient heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-core processors," in *IEEE International Green Computing Conference (IGCC)*, pp. 1–8, 2014.
- [83] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pp. 39–50, IEEE, 2008.
- [84] X. Lin, P. Bogdan, N. Chang, and M. Pedram, "Machine learning-based energy management in a hybrid electric vehicle to minimize total operating cost," in *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*, pp. 627–634, IEEE, 2015.
- [85] L. Monostori, A. Márkus, H. Van Brussel, and E. Westkämpfer, "Machine learning approaches to manufacturing," *CIRP Annals-Manufacturing Technology*, vol. 45, no. 2, 1996.

7 BIOGRAPHIES

Amit Kumar Singh received the B.Tech. degree in Electronics Engineering from Indian Institute of Technology (Indian School of Mines), Dhanbad, India, in 2006, and the Ph.D. degree from the School of Computer Engineering, Nanyang Technological University (NTU), Singapore, in 2013. He was with HCL Technologies, India for year and half before starting his PhD at NTU, Singapore, in 2008. He worked as a post-doctoral researcher at National University of Singapore (NUS) from 2012 to 2014 and at University of York, UK from 2014 to 2016. Currently, he is working as senior research fellow at University of Southampton, UK. His current research interests include system level design-time and run-time optimizations of 2D and 3D multi-core systems with focus on performance, energy, temperature, and reliability. He has published over 45 papers in the above areas in leading international journals/conferences. Dr. Singh was the receipt of ISORC 2016 Best Paper Award, PDP 2015 Best Paper Award, HiPEAC Paper Award, and GLSVLSI 2014 Best Paper Candidate. He has served on the TPC of IEEE/ACM conferences like ISED, MES, NoCArc and ESTIMedia.

Charles Leech is currently a Senior Research Assistant working for the PRiME Project in the Department of Electronics and Computer Science at the University of Southampton where he is completing his PhD and received a BEng Hons degree in Electronic Engineering. His focus is on the development of a cross-layer framework for run-time management and system-level approximate computing on embedded platforms. His work also includes power and performance optimisation of next-generation applications, including computer vision, on heterogeneous many-core systems.

Basireddy Karunakar Reddy received his M.Tech. degree in Microelectronics and VLSI from Indian Institute of Technology (IIT), Hyderabad, India in 2015. He is a Ph.D. student in Electronic and Electrical Engineering at the University of Southampton. His research interests include design-time and run-time optimization of performance and energy in many-core heterogeneous systems.

Bashir M. Al-Hashimi is an ARM Professor of Computer Engineering, Dean of the Faculty of Physical Sciences and Engineering, and the Co-Director of the ARM-ECS Research Centre, University of Southampton, Southampton, U.K. He has published over 380 technical papers. His current research interests include methods, algorithms, and design automation tools for low-power design and test of embedded computing systems. He has authored or co-authored five books and has graduated 35 Ph.D. students.

Geoff V. Merrett received the B.Eng. degree (Hons.) in electronic engineering and the Ph.D. degree from the University of Southampton, Southampton, U.K., in 2004 and 2009, respectively. He is currently an Associate Professor in electronic systems with the University of Southampton. His current research interests include low-power and energy harvesting aspects of embedded & mobile systems. He has published over 100 articles in journals/conferences in the above areas. Dr. Merrett was the General Chair of the Energy Neutral Sensing Systems Workshop from 2013 to 2015. He is a fellow of the The Higher Education Academy.

High Speed Low Complexity Guided Image Filtering Based Disparity Estimation

Charan Kumar Vala, Koushik Immadisetty, Amit Acharyya, *Member, IEEE*, Charles Leech, Vibishna Balagopal, Geoff V. Merrett, *Member, IEEE*, and Bashir M. Al-Hashimi, *Fellow, IEEE*,

Abstract—Stereo vision is a methodology to obtain depth in a scene based on the stereo image pair. In this paper we introduce a Discrete Wavelet Transform (DWT) based methodology for a state-of-the-art disparity estimation algorithm, that resulted in significant performance improvement in terms of speed and computational complexity. In the initial stage of the proposed algorithm, we apply DWT to the input images, reducing the number of samples to be processed in subsequent stages by 50%, thereby decreasing computational complexity and improving processing speed. Subsequently the architecture has been designed based on this proposed methodology and prototyped on a Xilinx Virtex-7 FPGA. The performance of the proposed methodology has been evaluated against four standard Middlebury Benchmark image pairs viz. *Tsukuba*, *Venus*, *Teddy* and *Cones*. The proposed methodology results in improvement of about 44.4% cycles per frame, 52% frames per second and 61.5% and 59.6% LUT and register utilization respectively, compared with state-of-the-art designs.

Index Terms—Guided Image Filtering, Stereo-Matching, FPGA, Discrete Wavelet Transform, Low-Complexity, High-Speed.

I. INTRODUCTION

STEREO matching, the task of matching images acquired by a pair of cameras (conventionally called reference and target images) and calculating the depth of objects in a scene, is being employed in sophisticated embedded vision applications including surveillance, autonomous vehicles and mobile robots [1]. A geometrical representation of the same is shown in Fig. 1. Since such systems are mostly used in mobile environments running from battery or harvested energy, the primary design-challenges are low power consumption and area-overhead reduction, whilst meeting the real-time processing requirement with acceptable precision. Hence the choice of matching algorithm and respective architecture implementation play an important role to tackle these challenges

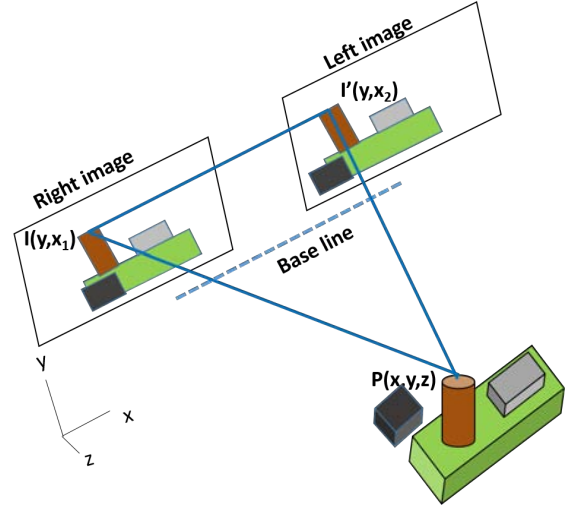


Fig. 1. Geometry of a stereo vision system where a pair of parallel cameras capture the scene from horizontally disparate viewpoints. Given that the input images are rectified, the correspondence of a pixel at coordinate (x, y) of the reference image can only be found at the same vertical coordinate y within a disparity range $(D_{min}$ to D_{max}) in the target image.

and meet these constraints for producing a viable embedded stereo matching system.

Thorough analysis of the existing literature shows that, despite having low-error rate in the disparity computation, state-of-the-art DSPs can not support global stereo matching algorithms due to intensive computational needs [2], [3]. A viable alternative to this could be GPU-based implementations but at the expense of high cost and power-consumption for real-time designs [4]. On the other hand, due to reduced algorithmic complexity, local stereo matching algorithms were getting implemented over the FPGA and ASIC platform to meet real-time requirements with higher frames per second compared to its global matching counter-part [5]–[7]. However the concern with the local matching algorithms is its low precision. To mitigate this issue, investigations have been made to implement dedicated hardware architectures of more precise algorithms, such as Semi Global Matching (SGM) [8], [9] and Adaptive Support Weight (ADSW) [10], [11]. For the past few years, hardware implementations predicated on SGM and ADSW algorithms have become the preferred solution towards higher matching precision in embedded vision applications [5], [7], [12], [13]. In addition, modifications and improvements have been made to implementations to habituate

CKV, CL, GVM and BMAH are with the School of Electronics and Computer Science, University of Southampton, UK. E-mail: (C.K.V, cl19g10, gvm, bmah)@ecs.soton.ac.uk

Koushik Immadisetty with Qualcomm Innovation Center Inc.: E-mail: kimmadis@qti.qualcomm.com

Amit.A with the Department of Electrical Engineering, Indian Institute of Technology Hyderabad, India. E-mail: (amit_acharyya)@iith.ac.in

Vibishna Balagopal with LumiraDx Technology, U.K E-mail: Vibishna.Balagopal@gmail.com

This research was supported in part by EPSRC grant EP/K034448/1 www.prime-project.org

Amit Acharyya would also like to acknowledge his Visvesvaraya Young Faculty Fellowship by the Ministry of electronics and Information Technology, Govt. of India.

Data supporting this research can be found at DOI <http://doi.org/10.5258/SOTON/D0170>

the algorithms for real-time processing, with improvement in execution times over existing designs [14] but at the cost of increased error rates when compared to state-of-the-art software implementations [15]. Furthermore, high memory and significant need of hardware resources are the bottleneck towards the scalability of these designs to higher resolution images. We believe that efficiently reducing the number of samples to be processed at different stages of the algorithm will reduce the consumption of hardware resources, improving the scalability of these designs. With this motivation, we introduce the Discrete Wavelet Transform (DWT) and the Inverse Discrete Wavelet Transform (IDWT) at the pre and post-processing stages of the traditional Guided Image Filter (GIF) based disparity estimation process flow, which results in reduced processing-data sizes in the intermediate stages and thus less complex hardware in the FPGA implementation and throughput improvement. It is to be noted that the DWT has been exploited in state-of-the-art disparity estimation for extracting the feature points of the image in [16] and for improving the quality of disparity map in [17], [18]. To the best of our knowledge it has never been attempted in GIF based disparity estimation on hardware complexity reduction.

The GIF algorithm has been shown to reduce the complexity of the cost aggregation step in local ADSW algorithms [19], [20]. Recently published literature detailed the fully pipelined and parallel GIF architecture and corresponding FPGA based implementations [21], [22]. However, instead of designing a stand-alone architecture, we believe holistic optimizations of algorithm and architecture would lead to improved performance. With this motivation, in this paper we:

- Introduce a Discrete Wavelet Transform (DWT) based methodology for the disparity estimation algorithm.
- Design the respective architecture and prototype it on a Xilinx Virtex-7 FPGA and validate it against the standard databases, comparing the performance to the outcomes of existing designs.
- Show that the proposed methodology results in improvement of about 44.4% Cycles per Frame (CPF), 52% Frames per Second (FPS) and 61.5% and 59.6% LUT and register utilization respectively compared with the state-of-the art designs [9], [13], [23].

The rest of the paper is organized as follows: Section II provides the necessary theoretical background, Section III introduces the proposed methodology and subsequent architecture, Section IV discusses the experimental results, compares performance of the proposed methodology with existing designs and finally Section V concludes the discussion. The acronyms used throughout the paper are explained in Table I.

II. RELATED WORKS

Stereo matching algorithms can broadly be classified into two categories: global and local [24]. Global algorithms are formulated as an energy minimization problem, which is solved with techniques such as Dynamic Programming, Graph Cuts and Belief Propagation. Such methods produce very accurate results at the expense of high computational complexity and memory needs. Semi-Global Matching (SGM) [8], [9],

TABLE I
FULL FORM OF ACRONYMS USED IN THE PAPER

Symbol	Explanation
BRAM	Block Random Access Memory
corr	Correlation
cov	Covariance
CPF	Cycles per Frame
CVC	Cost Volume Construction
CVF	Cost Volume Filtering
D_{max}	Maximum Disparity
D_{min}	Minimum Disparity
DWT	Discrete Wavelet Transform
FPS	Frames per second
GIF	Guided Image Filter
H_{ei}	Hight of image
W_{id}	Width of image
IDWT	Inverse Discrete Wavelet Transform
LUT	Look Up Table
MDE/s	Million Disparity Estimation per second
SAD	Sum of Absolute Difference
WTA	Winner-take-all

[15] methods renounce part of the accuracy by approximating a global 2D function using a sum of 1D optimizations from all directions through the image. SGM methods are therefore more affordable for dedicated hardware implementations, but still consume significant memory to store the interim cost of different aggregation paths. In contrast, local algorithms use block matching and winner-takes-all optimization to determine the disparity associated with a minimum cost function at each pixel [3]. Hence, they have lower computational complexity and memory requirements compared to global and SGM methods.

Among local algorithms, the most recent Adaptive Support Weight (ADSW) methods are currently the most accurate [2]. Despite their good results, ADSW algorithms cannot take advantage of the integral image or sliding window techniques, as the adaptive weights have to be recomputed at every pixel. This makes the cost aggregation's hardware complexity directly dependent on the support window size. To improve matching accuracy, a few attempts have been made by combining different stereo algorithms together or by implementing modified versions of SGM and ADSW algorithms. A modified version of the census transform in both the intensity and gradient images, in combination with the SAD correlation metric has been implemented in hardware [6]. A stereo algorithm based on the neural network and Disparity Space Image (DSI) data structure is introduced in [7] and implemented on an FPGA. Zhang *et. al.* combined both the mini-census transform and cross based cost aggregation for implementing real-time FPGA based stereo matching [25]. SGM-based stereo matching systems have been introduced in [8], [9], [15] and implemented on FPGAs and a hybrid FPGA/RISC architecture based platforms respectively. The VLSI design of an ADSW algorithm that adopted the mini-Census transform was implemented to improve the accuracy and robustness of the system to radiometric distortions [14]. Incorporating an ADSW algorithm and integration of pre and post- processing units, [11] proposed the implementation of a complete stereo vision system. Finally, a hardware oriented stereo matching

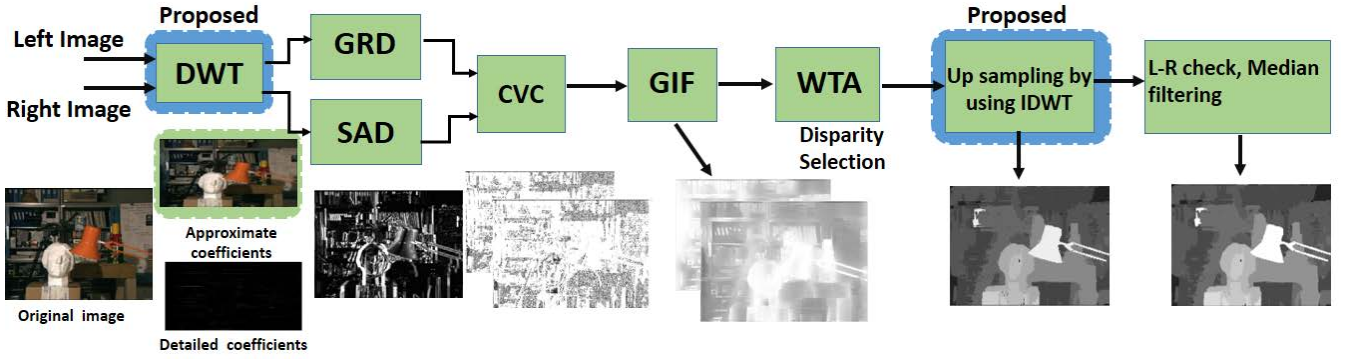


Fig. 2. Proposed DWT based methodology and the flow of Disparity estimation algorithm.

system based on the adaptive Census transform is presented in [10].

The aforementioned high-quality ADSW-predicated systems follow a homogeneous algorithm-to-hardware mapping methodology. The recently proposed Guided Image Filter (GIF) [19] has been employed in [20]–[22] to reduce the complexity of the cost aggregation step in ADSW methods, leading to a high-quality fast and simple algorithm consisting of the following steps: cost volume construction, cost volume filtering, disparity selection and disparity refinement. Cost Volume Construction (CVC) is a measure of calculating a cost between two pixels, one from the left image and one from the right image, to identify the disparity corresponding to the depth of a point in the scene [3]. The information obtained by matching single pixels is not sufficient for precise matching. So for accomplishing precise matching and to minimize the matching uncertainties, cost volume filtering is used in the next step. Following filter, the most probable disparity is selected by using a local winner-takes-all (WTA) strategy. Finally, a refinement step is used to reduce noise and improve the disparity map [3]. Recently, Ttofis *et al.* [21], [22] proposed a fully pipe-lined, parallel stereo matching FPGA-based hardware architecture based on the GIF, achieving real-time processing for high definition (HD) images.

III. PROPOSED METHODOLOGY AND ARCHITECTURE

A. Proposed Methodology

1) *Overview*: Fig. 2 shows the entire flow of the DWT based stereo vision algorithm, comprised of six stages. The proposed DWT is the first stage that receives the rectified stereo images captured from a stereo camera. The DWT transforms these images into approximate wavelet coefficients and feeds these coefficients into the second stage of CVC, which includes the Sum of Absolute Differences (SAD) and Gradient (GRD) stages. The GIF is used to filter each cost volume slice. In the fourth stage, the minimum among all cost volumes is selected by the WTA strategy for disparity selection. The first instance of the actual disparity (equivalent to the size of original image) is synthesized from the wavelet-domain using IDWT as part of the fifth stage, followed by post-processing in the sixth stage which includes left and right consistency check and median filtering.

2) *Proposed DWT based methodology*: In the context of stereo-vision algorithm, we propose to use a DWT as the first stage to reduce the number of samples required for subsequent computations [26]–[30]. Exploiting the property in stereoscopic images that neighboring pixels are highly correlated, without any loss of generality, the first resolution level of approximate wavelet coefficients can be considered for further computation and the detailed wavelet coefficients can be left apart, which results in simple down-sampling. This is evident in Fig. 2 where the original image, approximate wavelet coefficients and detailed wavelet coefficients are shown. When the approximate and detailed coefficients are compared, the approximate wavelet coefficients retain most of the features of the original image. Therefore, without sacrificing matching accuracy, the overall number of samples for subsequent computations can be reduced to half as compared to the original image. The Haar wavelet is chosen as the mother wavelet, because it can be implemented by using simple adder and shifter leading to a low-complexity hardware implementation. The approximate wavelet coefficients of the input images from the first resolution level, after applying a vertical 1D Haar mask [0.5, 0.5], can be represented as:

$$I_{left}^i(j) = (L^i(2j-1) + L^i(2j))/2 \quad (1)$$

$$I_{right}^i(j) = (R^i(2j-1) + R^i(2j))/2 \quad (2)$$

where I_{left}^i and I_{right}^i are the approximate wavelet coefficients of the input stereo images respectively (represented as L and R), i denotes the color channel in RGB space and $j = 1 : Hei/2 \forall Wid$.

At this stage the number of pixels of the input images to be processed are halved, i.e from $Hei \times Wid$ to $(Hei \times Wid)/2$ and thereby significantly reducing the computational complexity of subsequent modules.

3) *Processing of the Approximate Wavelet Coefficients using GIF Based stereo matching [20]*: CVC involves the cost computation of each pixel between the stereo images over a range of disparities d by considering the truncated absolute difference of colors and gradients as follows:

$$C(p, d) = a \cdot \min(T_c, M(p, d)) + (1 - a) \cdot \min(T_g, G(p, d)) \quad (3)$$

where,

$$M(p, d) = \sum_{i=1}^3 |I_{left}^i(p) - I_{right}^i(p-d)| \quad (4)$$

and

$$G(p, d) = |\nabla_x I_{left}^i(p) - \nabla_x I_{right}^i(p-d)|, \quad (5)$$

where i denotes the color channel in RGB space, $\nabla_x I_{left}^i$ denotes the gradient in x direction computed at pixel p , a is used to balance the influence of the color and gradient terms, T_c, T_g are the truncation thresholds respectively.

Next, the GIF is used to filter the cost volume. The filtered cost of a pixel p at disparity- d is given by:

$$q(p, d) = \sum W_{i,j}(I)C(p, d) \quad (6)$$

where,

$$W_{i,j} = \frac{1}{|w|^2} \sum_{k:(i,j) \in \omega_k} \left(1 + \frac{(I_i - \mu_k)(I_j - \mu_k)}{\sigma_k^2 + \varepsilon}\right) \quad (7)$$

In (6) and (7), i and j are pixel indices, $W_{i,j}$ is the weight corresponding to j in the window W_k with center i and radius r , $|w|$ is the number of pixels in W_k (i.e., $(2r+1)x(2r+1)$), μ_k and σ_k are the mean and the standard deviation of I (the first resolution level of wavelet approximate coefficients from the guidance image) in W_k . The weights ($W_{i,j}$) can be computed using linear operations [19] which can be decomposed into a series of mean filters with windows of radius r . The pseudo code of the GIF is given in Algorithm 1, where f_{mean} is a boxfilter (same as a mean filter) with a window of radius r . The abbreviations of correlation ($corr$), variance (var), and covariance (cov) hold their usual meaning.

Disparity selection, involving the condensation of the cost volume back into a single image, is performed through the WTA strategy where $d \in [D_{min}, D_{max}]$:

$$d'_p = \operatorname{argmin}_d(q(p, d)) \quad (8)$$

4) *Proposed IDWT based disparity estimation*: The first disparity map, d_p , that is the size of the original image is synthesized from d'_p by applying the IDWT as follows:

$$d_p(2k) = (d'_p(k) + d'_p(k+1))/2 \quad (9a)$$

$$d_p(2k-1) = d'_p(k) \quad (9b)$$

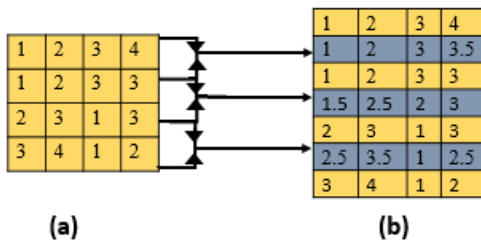


Fig. 3. Computed disparity in the wavelet domain (left) and the up-sampled disparity after IDWT (right).

Algorithm 1 Guided Image Filter [20]

```

1: INPUT: guidance image I, guided image p
2: meanI = fmean(I); meanP = fmean(P); corrI = fmean(I.*I);
   corrIP = fmean(I.*P);
3: varI = corrI - meanI.*meanI; covIP = corrIP - meanI.*meanP;
4: a = covIP/(varI+ε); b = meanP - a.*meanI;
5: meana = fmean(a); meanb = fmean(b);
6: q = meana.*I+meanb;
7: OUTPUT: output q (filtered image)

```

Algorithm 2 Proposed Disparity Estimation Algorithm

```

1: INPUT: pair of stereo images L(left), R(right)
2: for k = R,G,B do
3:   for i = 1 to Hei/2 do
4:     for j = 1 to Wid do
5:       Ileft(i,j,k) = (L(2i-1,j,k) + L(2i,j,k))/2
6:       Iright(i,j,k) = (R(2i-1,j,k) + R(2i,j,k))/2
7:     end for
8:   end for
9: end for
10: Find the left and right grayscale images
11: Cost Volume Construction:
12: for Disparity = Dmin to Dmax do
13:   for i = 1 to Hei/2 do
14:     for j = 1 to Wid do
15:       Find the SAD (4)
16:       Find the gradient (5)
17:       Find the cost (3)
18:     end for
19:   end for
20: end for
21: Cost Volume Filtering using GIF:
22: for CVC = Dmin to Dmax do
23:   Repeat GIF Algorithm 1
24: end for
25: Disparity Selection:
26: for CVF = Dmin to Dmax do
27:   dp' = argmind(q(p, d))
28: end for
29: Up-sampling of disparity:
30: for i = 1 to Hei/2 do
31:   for j = 1 to Wid do
32:     dp(2i,j) = (dp'(i,j) + dp'(i+1,j))/2
33:     dp(2i-1,j) = dp'(i,j)
34:   end for
35: end for
36: Post Processing:
37: Left-right consistency check and filling
38: Disparity refinement with a median filter
39: OUTPUT: Final Disparity

```

where $k = 1 : Hei/2 \forall Wid$. A demonstration of the up-sampling strategy for a 16-pixel template is shown in Fig. 2. To reduce noise and improve the quality of disparity map, a post-processing step is performed. This involves a left-right (L-R) consistency check. The L-R check requires the computation of both the left and right disparity maps. Pixels in the left disparity map are marked as inconsistent if the disparity value of its matching pixel in the right disparity map differs by more than one pixel. Inconsistent pixels are then filled by the disparity of the closest consistent pixel [20] and a median filter is used to smooth the filled regions and remove spikes. The pseudocode of the proposed DWT based disparity estimation methodology is given in Algorithm 2.

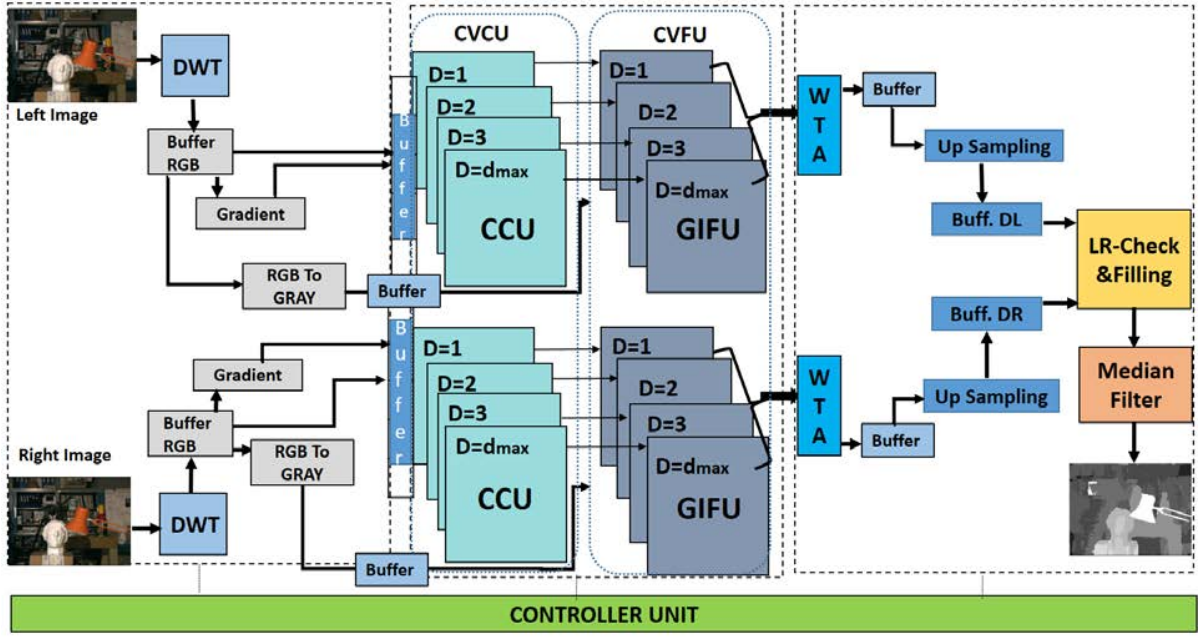


Fig. 4. Disparity estimation architecture based on the proposed DWT based methodology.

B. Architecture based on proposed methodology

Fig. 4 shows the disparity estimation architecture based on the proposed methodology. It can be observed that the architecture comprises of DWTs, RGB to gray scale image converters, gradient computation modules, Cost Computation Units (CCUs), Guided Image Filtering Units (GIFUs), WTAs, Upsampling modules, LR-check module, median filter and a controller. All the modules are fully pipelined to obtain high throughput. The rectified reference and target image's RGB data enters the processing pipeline consisting of the modules mentioned above. After multiple setup cycles from the pipeline latency, computed depth maps synchronized with the input pixel rate are forwarded successively through the scan-line to the output port.

For the purpose of demonstrating the proof of concept, 6 DWT modules (one for each color channel) for left and right R,G,B color channels based on the Haar wavelet (Equations 1 and 2) have been designed, each consisting of one adder and one right shifter. The outputs of DWT module are stored in a BRAM. At this stage, the number of samples is reduced by half. The gradient module comprises of 6 subtracters for calculating the left and right RGB gradients in the X-direction. The gradient values of the left and right images are then stored in the buffer. Furthermore, these values are accessed in the next stage for CVC.

The inputs to the cost volume construction unit (CVCU) are sent through a buffer, Fig. 5a, where two shift registers (length D_{max}) send data to the corresponding CVC module through a 2×1 mux depending on the select signal. The shift registers take the data from the input RGB BRAM alternately depending on the enable signal. All the remaining buffers in the design are realized using the FPGA's internal BRAMs.

The CVCU in Fig. 4 employs a cascade of CCUs to

calculate the pixel-wise cost between the pixel p in the left image and the corresponding $(p - d)$ pixel in the right image. The architecture of CCU, based on Equations (3) - (5), is shown in Fig. 5b. It consists of absolute difference units, adders and comparators that calculate the truncated color and gradient costs, which are summed to compute the overall cost. Prior to the summation, the truncated color and gradient costs are multiplied by constant values to normalize the overall cost. To avoid multiplication and without any loss of generality, these constants are selected to be powers of 2, which can be realized with shifting operations alone.

The output of the CCU is fed into the cost volume filtering unit (CVFU) in a row-wise manner. The CVFU employs a cascade of GIFUs. The architecture of GIFU is based on the pseudocode of Algorithm 1 and is shown in Fig. 5c. Each GIFU comprises boxfilters, subtracters, an adder and a shifter. Among all the components in GIFU, the boxfilter is the computationally intensive component. However, since the number of input samples to be processed has been reduced by the DWT module, the boxfilters benefit from the reduction in the number of computations per image.

The architecture of the boxfilter is shown in Fig. 6a. The boxfiltering operation for a 6×7 image template with window radius 1 is illustrated in Fig. 6b, and is explained as follows: If pixel- i of an image is updated then the mean of the box with pixel- i as end pixel is given by the boxfilter. The basic operation of the boxfilter is to maintain each column sum by adding the $(2r+1)$ pixels. The box sum is computed by adding the $(2r+1)$ adjacent column sums. From Fig. 6b, if pixel- i is updated, then the corresponding column sum is updated by adding and subtracting the new and old pixels respectively. In the same way, the box sum is computed by adding and subtracting the new and old column sums respectively. Storing

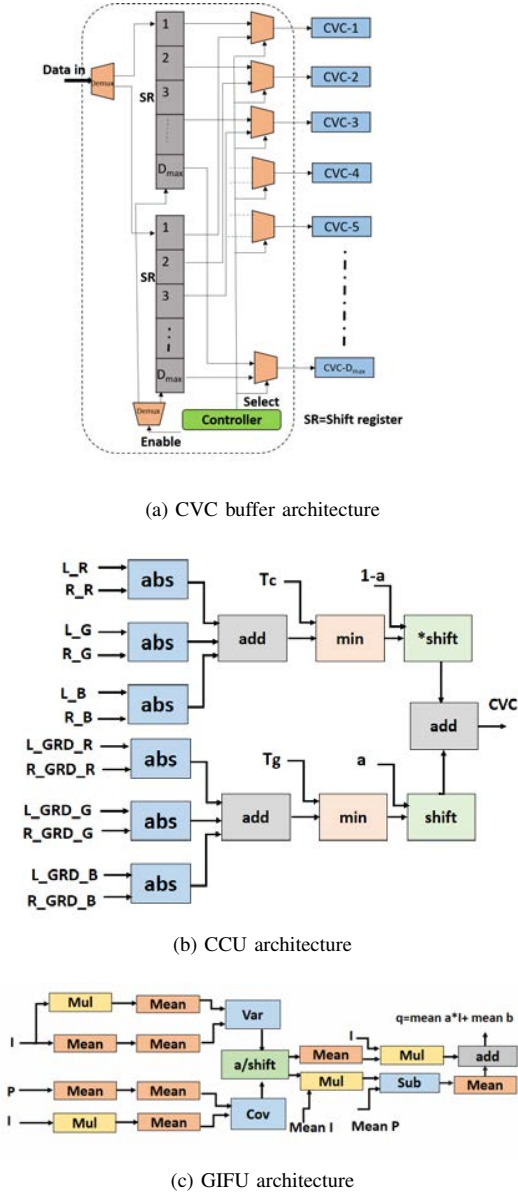


Fig. 5. The architectures of the CVC buffer, CCU and GIFU

only $(2r + 1)$ rows of pixels for boxfilter operations would be sufficient, resulting in reduction of memory consumption from entire image to just $(2r + 1)$ rows of pixels. The boxfilter architecture (Fig. 6a) has two submodules; the column sum computation unit (where the column sum to be updated is computed) and the box sum computation unit (where the box sum to be updated is computed).

The operation of the GIFU for a 6×7 image template and window radius 1 is shown in Fig. 7 and is explained according to the pseudocode of the GIF (algorithm 1) as follows: If pixel- i of an image is known then the mean of box with pixel- i as end pixel is computed by a box-filter (as described above). Similarly, from Fig. 7 as pixel- i of I (guidance image) and P (guided image) are updated, II and IP of pixel- i are computed by two multipliers. $mean_I$, $mean_P$, $mean_{II}$, $mean_{IP}$, var_I , cov_{IP} of pixel- j are computed by four

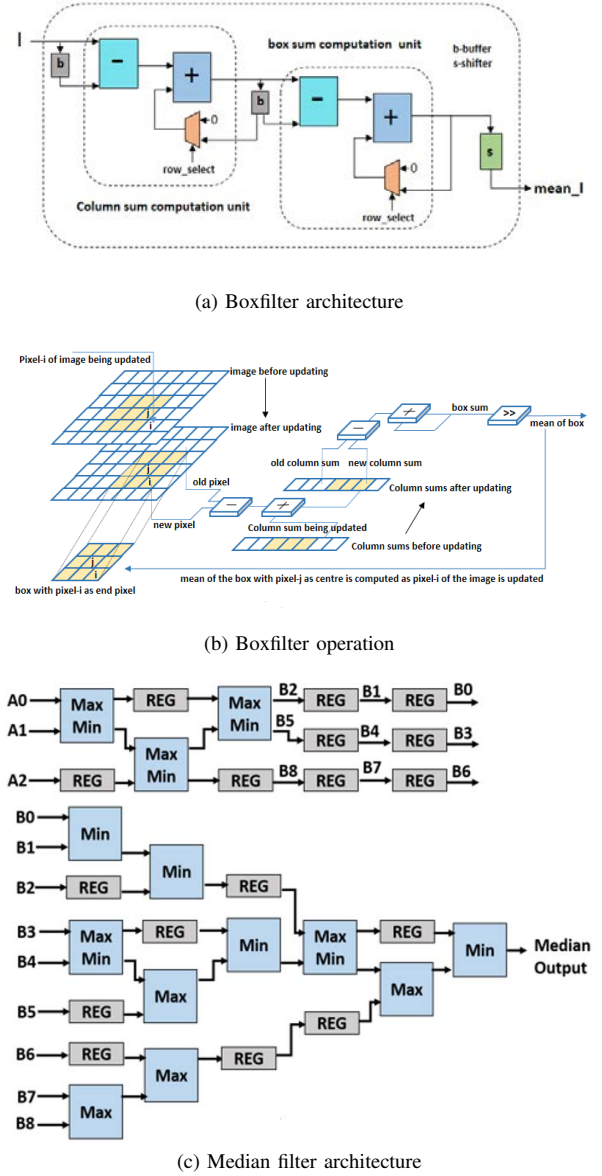


Fig. 6. The architecture of the boxfilter, an illustration of its operation and the architecture of the median filter.

box filters, four multipliers and two subtractors. Consequently, the a and b values of pixel- j are computed by a shifter, multiplier and subtractor. In a similar way, since the a and b values of pixel- j are known, the mean of the box with pixel- j as the end pixel, i.e. $mean_a$, $mean_b$ of pixel- k are computed by two box filters. Finally, q (i.e., filtered output) of pixel- k is computed by a multiplier and an adder. Thus the entire architecture of the CVFU is pipelined in parallel.

Two WTA modules, shown in Fig. 4, are used for the left and right disparity selection respectively. Each WTA module comprises of comparators organized in tree structures with $\log_2 D_{max}$ stages. The disparities obtained are stored in the buffers before being transferred to the up-sampling units which consist of an adder and shifter. The up sampled disparities are stored in the buffers DL and DR, as shown in Fig. 4, from where they are accessed by the post processing unit. The

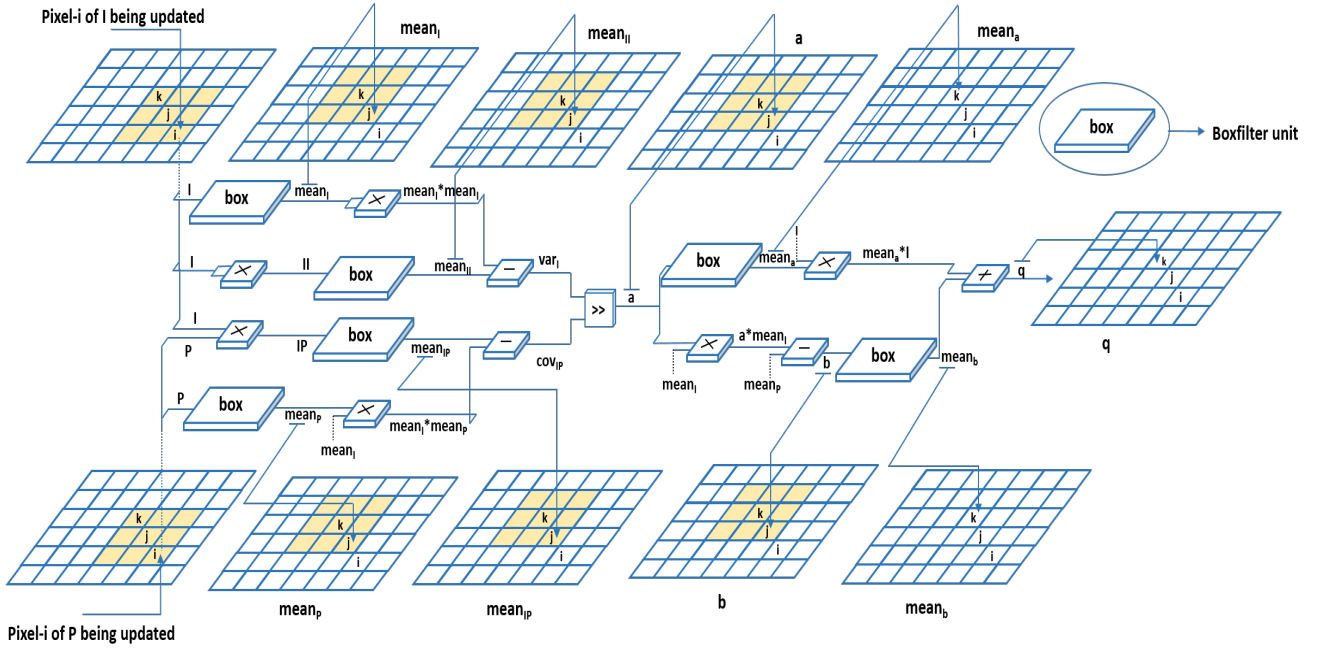


Fig. 7. The operation of the Guided Image Filtering Unit (GIFU).

Post Processing module implements the L-R check and filling (interpolation) through a set of comparators, multiplexer trees, and priority encoders that locate the nearest valid disparities in the left and right direction of the invalid pixels. To reduce noise and eliminate any remaining spikes in interpolated disparity map, an energy-efficient 3x3 median filter design has been adopted from [31], [32] as shown in Fig. 6c, where A_0, A_1, A_2 are the input disparities. The reason for choosing a 3x3 median filter is explained in Section IV.

IV. RESULTS AND DISCUSSION

To validate the proposed methodology, the stereo matching core has been implemented on a Xilinx Virtex-7 FPGA, with system parameters set to the constant values: $\{r, \epsilon, T_c, T_g\} = \{3, 0, 7, 2\}$ adopted from recent literature [20]–[22] and are used throughout our experiments. The metrics used to evaluate the proposed methodology are; error (nonocc, all, disc) (Table II), frames-per-second (FPS), millions of disparity estimations per second (MDE/s), cycles per frame (CPF) and hardware resource utilization (LUTs, slice registers, DSPs, BRAMs).

The performance of the system designed based on the proposed methodology has been evaluated against the standard Middlebury benchmark that is widely used in evaluating the quality of the stereo-matching algorithms. The four image pairs viz. *Tsukuba*, *Venus*, *Teddy* and *Cones* are processed by the proposed system and the results are shown in Fig. 8(a-l). A performance comparison of the proposed system with state-of-the-art designs in terms of error (i.e., the percentage of bad matching pixels when compared to the ground truth) is listed in Table II. From this table, it can be observed that the error of the proposed methodology is 5.71% in non-occluded regions, 9.89% across the whole disparity map, and 13.58%

in discontinuous regions, resulting in an accuracy of 90.27% (calculated as $100 - \text{average bad pixel percentage}$). To further validate the proposed methodology rectified real-time stereo images, captured using a stereo camera, are processed. The corresponding results are shown in Fig. 8(o,p). It can be seen that the disparity is sharp for clear edges.

The proposed methodology is verified by using different wavelets (db2, sym2, coif1, dmey) and without applying the DWT. The disparity maps obtained for the Tsukuba stereo pair are shown in Fig. 9a and the corresponding overall error graph is shown in Fig. 9b. The error for all the wavelets vary from 6.72% to 7.9%, which shows that any of the wavelets can be used for the proposed methodology as they are in acceptable range as per Table II. Since hardware complexity should also be considered, the Haar wavelet has been used in this paper as proof of concept as it has a low-complexity hardware implementation, using a simple adder and shifter.

The proposed algorithm is verified for different sizes of median filter (from 3 to 11 with step-size of 2) and the disparity maps are shown in Fig. 9d and the corresponding error of the obtained disparity maps are shown in Fig. 9c. Disparities obtained from the median filter with a smaller window size have a lower error compared to large sizes due to over smoothing. The complexity of the median filter increases as the window size of the median filter increases. Hence the median filter with a window size of 3x3 has been chosen for the implementation. The disparity map is obtained in two different cases, firstly by applying DWT in both dimension of the image (2D-DWT) and secondly by taking the second level approximate coefficients which are shown in Fig. 8(m,n) for the stereo image Tsukuba. In both cases, the computational complexity has been reduced, since the number of input samples is reduced by 75%, but at the cost of higher error rate

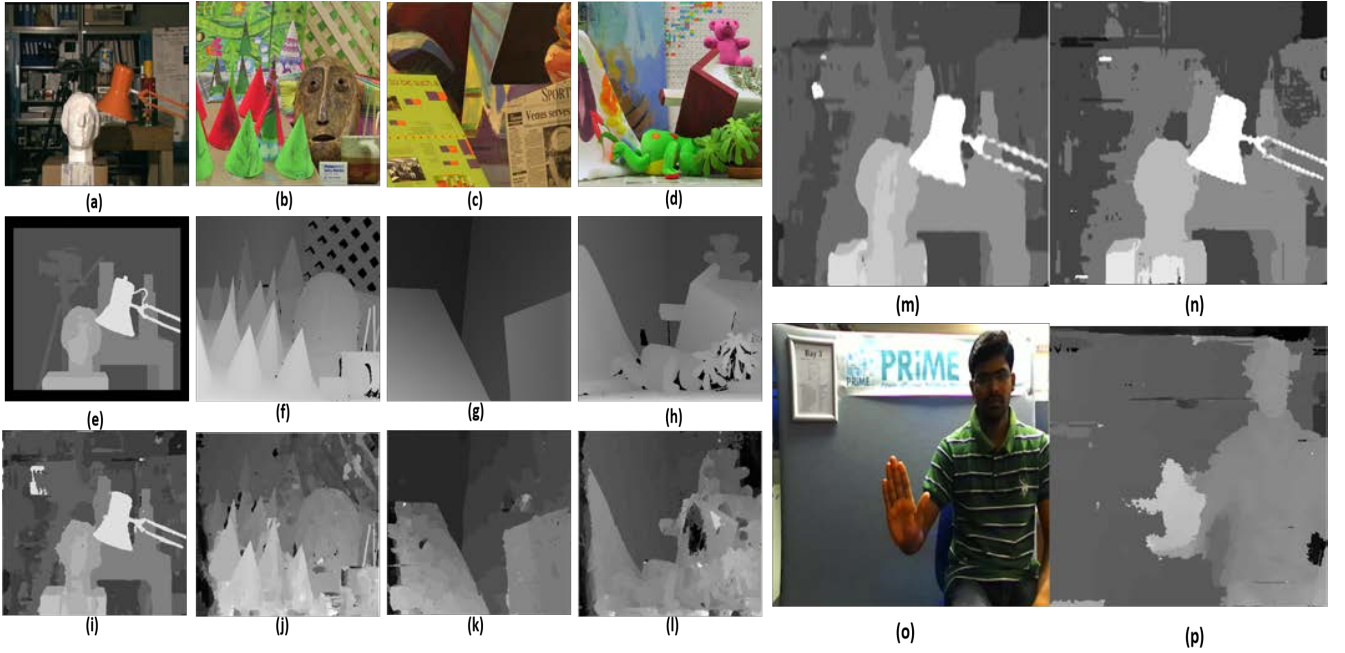


Fig. 8. Left images: (a) Tsukuba (b) Cones (c) Venus (d) Teddy; (e)-(h) Ground truth, (i)-(l) Outcome from the proposed methodology, Obtained disparity by applying (m) 2nd level DWT, (n) 2D DWT, (o) Real time left stereo image (p) Obtained disparity using proposed method.

TABLE II

COMPARISON OF ACCURACY USING THE MIDDLEBURY BENCHMARK. NONOCC: AVERAGE PERCENTAGE OF BAD PIXELS IN NON-OCCLUDED REGIONS; ALL: AVERAGE PERCENTAGE OF BAD PIXELS IN ALL REGIONS; DISC: AVERAGE PERCENTAGE OF BAD PIXELS IN DISCONTINUOUS REGIONS.

	Platform	Tsukuba			Venus			Teddy			Cones			Average percent bad pixel
		nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	
ADCensus [33]	GPU	1.07	1.48	5.73	0.09	0.25	1.15	4.10	6.22	10.9	2.42	7.25	6.95	3.97
Wenqiang [9]	FPGA	2.39	3.27	8.87	0.38	0.89	1.92	6.08	12.1	15.4	2.12	7.74	6.19	5.61
Jin et al. [23]	FPGA	1.66	2.17	7.64	0.40	0.60	1.95	6.79	12.4	17.1	3.34	8.97	9.62	6.05
Christos [21]	FPGA	2.38	3.01	9.38	0.40	0.7	3.62	7.23	12.7	17.2	2.87	8.59	8.27	6.36
Chang et al. [14]	DSP	-	2.80	-	-	0.64	-	-	13.7	-	-	10.1	-	6.81
SemiGlobal [34]	CPU	3.26	3.96	12.8	1.00	1.57	11.3	6.02	12.2	16.3	3.06	9.75	8.90	7.50
Banz et al. [15]	FPGA	4.1	-	-	2.7	-	-	11.4	-	-	8.4	-	-	noncc. = 6.7
MCADSR [35]	FPGA	3.62	4.15	14.0	0.48	0.87	2.79	7.54	14.7	19.4	3.51	11.1	9.64	7.65
Zhang et al. [25]	FPGA	3.84	4.34	14.2	1.20	1.68	5.62	7.17	12.6	17.4	5.41	11.0	13.9	8.20
Tree structured DP [36]	FPGA	1.43	2.51	6.60	2.37	2.97	13.1	8.11	13.6	15.5	8.12	13.8	16.4	8.71
Proposed	FPGA	4.5	6.72	11.34	2.3	3.88	9.8	8.61	14.2	18.04	7.28	14.67	15.07	9.73
S.perri [10]	FPGA	11.8	11.4	39.1	7.02	5.49	15.5	18.9	8.09	21	18	4.79	13.54	14.55
Jin et al. [5]	FPGA	9.79	11.6	20.3	3.59	5.27	36.8	12.5	21.5	30.6	7.34	17.6	21.0	17.2
Shan et al. [37]	FPGA	-	24.5	-	-	15.7	-	-	15.1	-	-	14.1	-	all = 17.3

of 26% for 2D-DWT and 22.6% for the second level DWT.

Fig. 10a shows a comparison of the proposed methodology with the state-of-the-art methodology of [20] in terms of computational complexity (i.e., number of pixels processed at every stage of algorithm). The x-axis shows the different stages of algorithm - DWT, CVC, GIF, WTA, IDWT and post processing as described in Section III-B, and the y-axis shows the number of pixels processed at every stage (where 1 is equivalent to size of an input image). The computational complexity of the proposed methodology is significantly reduced due to the reduction in the number of samples at the initial stage of the algorithm due to the application of the DWT. Fig. 10b gives the comparison of the latest designs [9], [22], [23], [35] with the proposed methodology in terms of the number of cycles needed for computing the disparity of a single frame

($CPF = Frequency/FPS$). The performance of proposed methodology does not degrade with the increase in image size.

Table III gives the comparison of the proposed methodology with related works in terms of quality (error), performance and hardware resource utilization. The resource utilization includes post-processing after FPGA-based optimizations. LUT utilization of proposed system is the least of all the approaches at the expense of 3.1 higher error on average compared to state-of-the-art designs [9], [21], [22], [35]. The error incurred is attributed to the application of the DWT and computing the approximate coefficients from the first resolution level of the DWT. Table IV provides the FPGA resource utilization for the system designed based on the proposed methodology in terms of LUTs, Registers, DSPs and BRAMs where core-{16, 32, 64} is the maximum disparity range number. Module-

TABLE III
QUALITY, SPEED AND ERROR COMPARISON WITH THE RELATED WORKS.(N.A = NOT APPLICABLE, N.M = NOT MENTIONED)

Work	Image	D_{max}	Speed (fps)	MDE/s (10^6)	Error	platform	LUT's	Slice Registers
ADSW [38]	320*240	30	0.01	0.0263	6.53	CPU	N.A	N.A
Chang [14]	352*288	64	42.5	272.5	N.A	ASIC	N.A	N.A
Jin [5]	640*480	64	230	4522	16.5	FPGA	114214	N.M
Ambrosch [6]	750*400	60	60	1080	12.5	FPGA	139606	N.M
Banz [15]	640*480	128	103	4050	6.7	FPGA	68427	N.M
Ding [11]	640*480	60	51	940	11.9	FPGA	50585	35020
Hosni [20]	640*480	26	25	200	5.57	GPU	N.A	N.A
Zhang [25]	1024*768	64	60	3019	8.20	FPGA	53095	74109
MCADSR [35]	1024*768	128	129	13076	7.65	FPGA	60160	33291
Perri [10]	640*480	60	45	829	10.09	FPGA	N.M	80270
Jin [23]	1024*768	60	199.3	9362	6.05	FPGA	122900	N.M
Ttofis [13]	640*480	64	60	1179	9.79	FPGA	88791	117260
Christos [22]	1280*720	64	60	3538	6.80	FPGA	57492	71192
Wenqiang [9]	1024*768	96	67.8	10472	5.61	FPGA	125255	81092
Proposed	1280*720	64	103	6075	9.73	FPGA	34181	47368

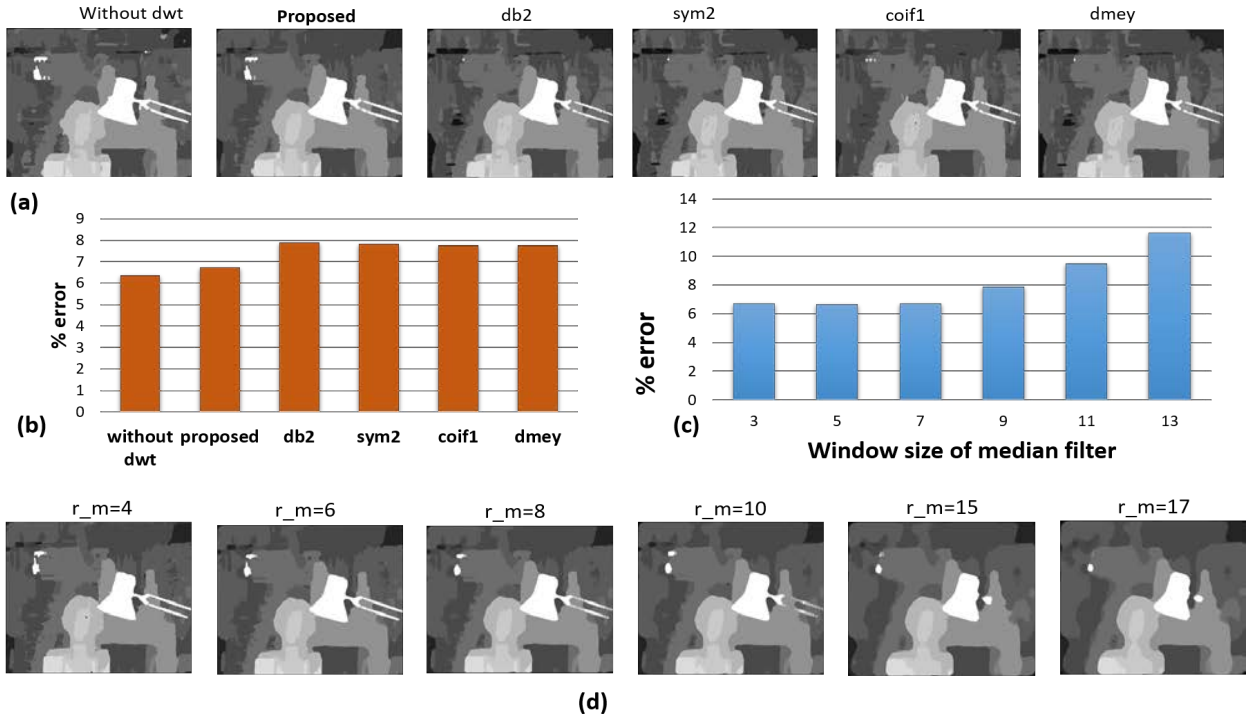
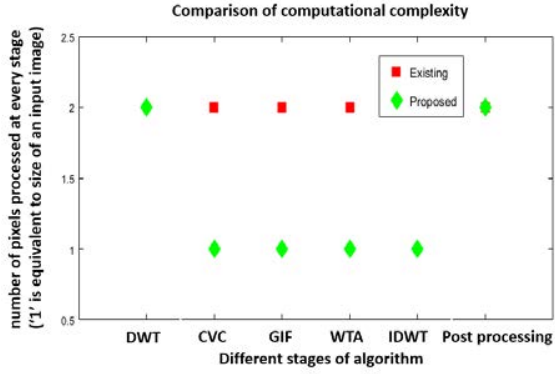


Fig. 9. (a) Obtained disparity for different wavelets where, 'db2' belongs to Daubechies family of wavelets, 'sym2' belongs to Symlets family of wavelets, 'coif1' belongs to Coiflets family of wavelets, 'dmey' belongs to Symlets family of wavelets, (b) Obtained disparity error for different wavelets, (c) Obtained error for Tsukuba image for different size of median filter (x-axis, y-axis represents window size of median filter and error when compared with ground-truth disparity respectively), (d) Error analysis by varying median filter widow size r_m over the range $\{3, 5, 7, 9, 11, 13\}$.

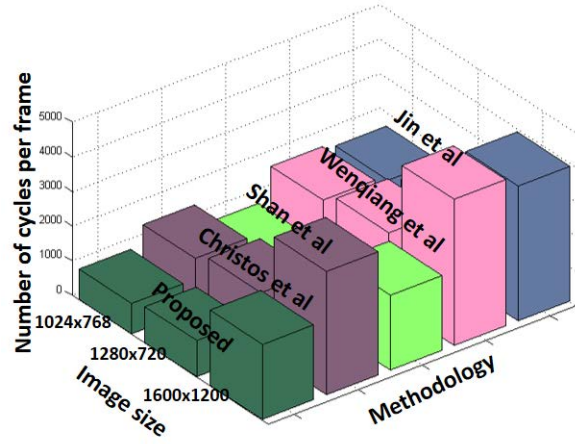
wise resource utilization for the proposed design is shown in Table V. Table VI gives a comparison of the proposed methodology with state-of-the-art designs in terms of CPF. This table shows that a 44.4% improvement is achieved by the proposed methodology as compared to the design of [23] for all the image resolutions. For the image size 1280×720 , the proposed methodology yields about 39.5% better performance in terms of CPF when compared with [22] because of the reduction in the number of samples.

A comparison of the system designed in terms of frequency, power and FPS is given in Table VII. A 52% improvement in

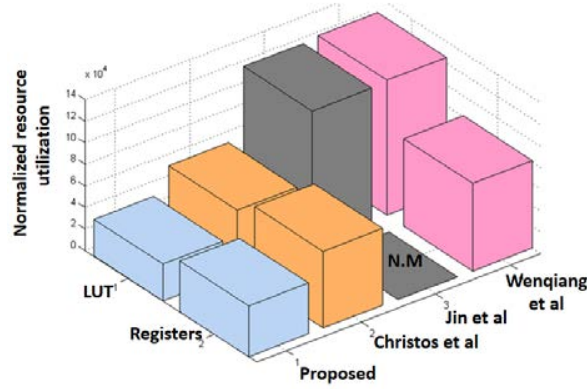
FPS is observed when the proposed methodology is compared with one of the recently reported design [9]. In particular LUT utilization is reduced by 61.5%, 40.5%, 43.2%, 72.7%, 35.6% when compared to latest reported results in [9], [13], [22], [25], [35] respectively. Slice registers utilization is reduced by 59.6%, 33.46%, 41.6%, 36% when compared to [9], [13], [22], [25] respectively. A detailed comparison is given in Table VIII and Fig. 10c.



(a) Comparison of computational complexity



(b) State-of-the-art comparison for cycles-per-frame



(c) State-of-the-art comparison for resource utilization

Fig. 10. Jin *et. al.* [23], Christos *et. al.* [22], Wenqiang *et. al.* [9]. (a) Comparison of computational complexity between the proposed design and [20] (b) Comparison to [9], [22], [23], [35] in terms of CPF and (c) Comparison in terms of FPGA resource utilization LUTs, Registers (N.M=Not mentioned).

TABLE IV
FPGA RESOURCE UTILIZATION BY THE SYSTEM DESIGNED BASED ON THE PROPOSED METHODOLOGY.

Design	LUT	Slice register	DSP 48E	BRAM
Stereo Matching core-16	8709	12561	68	56
Stereo Matching core-32	18256	24157	129	120
Stereo Matching core-64	34181	47368	273	247

TABLE V
MODULE-WISE RESOURCE UTILIZATION

Module Name	LUT	Slice register	DSP 48E	BRAM
DWT	25	42	0	0
CVC	9328	5320	0	0
GIF	21053	39953	273	228
WTA	494	275	0	0
LR-Check Filling	515	720	0	0
Median filter	366	328	0	0

V. CONCLUSION

In this paper we introduced a DWT based methodology into a state-of-the-art disparity estimation algorithm, resulting in a significant performance improvement in terms of speed

TABLE VI
NUMBER OF CYCLES PER FRAME COMPARISON ($\times 10^3$)

Methodology/Image Size	1024x768	1280x720	1600x1200
Jin 2014 [23]	1595	1869	3895
Shan 2014 [35]	790	926	2170
Wang 2015 [9]	1730	2027	4224
Christos 2016 [22]	1464	1716	3576
Proposed	886	1038	2164

TABLE VII
FPS, FREQUENCY AND POWER COMPARISON (N.M= NOT MENTIONED)

	Image Size	Frequency (MHz)	Power (W)	FPS
Jin 2014 [23]	1024x768	318	10.6	199.3
Shan 2014 [35]	1024x768	102	N.M	129
Wang 2015 [9]	1024x768	180	N.M	67.8
Christos 2016 [22]	1280x720	103	2.8	60
proposed	1280x720	107	2.1	103

and computational complexity. The performance of a system designed from the proposed methodology has been evaluated against standard Middlebury benchmarks that are widely used in evaluating the quality of stereo matching algorithms.

TABLE VIII
HARDWARE RESOURCE UTILIZATION COMPARISON WITH
STATE-OF-THE-ART DESIGNS (N.M= NOT MENTIONED)

Work	Image Size	LUT	Slice Register
Jin 2014 [23]	1024x768	122900	N.M
Shan 2014 [35]	1024x768	60160	33291
Wang 2015 [9]	1024x768	125255	81092
Christos 2016 [22]	1280x720	57492	71192
Proposed	1280x720	34181	47368

The four image pairs of *Tsukuba*, *Venus*, *Teddy* and *Cones* have been used to test the proposed system. It has been demonstrated that the system achieves an improvement of 44.4% cycles per frame, 52% frames per second and 61.5% and 59.6% LUT and registers utilization respectively on an FPGA compared with state-of-the-art designs. We believe that our system has significant impact for applications in autonomous vehicles and mobile robotics by meeting real-time processing requirements in a resource-constrained scenario.

REFERENCES

- [1] B. Cyganek and J. P. Siebert, *An introduction to 3D computer vision techniques and algorithms*. John Wiley & Sons, 2011.
- [2] B. Tippetts, D. J. Lee, K. Lillywhite, and J. Archibald, "Review of stereo vision algorithms and their suitability for resource-limited systems," *Journal of Real-Time Image Processing*, vol. 11, no. 1, pp. 5–25, 2016.
- [3] R. A. Hamzah and H. Ibrahim, "Literature survey on stereo vision disparity map algorithms," *Journal of Sensors*, 2016.
- [4] R. Kalarot, J. Morris, and G. Gimel'farb, "Performance analysis of multi-resolution symmetric dynamic programming stereo on gpu," in *Image and Vision Computing New Zealand (IVCNZ), 2010 25th International Conference of*. IEEE, 2010, pp. 1–7.
- [5] S. Jin, J. Cho, X. Dai Pham, K. M. Lee, S.-K. Park, M. Kim, and J. W. Jeon, "Fpga design and implementation of a real-time stereo vision system," *IEEE transactions on circuits and systems for video technology*, vol. 20, no. 1, pp. 15–26, 2010.
- [6] K. Ambrosch and W. Kubinger, "Accurate hardware-based stereo vision," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1303–1316, 2010.
- [7] N. Baha and S. Larabi, "Accurate real-time neural disparity map estimation with fpga," *Pattern Recognition*, vol. 45, no. 3, pp. 1195–1204, 2012.
- [8] S. K. Gehrig, F. Eberli, and T. Meyer, "A real-time low-power stereo vision engine using semi-global matching," in *International Conference on Computer Vision Systems*. Springer, 2009, pp. 134–143.
- [9] W. Wang, J. Yan, N. Xu, Y. Wang, and F.-H. Hsu, "Real-time high-quality stereo vision system in fpga," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 10, pp. 1696–1708, 2015.
- [10] S. Perri, P. Corsonello, and G. Cocorullo, "Adaptive census transform: A novel hardware-oriented stereovision algorithm," *Computer Vision and Image Understanding*, vol. 117, no. 1, pp. 29–41, 2013.
- [11] J. Ding, J. Liu, W. Zhou, H. Yu, Y. Wang, and X. Gong, "Real-time stereo vision system using adaptive weight cost aggregation approach," *EURASIP Journal on Image and Video Processing*, vol. 2011, no. 1, p. 1, 2011.
- [12] C. Colodro-Conde, F. J. Toledo-Moreo, R. Toledo-Moreo, J. J. Martínez-Álvarez, J. G. Guerrero, and J. M. Ferrández-Vicente, "Evaluation of stereo correspondence algorithms and their implementation on fpga," *Journal of Systems Architecture*, vol. 60, no. 1, pp. 22–31, 2014.
- [13] C. Ttofis, C. Kyrkou, and T. Theoharides, "A hardware-efficient architecture for accurate real-time disparity map estimation," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 2, p. 36, 2015.
- [14] N. Y.-C. Chang, T.-H. Tsai, B.-H. Hsu, Y.-C. Chen, and T.-S. Chang, "Algorithm and architecture of disparity estimation with mini-census adaptive support weight," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 6, pp. 792–805, 2010.
- [15] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and fpga-implementation," in *Embedded Computer Systems (SAMOS), 2010 International Conference on*. IEEE, 2010, pp. 93–101.
- [16] C. Prabhakar and K. Jyothi, "A wavelet-based multiresolution approach to stereo matching of face images," in *Advanced Communication Control and Computing Technologies (ICACCCT), 2012 IEEE International Conference on*. IEEE, 2012, pp. 316–320.
- [17] T. Gao, "3-d video based disparity estimation and object segmentation," *Global Applications of Pervasive and Ubiquitous Computing*, p. 194, 2012.
- [18] S. Kumar, S. Kumar, N. Sukavanam, and B. Raman, "Human visual system and segment-based disparity estimation," *AEU-International Journal of Electronics and Communications*, vol. 67, no. 5, pp. 372–381, 2013.
- [19] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 6, pp. 1397–1409, 2013.
- [20] A. Hosni, M. Bleyer, C. Rhemann, M. Gelautz, and C. Rother, "Real-time local stereo matching using guided image filtering," in *2011 IEEE International Conference on Multimedia and Expo*. IEEE, 2011, pp. 1–6.
- [21] C. Ttofis and T. Theoharides, "High-quality real-time hardware stereo matching based on guided image filtering," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 356.
- [22] C. Ttofis, C. Kyrkou, and T. Theoharides, "A low-cost real-time embedded stereo vision system for accurate disparity estimation based on guided image filtering," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2678–2693, 2016.
- [23] M. Jin and T. Maruyama, "Fast and accurate stereo vision system on fpga," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 1, p. 3, 2014.
- [24] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1–3, pp. 7–42, 2002.
- [25] L. Zhang, K. Zhang, T. S. Chang, G. Lafruit, G. K. Kuzmanov, and D. Verkest, "Real-time high-definition stereo matching on fpga," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2011, pp. 55–64.
- [26] A. Acharyya, K. Maharatna, B. M. Al-Hashimi, and S. R. Gunn, "Memory reduction methodology for distributed-arithmetic-based dwt/idwt exploiting data symmetry," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 4, pp. 285–289, 2009.
- [27] C. Zhang, C. Wang, and M. O. Ahmad, "A pipeline vlsi architecture for high-speed computation of the 1-d discrete wavelet transform," *IEEE transactions on circuits and systems i: regular papers*, vol. 57, no. 10, pp. 2729–2740, 2010.
- [28] M. Martina and G. Masera, "Low-complexity, efficient 9/7 wavelet filters vlsi implementation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 11, pp. 1289–1293, 2006.
- [29] K. G. Oweiss, A. Mason, Y. Suhail, A. M. Kamboh, and K. E. Thomson, "A scalable wavelet transform vlsi architecture for real-time signal processing in high-density intra-cortical implants," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 6, pp. 1266–1278, 2007.
- [30] M. Martina and G. Masera, "Multiplierless, folded 9/7–5/3 wavelet vlsi architecture," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 9, pp. 770–774, 2007.
- [31] A. Sanny and V. K. Prasanna, "Energy-efficient median filter on fpga," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*. IEEE, 2013, pp. 1–8.
- [32] D. Prokin and M. Prokin, "Low hardware complexity pipelined rank filter," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 6, pp. 446–450, 2010.
- [33] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On building an accurate stereo matching system on graphics hardware," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, 2011, pp. 467–474.
- [34] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, 2008.
- [35] Y. Shan, Y. Hao, W. Wang, Y. Wang, X. Chen, H. Yang, and W. Luk, "Hardware acceleration for an accurate stereo vision system using mini-census adaptive support region," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4s, p. 132, 2014.

- [36] M. Jin and T. Maruyama, "A real-time stereo vision system using a tree-structured dynamic programming on fpga," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 21–24.
- [37] Y. Shan, Z. Wang, W. Wang, Y. Hao, Y. Wang, K. Tsoi, W. Luk, and H. Yang, "Fpga based memory efficient high resolution stereo vision system for video tolling," in *Field-Programmable Technology (FPT), 2012 International Conference on*. IEEE, 2012, pp. 29–32.
- [38] K.-J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 650–656, 2006.



Charan Kumar Vala received his masters degree in microelectronics and very large scale integration (VLSI) from Indian Institute of Technology Hyderabad, INDIA. Currently he is a research scholar at School of Electronics and Computer Science, University of Southampton, U.K. His research interests include VLSI architectures, FPGA implementation, VLSI for Cyber physical systems, low-power design techniques, and analog/mixed-signal ASICs. He is runner-up in all India Cadence Circuit Design Contest.



Koushik Immadisetty received the B.Tech in Electrical Engineering from Indian Institute of Technology Hyderabad, Hyderabad, India, in 2017. His research interests include Efficient algorithms and architectures for image/video processing, stereo vision, VLSI architectures, low-power design techniques. He is currently working in Qualcomm Innovation Center Inc., India as a associate engineer.



Amit Acharyya (M11) received the Ph.D. degree from the School of Electronics and Computer Science, University of Southampton, U.K., in 2011. He is currently an Assistant Professor with IIT Hyderabad, Hyderabad, India. His research interests include signal processing algorithms, VLSI architectures, low power design techniques, computer arithmetic, numerical analysis, linear algebra, bio-informatics, and electronic aspects of pervasive computing.



Charles Leech is a senior research assistant for the PRiME project at the University of Southampton where he received his 1st class hon BEng Electronic Engineering degree. His work focuses on power and performance optimisation of computer vision applications on heterogeneous embedded systems. Additionally, he is involved in the development of software frameworks for runtime management of many-core systems. He has interests in approximate computing, stereo vision, and machine learning for embedded devices.



Vibishna Balagopal is a Senior Electronics Design Engineer at LumiraDx Technology, UK. She was awarded a first-class honors degree in Electronics and Communication Engineering from University of Calicut, India in 2008 and completed her Post Graduation Diploma in VLSI design in 2009 from National Institute of Electronics and IT, India. Currently, she is working in digital system design in embedded/ FPGA platforms for Point-Of-Care medical devices. Her research interests include run-time energy management in many/ multi-core Embedded systems. Previously, she was working for the design and development of beam steering controller for AESA Radar application. She was working as Senior Research Assistant at University of Southampton UK, Scientist at Electronics and Radar Development Establishment - DRDO, India and Project Engineer at Wipro Technologies, India.



Geoff V Merrett (GSM06-M09) received the B.Eng. degree (Hons.) in electronic engineering and the Ph.D. degree from the University of Southampton, Southampton, U.K., in 2004 and 2009, respectively.

He is currently an Associate Professor in electronic systems with the University of Southampton. His current research interests include low-power and energy harvesting aspects of embedded & mobile systems. He has published over 100 articles in journals/conferences in the above areas.

Dr. Merrett was the General Chair of the Energy Neutral Sensing Systems Workshop from 2013 to 2015. He is a fellow of the The Higher Education Academy.



Bashir M. Al-Hashimi (M99-SM01-F09) is an ARM Professor of Computer Engineering, Dean of the Faculty of Physical Sciences and Engineering, and the Co-Director of the ARM-ECS Research Centre, University of Southampton, Southampton, U.K. He has published over 380 technical papers. His current research interests include methods, algorithms, and design automation tools for low-power design and test of embedded computing systems. He has authored or co-authored five books and has graduated 35 Ph.D. students.

Energy Efficient Multi-Core Processing

Charles Leech and Tom J. Kazmierski

(Invited Paper)

Abstract—This paper evaluates the present state of the art of energy-efficient embedded processor design techniques and demonstrates, how small, variable-architecture embedded processors may exploit a run-time minimal architectural synthesis technique to achieve greater energy and area efficiency whilst maintaining performance. The picoMIPS architecture is presented, inspired by the MIPS, as an example of a minimal and energy efficient processor. The picoMIPS is a variable-architecture RISC microprocessor with an application-specific minimised instruction set. Each implementation will contain only the necessary datapath elements in order to maximise area efficiency. Due to the relationship between logic gate count and power consumption, energy efficiency is also maximised in the processor therefore the system is designed to perform a specific task in the most efficient processor-based form. The principles of the picoMIPS processor are illustrated with an example of the discrete cosine transform (DCT) and inverse DCT (IDCT) algorithms implemented in a multi-core context to demonstrate the concept of minimal architecture synthesis and how it can be used to produce an application specific, energy efficient processor.

Index Terms—Embedded processors, application specific architectures, MIPS architecture, digital synthesis, energy efficient design, low power design.

Review Paper

DOI: 10.7251/ELS1418003L

I. INTRODUCTION

THE energy efficiency of embedded processors is essential in mobile electronics where devices are powered by batteries. Processor performance has been increasing over the last few decades at a rate faster than the developments in battery technologies. This has led to a significant reduction of the battery life in mobile devices from days to hours. Also, new mobile applications demand higher performance and more graphically intensive processing. These demands are currently being addressed by many-core, high-frequency architectures which can deliver high-speed processing necessary to meet the tight execution deadlines. These two contradictory demands, the need to save energy and the requirement to deliver outstanding performance must be addressed by entirely new approaches. A number of research directions have appeared. Heterogeneous and reconfigurable embedded many-core systems can improve energy efficiency while maintaining high speed through judicious task scheduling and hardware adaptability. In a heterogeneous system, such as the ARM big.LITTLE architecture [1] smaller cores are employed to process simple and less demanding tasks to save energy while larger cores

handle high performance and energy hungry processing when necessary. Reconfigurable architectures use flexible interconnect, power gating and software control within each core, thus achieving heterogeneity within the core. Reconfigurable cores can be configured in this way as either slower, but energy efficient processors, or faster, high-performance cores. A number of approaches have been proposed to save energy within a core. Dynamic Voltage and Frequency Scaling (DVFS) [2] is a popular and well established technique where the supply voltage and the clock frequency are scaled to trade energy for performance and vice-versa. DVFS is typically implemented by including voltage regulators and phase-lock loop controlled clocks in the processor. The architecture is modified to allow the operating system to select a desired voltage and frequency through writing data to a DVFS control register. At any desired performance level, the operating system will put the processor into a minimum energy consumption mode. DVFS has proved very effective especially in applications where high performance is peaking only during a small fraction of the operating time as significant energy savings are achieved. Many other energy saving design techniques are currently being explored at the circuit, architecture and even system level. For example supply voltage in bus drivers can be reduced to extremely low levels to reduce bus energy consumption. New SRAM designs are being developed where energy consumption is reduced to extremely low levels in both the on-chip caches and the external memories. The architecture of processor cores are traditionally determined from a compromise of speed, power consumption, scalability, maintainability and extensibility. However, applications have different characteristics that require specific hardware implementations to enable optimal performance and therefore a system should be able to adapt its architecture to each application scenario. In this paper, we aim to demonstrate, through the evaluation of present technology, how small, variable-architecture embedded processors may exploit a run-time minimal architectural synthesis technique to achieve greater energy and area efficiency whilst maintaining performance.

II. OVERVIEW OF ENERGY EFFICIENCY TECHNOLOGIES

This section presents the current state of research in energy efficient technologies in multi-core systems for both traditional power saving techniques and novel technologies including heterogeneous and reconfigurable architectures. Through the analysis of present technology, we aim to demonstrate how a greater performance, energy efficiency and area efficiency balance can be achieved.

The introduction of multi-core structures to processor architectures has caused a significant increase in the power

Manuscript received 29 May 2014. Accepted for publication 12 June 2014.

C. Leech and T. J. Kazmierski are with the University of Southampton, UK (e-mail: {cl19g10, tj.k}@ecs.soton.ac.uk).

consumption of these systems. In addition, the gap between average power and peak power has widened as the level of core integration increases [3]. A global power manager policy, such as that proposed by Isci et al, that has per-core control of parameters such as voltage and frequency levels is required in order to provide effective dynamic control of the power consumption [3]. Metrics such as performance-per-watt [4], [5], average and peak power or energy-delay product [6] are all used to quantify the power or energy efficiency of a system in order to evaluate it, however properties are prioritised differently depending on the application requirements. Modelling and simulation of multi-core processors is also an important process in order to better understand the complex interactions that occur inside a system and cause power and energy consumption [7]–[11]. For example, the model created by Basmadjian et al is tailored for multi-core architectures in that it accounts for resource sharing and power saving mechanisms [8].

A. Energy Efficiency techniques in Static Homogeneous Multi-core Architectures

Many energy efficiency and power saving technologies are already integrated into processor architectures in order to reduce power dissipation and extend battery life, especially in mobile devices. A combination of technologies is most commonly implemented to achieve the best energy efficiency whilst still allowing the system to meet performance targets [4]. Techniques to increase energy efficiency can be applied at many development levels from architecture co-design and code compilation to task scheduling, run-time management and application design [12]. A summary and analysis of these technologies is presented in the following section.

1) *Dynamic Voltage and Frequency Scaling*: Dynamic Voltage and Frequency Scaling (DVFS) is a technique used to control the power consumption of a processor through fine adjustment of the clock frequency and supply voltage levels [3], [4], [12], [13]. High levels are used when meeting performance targets is a priority and low levels (known as CPU throttling) are used when energy efficiency is most important or high performance is not required. When the supply voltage is lowered and the frequency reduced, the execution of instructions by the processor is slower but performed more energy efficiently due to the extension of delays in the pipeline stages. The rise and fall times for logic circuitry is increased along with the clock period meaning performance targets for applications must be relaxed. DVFS can be used in homogeneous multi-core architectures to emulate heterogeneity by controlling the frequency and supply to each core individually [14]. Each core therefore appears as though it has different delay properties however the architectures are still essentially identical. This per-core DVFS mechanism is investigated by Wonyoung et al who conclude that significant energy saving opportunities exist where on-chip integrated voltage regulators are used to provide nanosecond scale voltage switching [13]. DVFS can also be combined with thread migration to reduce energy consumption [4], [15]. Cai et al cite the problem that present DVFS energy saving techniques on multi-core systems assume one

hardware context for each core whereas simultaneous multi-threading (SMT) is commonly implemented which causes these techniques to be less effective. Their novel technique, known as thread shuffling, uses concepts of thread criticality and thread criticality degree instead of thread-delay to maps together threads with similar criticality degree. This accounts for SMT when implementing DVFS and thread migration and achieves energy savings of up to 56% without impeding performance at all.

2) *Clock Gating and Clock Distribution*: Clock gating is a process, applied at the architectural design phase, to insert additional logic between the clock source and clock input of the processor's circuitry. During program execution, it reduces power consumption by logically disconnecting the clock of synchronous logic circuits to prevent unnecessary switching. Classed as a Dynamic Power Management (DPM) technique, as it is applied at run-time along with other techniques such as thread scheduling and DVFS to optimise the power/performance trade-off of a system [12]. The clock gating and distribution techniques implemented by Qualcomm in the Hexagon processor are analysed by Bassett et al on their ability to improve the energy efficiency of a digital signal processors (DSP) [16]. A low power clock network is implemented using multi-level clock gating strategies and spine-based clock distribution. The 4 levels of clock gating allows different size regions of the chip to be deactivated, from entire cores down to single logic cells. Further power reduction is achieved through a structured clock tree that aims to minimise the power consumed in distributing the clock signal across the chip whilst avoiding clock skew and delay. The clock tree structure (CTS) examined by Bassett et al is tested to give a 2 time reduction in skew over traditional CTS while power tests show reductions in power consumption by 8% for high-activity and over 35% for idle mode. Large portions of the chip will spend the majority of their time in idle more therefore high efficiency in this mode is critical.

3) *Power Domains*: Power domains are regions of a system or processor that are controlled from a single supply and can be completely powered down in order to minimise power consumption without entirely removing the power supply to the system. Power domains can be used dynamically and when used in conjunction with clock gating, lead to further improvements in energy efficiency. The ARM Cortex-A15 MPCore processor supports multiple power domains both for the core and for the surrounding logic [17]. Figure 1 shows these domains, labelled Processor and Non-Processor, that allow large parts of the processor to be deactivated. Smaller internal domains, such as CK_GCLKCR, are implemented to allow smaller sections to be deactivated for finer performance and power variations.

Power domains are often coupled with power modes as a means of switching on or off several power domains in order to enter low power, idle or shutdown states. The Cortex-A15 features multiple power modes with specific power domain configurations such as Dormant mode, where some Debug and L2 cache logic is powered down but the L2 cache RAMs are powered up to retain their state, or Powerdown mode where all power domains are powered down and the processor state

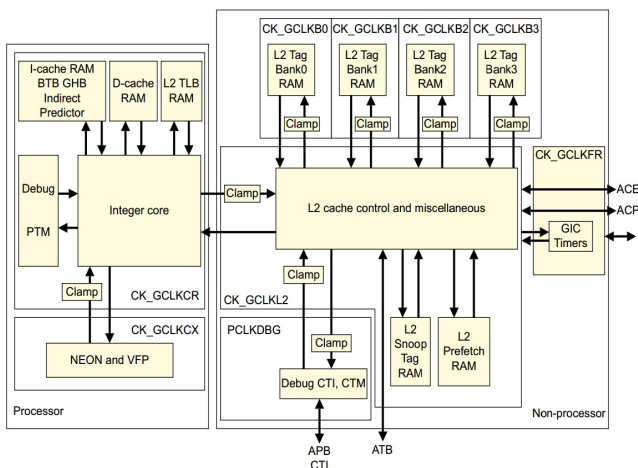


Fig. 1. The ARM Cortex-A15 features multiple power domains for the core and surrounding logic, reprinted from [17].

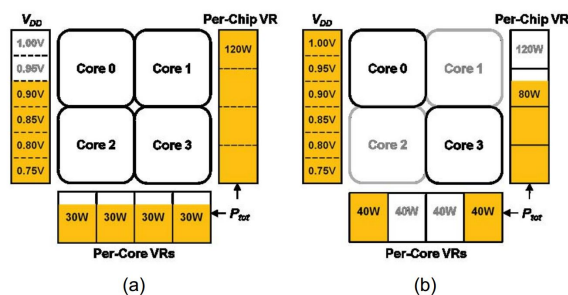


Fig. 2. Per-core power domains can provide reduce power consumption and higher performance levels, reprinted from [18].

is lost [17]. In multi-core architectures, power domains can be used to power down individual cores when idle or during non-parallel tasks in order to manage power consumption. Sinkar et al [18] and Ghasemi et al [19] present low-cost, per-core voltage domain solutions in order to improve performance in power-constrained processors. Figure 2 shows how Sinkar's mechanism can provide reduced chip power consumption but maintain per-core performance. In (a), a chip-wide power domain is shown with all cores active at the same level. In contrast, (b) shows a per-core power domain which allows unnecessary cores to be powered down and active cores to provide a higher performance level while still reducing the overall chip power level.

In the work by Rotem et al [20], topologies containing multiple clock, voltage and frequency domains are investigated in order to build a high-end chip multiprocessor (CMP) considering both power constraints and performance demands. These domains are controlled using DVFS techniques in connection with clock gating and are exercised by simulating power supply constraints. Results showed that multiple power domains can be beneficial for fully threaded applications whereas a single power domain is more suitable for light-threaded workloads. Power domains can be linked to DVFS techniques if the domain contains multiple voltage levels. Per-core power

domains therefore enables per-core DVFS control such that each core can exploit run-time performance variations in multi-threaded applications [13], [18], [20].

4) **Pipeline Balancing:** Pipeline balancing (PLB) is a technique used to dynamically adjust the resources of the pipeline of a processor such that it retains performance while reducing power consumption [21]. The delay constraints on microarchitectural pipeline stages can be modified in order to make them more power efficient, in a similar way to DVFS, when the performance demand of the application is relaxed [22]. In work by Bahar et al, PLB operates in response to instruction per cycle (IPC) variations within a program [21]. The PLB mechanism dynamically reduces the issue width of the pipeline to save power or increases it to boost throughput. An 8-wide issue pipeline that has its unified issue queue divided into a left and right arbiter, separate left and right register files and functional units. A control unit is included can deactivate the right arbiter and its functional units to allow the pipeline to enter a low power mode. They show that this PLB technique can reduce power consumption of the issue queue and execution unit by up to 23% and 13% respectively with only an average performance loss of 1% to 2% [21]. Power Balanced pipelines is a concept in which the power disparity of pipeline stages is reduced by assigning different delays to each microarchitectural pipe stage while guaranteeing a certain level of performance/throughput [22]. In a similar fashion to [21], the concept uses cycle time stealing to redistribute cycle time from low power stages to stages that can perform more efficiently if given more time. Static power balancing is performed during design time to identify power heavy circuitry in pipe stages for which consumption remains fairly constant for different programs and reallocate cycle time accordingly. Dynamic power balancing is implemented on top of this to manage power fluctuations within each workload and further reduce the total power cost. Balancing of power rather than delay can result in a 46% power consumption reduction by the processor with no loss in throughput for a FabScalar processor over a baseline comparison. Power savings are also greater at lower frequencies.

5) *Caches and Interconnects*: It is not only the design of the processor's internal circuitry that is important in maintaining energy efficiency. Kumar et al conclude, from a paper examining the interconnections in multi-core architectures, that careful co-design of the interconnect, caches and the processor cores is required to achieve high performance and energy efficiency [23]. Through several theoretical examples, they examine parameters such as the area, power and bandwidth costs required to implement the processor-cache interconnect, showing that large caches sizes can constrain the interconnect's size and large interconnects can be power-hungry and inefficient. Zeng et al also recognise the high level of integration that is inherent in CMPs and attempt to reduce the interconnect power consumption by developing a novel cache coherence protocol [24]. Using their technique, results show that an average of 16.3% of L2 cache accesses could be optimised and as every access consumes time and power, an average 9.3% power reduction is recorded while increasing system performance by 1.4%.

B. Energy Efficiency techniques in Heterogeneous Multi-core Architectures

A heterogeneous or asymmetric multi-core architecture is composed of cores of varying size and complexity which are designed to compliment each other in terms of performance and energy efficiency [6]. A typical system will implement a small core to process simple tasks, in an energy efficient way, while a larger core provides higher performance processing for when computationally demanding tasks are presented. The cores represent different points in the power/performance design space and significant energy efficiency benefits can be achieved by dynamically allocating application execution to the most appropriate core [25]. A task matching or switching system is also implemented to intelligently assign tasks to cores; balancing a performance demand against maintaining system energy efficiency. These systems are particularly good at saving power whilst handling a diverse workload where fluctuations of high and low computational demand are common [26].

A heterogeneous architecture can be created in many different ways and many alternative have been developed due to the heavy research interest in this area. Modifications to general purpose processors, such as asymmetric core sizes [11], custom accelerators [27], varied caches sizes [14] and heterogeneity within each core [5], [28], have all been demonstrated to introduce heterogeneous features into a system.

One of the most prominent and successful heterogeneous architectures to date is the ARM big.LITTLE system. This is a production example of a heterogeneous multiprocessor system consisting of a compact and energy efficient “LITTLE” Cortex-A7 processor coupled with a higher performance “big” Cortex-A15 processor [26]. The system is designed with the dynamic usage patterns of modern smart phones in mind where there are typically periods of high intensity processing followed by longer periods of low intensity processing [29]. Low intensity tasks, such as texting and audio, can be handled by the A7 processor enabling a mobile device to save battery life. When a period of high intensity occurs, the A15 processor can be activated to increase the system’s throughput and meet tighter performance deadlines. A power saving of up to 70% is advertised for a light workload, where the A7 processor can handle all of the tasks, and a 50% saving for medium workloads where some tasks will require allocation to the A15 processor.

Kumar et al present an alternative implementation where two architectures from the Alpha family, the EV5 and EV6, are combined to be more energy and area efficient than a homogeneous equivalent [6], [25]. They demonstrate that a much higher throughput can be achieved due to the ability of a heterogeneous multi-core architecture to better exploit changes in thread-level parallelism as well as inter- and intra- thread diversity [6]. In [25], they evaluate the system in terms of its power efficiency indicating a 39% average energy reduction for only a 3% performance drop [25].

Composite Cores is a microarchitectural design that reduces the migration overhead of task switching by bringing heterogeneity inside each individual core [28]. The design

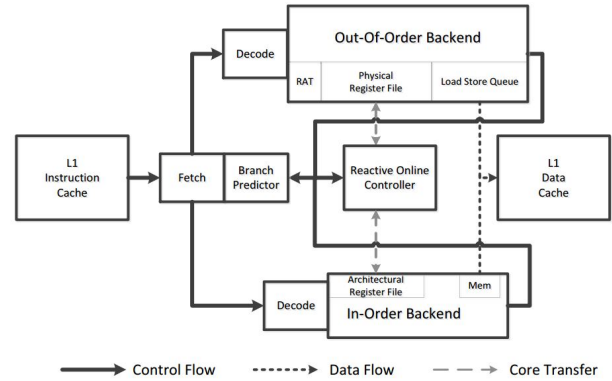


Fig. 3. The microarchitecture for Composite Cores, featuring two μ Engines, reprinted from [28].

contains 2 separate backend modules, called μ Engines, one of which features a deeper and more complex out-of-order pipeline, tailored for higher performance, while the other features a smaller, compact in-order pipeline designed with energy efficiency in mind. Figure Due to the high level of hardware resource sharing and the small μ Engine state, the migration overhead is brought down from the order of 20,000 instructions to 2000 instructions. This greatly reduces the energy expenditure associated with migration and also allows more of the task to be run in an efficient mode. Their results show that the system can achieve an energy saving of 18% using dynamic task migration whilst only suffering a 5% performance loss.

Using both a heterogeneous architecture and hardware re-configuration, a technique called Dynamic Core Morphing (DCM) is developed by Rodrigues et al to allow the shared hardware of a few tightly coupled cores to be morphed at runtime [5]. The cores all feature a baseline configuration but reconfiguration can trigger the re-assignment of high performance functional units to different cores to speed up execution. The efficiency of the system can lead to performance per watt gains of up to 43% and an average saving of 16% compared to a homogeneous static architecture.

The energy efficiency benefits of heterogeneity can only be exploited with the correct assignment of tasks or applications to each core [7], [10], [30]–[32]. Tasks must be assigned in order to maximise energy efficiency whilst ensuring performance deadlines are met. Awan et al perform scheduling in two phases to improve energy efficiency; task allocation to minimise active energy consumption and exchange of higher energy states to lower, more energy efficient sleep states [7]. Alternatively, Calcado et al propose division of tasks into m-threads to introduce fine-grain parallelism below thread level [33]. Moreover, Saha et al include power and temperature models into an adaptive task partitioning mechanism in order to allocate task according to their actual utilisations rather than based on a worst case execution time [10]. Simulation results confirm that the mechanism is effective in minimising energy consumption by 55% and reduces task migrations by 60% over

alternative task partitioning schemes.

Tasks assignment can also be performed in response to program phases which naturally occur during execution when the resource demands of the application change. Phase detection is used by Jooya and Analoui to dynamically re-assigning programs for each phase to improve the performance and power dissipation of heterogeneous multi-core processors [31]. Programs are profiled in dynamic time intervals in order to detect phase changes. Sawalha et al also propose an online scheduling technique that dynamically adjusts the program-to-core assignment as application behaviour changes between phases with an aim to maximise energy efficiency [32]. Simulated evaluation of the scheduler shows energy saving of 16% on average and up to 29% reductions in energy-delay product can be achieved as compared to static assignments.

C. Energy Efficiency techniques in Reconfigurable Multi-core Architectures

Reconfigurability is another property that has the potential to increase the energy and area efficiency of processors and systems on chip by introducing adaptability and hardware flexibility into the architecture. Building on the innovations that heterogeneous architectures bring, reconfigurable architectures aim to achieve both energy efficiency and high performance but within the same processor and therefore meet the requirements of many embedded systems. The flexible heterogeneous Multi-Core processor (FMC) is an example of the fusion of these two architectures that can deliver both a high throughput for uniform parallel applications and high performance for fluctuating general purpose workloads [34]. Reconfigurable architectures are dynamic, adjusting their complexity, speed and performance level in response to the currently executing application. With this property in mind, we disregard systems that are statically reconfigurable but fixed while operating, such as traditional FPGAs, considering only architectures that are run-time reconfigurable.

1) *Dynamic Partial Reconfiguration*: FPGA manufacturers such as Xilinx and Altera now offer a mechanism called Dynamic Partial Reconfiguration (DPR) [35] or Self-Reconfiguration (DPSR) [36] to enable reconfiguration during run-time of the circuits within an FPGA, allowing a region of the design to change dynamically while other areas remain active [37]. The FPGA's architecture is partitioned into a static region consisting of fixed logic, control circuits and an embedded processor that control and monitor the system. The rest of the design space is allocated to a dynamic/reconfigurable region containing a reconfigurable logic fabric that can be formed into any circuit whenever hardware acceleration is required.

PDR/PDSR presents energy efficiency opportunities over fixed architectures. PDR enables the system to react dynamically to changes in the structure or performance and power constraints of the application, allowing it to address inefficiencies in the allocation of resources and more accurately implement changing software routines as dynamic hardware accelerators [35]. These circuits can then be easily removed or gated when they are no longer required to reduce power

consumption [38]. PDR can also increase the performance of an FPGA based system because it permits the continued operation of portions of the dynamic region unaffected by re-configuration tasks. Therefore, it allows multiple applications to be run in parallel on a single FPGA [36]. This property also improves the hardware efficiency of the system as, where separate devices were required, different tasks can now be implemented on a single FPGA, reducing power consumption and board dimensions. In addition, PDR reduces reconfiguration times due to the fact that only small modification are made to the bitstream over time and the entire design does not need to be reloaded for each change.

A study into the power consumption patterns of DPSR programming was conducted by Bonamy et al [9] to investigate to what degree the sharing of silicon area between multiple accelerators will help to reduce power consumption. However, many parameters must be considered to assess whether the performance improvement outweighs preventative factors such as reconfiguration overhead, accelerator area and idle power consumption and as such any gain can be difficult to evaluate. Their results show complex variations in power usage at different stages during reconfiguration that is dependent on factors like the previous configuration and the contents of the configured circuit. In response to these experiments, three power models are proposed to help analyse the trade-off between implementing tasks as dynamically reconfigurable, in static configuration or in full software execution.

Despite clear benefits, several disadvantages become apparent with this form of reconfigurable technology. As was shown above, the power consumption overhead associated with programming new circuits can effectively imposed a minimum size or usage time on circuits for implementation to be validated. In addition, a baseline power and area cost is also always created due to the large static region which continuously consumes power and can contain unnecessary hardware. Finally, the FPGA interconnect reduces the speed and increases the power consumption of the circuit compared to an ASIC implementation because of an increased gate count required to give the system flexibility.

2) *Composable and Partitionable Architectures*: Partitioning and composition are techniques employed by some dynamically reconfigurable systems to provide adaptive parallel granularity [39]. Composition involves synthesising a larger logical processor from smaller processing elements when higher performance computation or greater instruction or thread level parallelism (ILP or TLP) is required. Partitioning on the other hand will divide up a large design in the most appropriate way and assign shared hardware resources to individual cores to meet the needs of an application.

Composable Lightweight Processors (CLP) is an example of a flexible architectural approach to designing a Chip Multiprocessor (CMP) where low-power processor cores can be aggregated together dynamically to form larger single-threaded processors [39]. The system has an advantage over other reconfigurable techniques in that there are no monolithic structure spanning the cores which instead communicate using a microarchitectural protocol. In tests against a fixed-granularity processor, the CLP has been shown to provide a

42% performance improvement whilst being on average 3.4 times as area efficient and 2 times as power efficient.

Core Fusion is a similar technique to CLP in that it allows multiple processors to be dynamically allocated to a single instruction window and operated as if there were one larger processor [40]. The main difference from CLP is that Core Fusion operates on conventional RISC or CISC ISAs giving it an advantage over CLP in terms of compatibility. However, this also requires that the standard structures in these ISAs are present and so can limit the scalability of the architecture.

3) *Coarse Grained Reconfigurable Array Architectures:* Coarse-Grained Reconfigurable Array (CGRA) architectures represent an important class of programmable system that act as an intermediate state between fixed general purpose processors and fine-grain reconfigurable FPGAs. They are designed to be reconfigurable at a module or block level rather than at the gate level in order to trade-off flexibility for reduced reconfiguration time [41].

One example of a CGRA designed with energy efficiency as the priority is the Ultra Low Power Samsung Reconfigurable Processor (ULP-SRP) presented by Changmoo et al [42]. Intended for biomedical applications as a mobile healthcare solution, the ULP-SRP is a variation of the ADRES processor [43] and uses 3 run-time switch-able power modes and automatic power gating to optimise the energy consumption of the device. Experimental results when running a low power monitoring application show a 46.1% energy consumption reduction compared to previous works.

III. CASE STUDY - PICO MIPS

In this section, we present the picoMIPS architecture as an example of a minimal and energy efficient processor implementation. The key points of the architecture will be described and evaluated, showing how it is a novel concept in minimal architecture synthesis. Developments are proposed to the architecture, that can improve performance and maintain energy efficiency, using the technologies described in the previous section.

The picoMIPS architecture is foremost a RISC microprocessor with a minimised instruction set architecture (ISA). Each implementation will contain only the necessary datapath elements in order to maximise area efficiency as the priority. For example, the instruction decoder will only recognise instructions that the user specifies and the ALU will only perform the required logic or arithmetic functions. Due to the proportionality between logic gate count and power consumption, energy efficiency is also maximised in the processor therefore the system is designed to perform a specific task in the most efficient processor-based form.

By synthesising the picoMIPS as a microprocessor, a base-line configuration is established upon which functionality can be added or removed, in the form of instructions or functions, while incurring only minimal changes to the area consumption of the design. If the task was implemented as a specific dedicated hardware circuit, any changes to the functionality could have a large influence on the area consumption of the design. Figure 4 shows an example configuration for the

picoMIPS which can accommodate the majority of the simple RISC instructions. It is a Harvard architecture, with separate program and data memories, although the designer may choose to exclude a data memory entirely. The user can also specify the widths of each data bus to avoid unnecessary opcode bits from wasting logic gates.

The principles of the picoMIPS processor have been implemented in a few projects to demonstrate the concept of minimal architecture synthesis and how it can be used to produce an application specific, energy efficient processor. The discrete cosine transform (DCT) algorithm, a stage in JPEG compression, was synthesised into a processor architecture based on the picoMIPS concept. The resulting processor was more area efficient than a GPP due to the removal of unnecessary circuitry however its functionality was also reduced to performing only those functions which appear in the DCT algorithm. The processor can also be compared to a dedicated ASIC hardware implementation of the DCT algorithm. An ASIC implementations have a much higher performance and throughput of data however this is at the cost of area and energy efficiency. The picoMIPS therefore represents a balance between the two, sacrificing some performance for area and energy efficiency benefits.

The picoMIPS has also been implemented to perform the DCT and inverse DCT (IDCT) in a multi-core context [44]. A homogeneous architecture was deployed with the same single core structure, as in figure 4, being replicated 3 times. The cores are connected via a data bus to a distribution module as shown in figure 5 where block data is transferred to each core in turn. This structure theoretically triples the throughput of the system as it can process multiple data blocks in parallel.

As a microprocessor architecture, the picoMIPS can implement many of the technologies discussed in section II in order to improve energy efficiency. Clock gating, power domains and DVFS will all benefit the system however the area overhead of implementing them must first be considered as necessary. Pipeline balancing and caching can be integrated into more complex picoMIPS architectures however these are performance focused improvements and so are not priorities in the picoMIPS concept. The expansion of the system to multi-core is also one that can be employed to improve performance. Moreover, a heterogeneous architecture could be implemented to allow the picoMIPS to process multiple different applications simultaneously using several tailored ISAs. Reconfigurability can also be applied to picoMIPS to create an architecture which can be specific to each application that is executed, effectively creating a general purpose yet application specific processor. This property would require run-time synthesis algorithms to detect and develop the instructions and functional units that are required, before executing the application.

IV. CONCLUSION

A new concept of variable-architecture application-specific approach to embedded processor design has been presented. Over the past few decades, the trend in processor architectures, has evolved from single core to homogeneous multi-core and

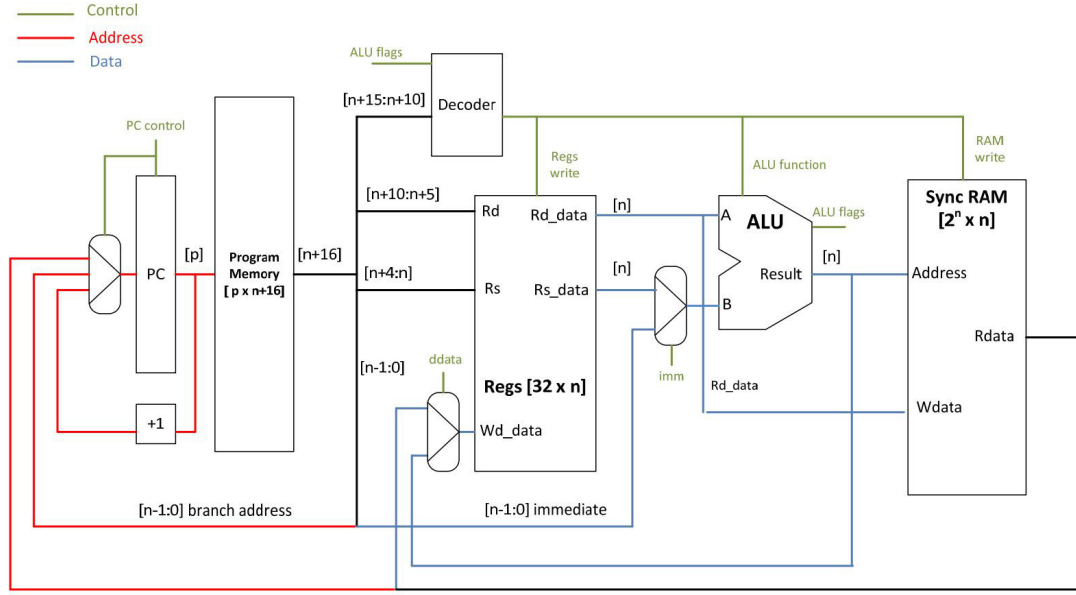


Fig. 4. An example implementation of the picoMIPS architecture.

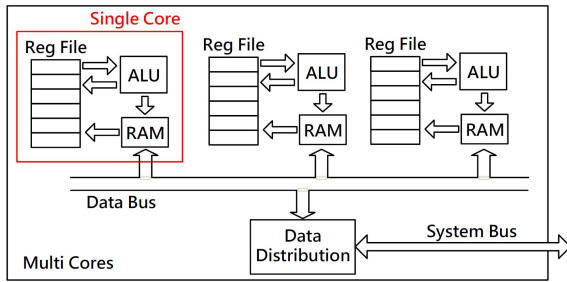


Fig. 5. A Multi-core implementation of the picoMIPS architecture, reprinted from [44].

into heterogeneous multi-core at the present. The proposed approach lends itself easily to a multi or many core architecture design where the performance is improved by enabling the simultaneous execution of threads independently on each core. The basic core design, of a core datapath and accelerators, may be replicated many times and integrated with some form of interconnect. This may form a homogeneous many-core processor, as all the cores are identical when no accelerators are connect. However, the variable-architecture processor is classed as a heterogeneous many-core processor due to the ability of run-time reconfiguration to make each core specific to the particular application that is currently executing on it during normal operation. Moreover, per-core DVFS controls can further differentiate each core using fine grain voltage and frequency adjustments that will affect the power consumption and performance of the core.

In addition to per-core DVFS control, an even finer granularity of control could be permitted through the use of per-accelerator DVFS controls. This links in with the per-accelerator power domains feature which is the first step

towards DVFS control. This allows a range of operating modes for each accelerator to allow fine tuning of the systems energy consumption. Accelerators which feature in critical paths of the architecture could be run at higher DVFS levels in order to reduce their latency.

The core level in a many-core system is the smallest duplicative region of the design featuring individual cores that contain their own core datapath and application specific accelerators. An intermediate layer of shared accelerators is implemented to allow neighbouring cores to share accelerators should they required additional hardware support. This approach is similar to core morphing, where cores are weak and strong in the execution of different instruction types. These levels can also form the basis for power domains so that a multi-level power control system can be implemented to allow fine grain control of the power consumption of the chip.

ACKNOWLEDGMENT

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC), UK under grant number EP/K034448/1 "PRiME: Power-efficient, Reliable, Many-core Embedded systems" website: (<http://www.prime-project.org/>).

REFERENCES

- [1] P. Greenhalgh, "big. LITTLE processing with ARM Cortex-A15 & Cortex-A7," *ARM White Paper*, 2011.
- [2] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, pp. 89–102, 2001.
- [3] C. Isci, A. Buyuktosunoglu, C.-Y. Chen, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, 2006, pp. 347–358.
- [4] V. Hanumaiah and S. Vrudhula, "Energy-efficient Operation of Multi-core Processors by DVFS, Task Migration and Active Cooling," *Computers, IEEE Transactions on*, vol. 63, no. 2, pp. 349–360, 2012.

- [5] R. Rodrigues, A. Annamalai, I. Koren, S. Kundu, and O. Khan, "Performance Per Watt Benefits of Dynamic Core Morphing in Asymmetric Multicores," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, 2011, pp. 121–130.
- [6] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," in *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, 2004, pp. 64–75.
- [7] M. Awan and S. Petters, "Energy-aware partitioning of tasks onto a heterogeneous multi-core platform," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, 2013, pp. 205–214.
- [8] R. Basmadjian and H. De Meer, "Evaluating and modeling power consumption of multi-core processors," in *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*, 2012, pp. 1–10. [Online]. Available: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6221107>
- [9] R. Bonamy, D. Chillet, S. Bilavarn, and O. Sentieys, "Power consumption model for partial and dynamic reconfiguration," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, 2012, pp. 1–8.
- [10] S. Saha, J. Deogun, and Y. Lu, "Adaptive energy-efficient task partitioning for heterogeneous multi-core multiprocessor real-time systems," in *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, 2012, pp. 147–153.
- [11] D. . Woo and H.-H. Lee, "Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era," *Computer*, vol. 41, no. 12, pp. 24–31, 2008.
- [12] B. de Abreu Silva and V. Bonato, "Power/performance optimization in FPGA-based asymmetric multi-core systems," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, 2012, pp. 473–474.
- [13] K. Wonyoung, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, 2008, pp. 123–134.
- [14] B. de Abreu Silva, L. Cuminato, and V. Bonato, "Reducing the overall cache miss rate using different cache sizes for Heterogeneous Multi-core Processors," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, 2012, pp. 1–6.
- [15] Q. Cai, J. Gonzalez, G. Magklis, P. Chaparro, and A. Gonzalez, "Thread shuffling: Combining DVFS and thread migration to reduce energy consumptions for multi-core systems," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, 2011, pp. 379–384.
- [16] P. Basset and M. Saint-Laurent, "Energy efficient design techniques for a digital signal processor," in *IC Design Technology (ICIDT), 2012 IEEE International Conference on*, 2012, pp. 1–4.
- [17] ARM, *ARM Cortex-A15 MPCore Processor Technical Reference Manual*, ARM, June 2013, pages 53 - 63.
- [18] A. Sinkar, H. Ghasemi, M. Schulte, U. Karpuzcu, and N. Kim, "Low-Cost Per-Core Voltage Domain Support for Power-Constrained High-Performance Processors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 4, pp. 747–758, 2013.
- [19] H. Ghasemi, A. Sinkar, M. Schulte, and N. S. Kim, "Cost-effective power delivery to support per-core voltage domains for power-constrained processors," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, 2012, pp. 56–61.
- [20] E. Rotem, A. Mendelson, R. Ginosar, and U. Weiser, "Multiple clock and Voltage Domains for chip multi processors," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, 2009, pp. 459–468. [Online]. Available: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5375435>
- [21] R. Bahar and S. Manne, "Power and energy reduction via pipeline balancing," in *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on*, 2001, pp. 218–229.
- [22] J. Sartori, B. Ahrens, and R. Kumar, "Power balanced pipelines," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, 2012, pp. 1–12.
- [23] R. Kumar, V. Zyuban, and D. Tullsen, "Interconnections in multi-core architectures: understanding mechanisms, overheads and scaling," in *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, 2005, pp. 408–419.
- [24] H. Zeng, J. Wang, G. Zhang, and W. Hu, "An interconnect-aware power efficient cache coherence protocol for CMPs," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1–11.
- [25] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen, "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction," in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, 2003, pp. 81–92.
- [26] P. Greenhalgh, "big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," ARM, Tech. Rep., September 2011.
- [27] H. M. Waidyasooriya, Y. Takei, M. Hariyama, and M. Kameyama, "FPGA implementation of heterogeneous multicore platform with SIMD/MIMD custom accelerators," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, 2012, pp. 1339–1342.
- [28] A. Lukefahr, S. Padmanabha, R. Das, F. Sleiman, R. Dreslinski, T. Wenisch, and S. Mahlke, "Composite Cores: Pushing Heterogeneity Into a Core," in *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, 2012, pp. 317–328.
- [29] B. Jeff, "Advances in big.LITTLE Technology for Power and Energy Savings," ARM, Tech. Rep., September 2012.
- [30] S. Zhang and K. Chatha, "Automated techniques for energy efficient scheduling on homogeneous and heterogeneous chip multi-processor architectures," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, 2008, pp. 61–66.
- [31] A. Z. Jooya and M. Analoui, "Program phase detection in heterogeneous multi-core processors," in *Computer Conference, 2009. CISCC 2009. 14th International CSI*, 2009, pp. 219–224.
- [32] L. Sawalha and R. Barnes, "Energy-Efficient Phase-Aware Scheduling for Heterogeneous Multicore Processors," in *Green Technologies Conference, 2012 IEEE*, 2012, pp. 1–6.
- [33] F. Calçado, S. Louise, V. David, and A. Merigot, "Efficient Use of Processing Cores on Heterogeneous Multicore Architecture," in *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on*, 2009, pp. 669–674.
- [34] M. Pericas, A. Cristal, F. Cazorla, R. Gonzalez, D. Jimenez, and M. Valero, "A Flexible Heterogeneous Multi-Core Architecture," in *Parallel Architecture and Compilation Techniques, 2007. PACT 2007. 16th International Conference on*, 2007, pp. 13–24.
- [35] M. Santambrogio, "From Reconfigurable Architectures to Self-Adaptive Autonomic Systems," in *Computational Science and Engineering, 2009. CSE '09. International Conference on*, vol. 2, 2009, pp. 926–931.
- [36] J. Zalke and S. Pandey, "Dynamic Partial Reconfigurable Embedded System to Achieve Hardware Flexibility Using 8051 Based RTOS on Xilinx FPGA," in *Advances in Computing, Control, Telecommunication Technologies, 2009. ACT '09. International Conference on*, 2009, pp. 684–686.
- [37] S. Bhandari, S. Subbaraman, S. Pujari, F. Cancare, F. Bruschi, M. Santambrogio, and P. Grassi, "High Speed Dynamic Partial Reconfiguration for Real Time Multimedia Signal Processing," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, 2012, pp. 319–326.
- [38] S. Liu, R. Pittman, A. Forin, and J.-L. Gaudiot, "On energy efficiency of reconfigurable systems with run-time partial reconfiguration," in *Application-specific Systems Architectures and Processors (ASAP), 2010 21st IEEE International Conference on*, 2010, pp. 265–272.
- [39] K. Changkyu, S. Sethumadhavan, M. S. Govindan, N. Ranganathan, D. Gulati, D. Burger, and S. Keckler, "Composable Lightweight Processors," in *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, 2007, pp. 381–394.
- [40] E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez, "Core fusion: accommodating software diversity in chip multiprocessors," in *Proceedings of the 34th annual international symposium on Computer architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 186–197. [Online]. Available: <http://doi.acm.org/10.1145/1250662.1250686>
- [41] Z. Rakossy, T. Naphade, and A. Chattopadhyay, "Design and analysis of layered coarse-grained reconfigurable architecture," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, 2012, pp. 1–6.
- [42] K. Changmoo, C. Mookyoung, C. Yeongon, M. Konijnenburg, R. Soojung, and K. Jeongwook, "ULP-SRP: Ultra low power Samsung Reconfigurable Processor for biomedical applications," in *Field-Programmable Technology (FPT), 2012 International Conference on*, 2012, pp. 329–334.
- [43] F. J. Veredas, M. Scheppler, W. Moffat, and B. Mei, "Custom implementation of the coarse-grained reconfigurable ADRES architecture for multimedia purposes," in *Field Programmable Logic and Applications, 2005. International Conference on*, 2005, pp. 106–111.
- [44] G. Liu, "Fpga implementation of 2d-dct/iddct algorithm using multi-core picomips," Master's thesis, University of Southampton, School of Electronics and Computer Science, September 2013.

The PRiME Framework: Application- and Platform-agnostic Runtime Management

Charles Leech, *Member, IEEE*, Graeme M. Bragg, James J. Davis, *Member, IEEE*, Eduardo Wachter, Geoff V. Merrett, *Member, IEEE*, George A. Constantinides, *Senior Member, IEEE*, and Bashir M. Al-Hashimi, *Fellow, IEEE*

Abstract—Heterogeneous processors in modern embedded platforms have increased in complexity to provide both high-performance and energy-efficient execution of applications. As a result, the management and control of hardware settings at runtime has become a non-trivial process. In addition, to exploit these processors, applications have become increasingly dynamic, with adjustable parameters that must be tuned to optimise their behaviour. These two challenges motivate the need for a runtime management approach that is cross-platform and generic in the support of applications. We present the PRiME Framework and API; a novel approach to standardise the runtime management of software applications executing on homogeneous and heterogeneous platforms. Our framework increases the mobility of applications, runtime management software and hardware platforms by separating them into three layers, connected via dynamic *knobs* and *monitors*, with a energy overhead of only 4.23% in the worst case. We experimentally demonstrate our framework across multiple applications and two heterogeneous platforms (Odroid-XU3 and Cyclone V). We demonstrate two state-of-the-art runtime management approaches that reduce the energy consumption of application execution by 18.2% and 17.2%. We present tradeoffs between power, performance and accuracy in three application-platform scenarios.

Index Terms—Heterogeneous many-core, runtime management, cross-layer frameworks, embedded systems, multicore processing, software tools, system software.

I. INTRODUCTION AND RELATED WORK

RUNTIME adaption and control are crucial to the efficient execution of applications with varying performance requirements on embedded heterogeneous platforms. Users demand ever-greater energy efficiency and throughput from these systems, therefore proactive optimisation of their performance, energy and reliability are key research challenges.

PRiME (Power-efficient, Reliable, Many-core Embedded systems) is an EPSRC funded research programme which is developing the theory and practices of future high-performance embedded systems utilising many-core processors.

C. Leech, G. M. Bragg, G. V. Merrett and B. M. Al-Hashimi are with the Department of Electronics and Computer Science, University of Southampton, University Road, Southampton, SO17 1BJ, UK (e-mail: c.leech@soton.ac.uk)

J. J. Davis and G. A. Constantinides are with the Department of Electrical and Electronic Engineering, Imperial College London, London, SW7 2AZ, UK (e-mail: james.davis@imperial.ac.uk)

This work was supported in parts by the PRiME programme grant EP/K034448/1 (<http://www.prime-project.org>) and EPSRC grant EP/L000563/1.

Data supporting the results presented in this paper will be made available online upon acceptance.

An open-source software implementation of the framework is available online at: https://github.com/PRiME-project/PRiME_Framework.

Manuscript received Month Date, Year; revised Month Date, Year.

Runtime management represents an essential paradigm in tackling these challenges by enabling optimisation and trade-offs between computational quality, application throughput, system reliability and energy efficiency. An increasing number of runtime management algorithms are being employed to control and optimise the execution of applications on heterogeneous embedded systems [1]–[6]. These approaches can be broadly classified by criticality (hard or soft real-time), optimisation technology or the employed learning and control method [7]. Dynamic power management (DPM) technologies are widely used in embedded platforms to reduce power consumption when workload requirements change. The most common of these is dynamic voltage and frequency scaling (DVFS), which allows operating systems (OSes) to change processor core voltages and clock frequencies during process execution, usually in accordance with manufacturer-determined voltage-frequency (V - f) pairs. Runtime algorithms controlling DVFS are commonly based on the utilisation of processor cores, including the Linux CPUFreq governors. Each governor operates a policy which defines upper and lower V and f limits as well as the rate and step size at which their transitions occur.

State-of-the-art DVFS-based runtime management algorithms are often based on machine learning. They are designed to either build predictive models of systems from historical data [8]–[10] or use reinforcement learning to iteratively adapt to changes in workload [11]–[13]. More sophisticated DPM approaches incorporate mapping and scheduling algorithms. In multi-threaded applications, levels of parallelism can be tuned using concurrency throttling [14]–[16]. For heterogeneous platforms, applications or their sub-processes may be assigned to specific hardware resources to optimise energy efficiency, leading to runtime resource management [17]–[21]. This is advantageous when portions of application code express differing computational behaviour at runtime, for example the level of data parallelism.

A drawback with existing runtime management algorithms is that they are usually designed for only specific classes of application, such as multimedia [22], [23], Web [24] or image processing [19]. In addition, the algorithms may have only been validated with specific applications or benchmarks, such as PARSEC [25], [26], SPLASH-2 [13] or SPEC [10], which can lead to over-optimistic results that are, in general, not transferable to new applications. Similarly, the runtime algorithm may only be designed to manage specific types of architectural configurations, *e.g.* homogeneous multipro-

cessors [12], [26], data centres [24], [27] or embedded systems [11], [18], or the experimentation has been conducted only on specific hardware platforms [10], [28]. Together, these factors limit the potential of these approaches. The provision of an application- and platform-agnostic framework for runtime management algorithms will alleviate many barriers to the wider development and testing of these runtime algorithms.

The runtime management of applications can be extended by the exposure and adaptation of tunable parameters through a defined framework interface. The concept of enhancing static applications with dynamic knobs has lead to methodologies such as PowerDial [29], Heterogeneous Heartbeats [2], ARGO [4] and AS-RTM [3]. The dynamic adaption of application knobs has been used for throughput-power trade-offs [30] in addition to precision-throughput tradeoffs [31]. Furthermore, exploration of the platform operating space has been used to locate Pareto-optimal points for tunable applications [32]. Formalisation of the monitoring of performance properties can be seen in the Application Heartbeats Application Programming Interface (API) [33], which allows information on the behaviour of applications to be conveyed to a RunTime Manager (RTM). The heartbeat concept has also been used to explore reliability-performance tradeoffs [34] and for task management on heterogeneous systems [35]. Additionally, methods of exposing device-level metrics in a standardised way have been demonstrated [36], [37].

These existing framework-based approaches can be broadly placed into three overlapping classifications: those that only abstract runtime algorithm-application [2]–[4], [34] or algorithm-device [36], [37] interaction; those that only expose monitors [2], [3], [35]–[37]; and those that are tightly coupled to a particular platform or device type or use device-specific runtime management algorithms [2], [3], [29], [35], [37]. Additionally, frameworks based on the concept of heartbeats are limited to applications that express their performance in terms of time [33]–[35].

The specification of a framework and API to support application- and platform-agnostic runtime management and control of both software applications and hardware platforms simultaneously is yet to exist. Many current runtime algorithms do not support tuning of application parameters or are tightly coupled to specific hardware platforms and/or applications. This limits the cross-application and cross-platform implementation of these techniques, preventing widespread adoption and direct comparison. We make the following novel contributions in addressing these shortcomings:

- We propose the separation of embedded systems into three distinct layers: device, runtime management and application, as shown in Fig. 1.
- We link these layers through an API and create cross-layer connections with constructs called knobs and monitors to control and capture runtime-tunable and -observable parameters.
- We experiment on two heterogeneous multiprocessor platforms to demonstrate cross-platform support: the Odroid-XU3 and the Altera Cyclone V.
- We present tradeoffs between power, performance and accuracy in three application scenarios, highlighting how

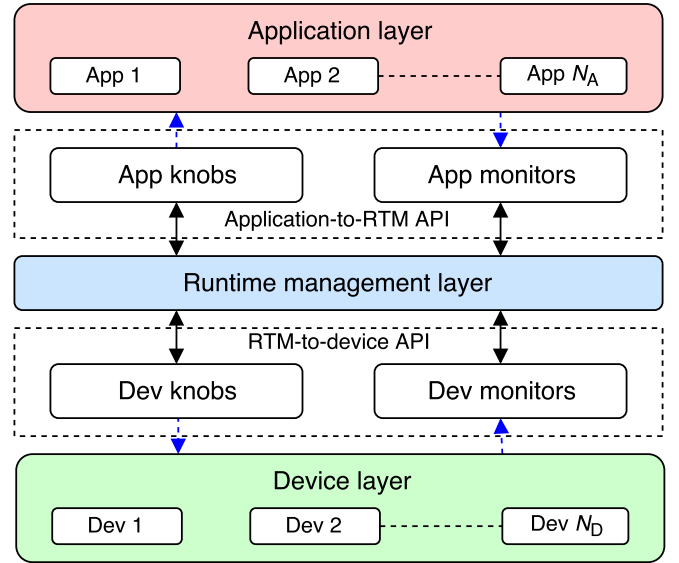


Fig. 1. Cross-layer framework and APIs enabling communication between the application, runtime management and device layers using knobs and monitors.

this operating space can be traversed at runtime.

- We demonstrate that our framework enables the reduction of the energy consumption of applications with two state-of-the-art runtime management approaches.
- We release an open-source implementation of the framework and API, developed in C++¹.

The remainder of this paper is divided as follows. In Section II, we outline the concepts behind the application- and platform-agnostic PRiME Framework, with the specification of an API presented in Section III. Section IV outlines our experimental methodology, providing technical details of the applications, RTMs and platforms that are demonstrated in Subsections IV-A, IV-B and IV-C. Experimental results of the framework are presented in Section V, including the cross-layer tradeoffs from application profiling and the demonstration of runtime management layer validation. Finally, we conclude in Section VII.

II. FRAMEWORK CONCEPTS

The PRiME Framework is an application-agnostic, cross-platform approach to runtime management. To achieve this, we set out its seven key concepts, detailed below.

Structure: We propose the separation of systems into three distinct layers—application, runtime management and device—as illustrated in Fig. 1. The application layer comprises any number of executing software processes, while the device layer includes the physical hardware and its software drivers. The runtime management layer contains an algorithm responsible for the control of both applications and devices in order to meet application-specified performance requirements while maximising the energy efficiency of the platform upon which they execute. By enforcing this separation, we ensure portability and cross-compatibility: applications and device

¹Available at: https://github.com/PRiME-project/PRiME_Framework

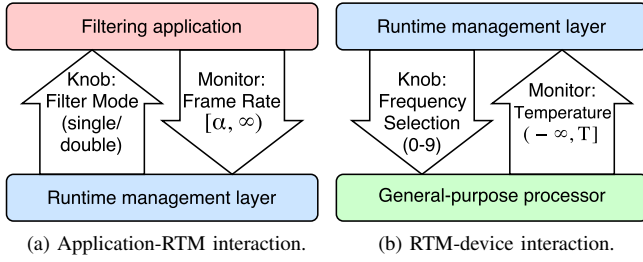


Fig. 2. Cross-platform knob and monitor use within the PRiME Framework.

drivers need only be written once to be used with any runtime management approach implemented within the framework.

Communication: Key to the framework are the *knobs* and *monitors* that facilitate the flow of information between the system layers. Knobs embedded within applications by their developers allow the RTM to adjust runtime-tunable parameters, if they provide any, in order to meet objectives that can be verified by observing the values of similarly embedded application monitors. Consider, for example, the filtering application shown in Fig. 2a, whose precision can be varied at runtime, with resultant performance (expressed as a frame rate) updated during execution. If, in this case, the application exposed a precision knob and a throughput monitor, the RTM would be able to establish the effect of adjusting the application’s precision, via the knob, on its performance through the monitor. Mirroring this, device knobs facilitate the tuning of their runtime-adjustable parameters, such as voltage or frequency, while device monitors allow hardware properties, such as power or temperature, to be read; an example of frequency selection and subsequent temperature observation is shown in Fig. 2b.

We propose that knobs and monitors be presented in a standardised, unitless format; their values are ‘just numbers’ that are passed between layers. The framework can be viewed hierarchically ‘downwards’ since, as far as knob and monitor control is concerned, applications are masters of the RTM, making calls to the API and controlling the presence and configuration of each knob and monitor, while devices are the RTM’s slaves since they must respond to requests to set and get knob and monitor values, respectively. Thus, applications ‘pull’ their knob settings from the RTM and ‘push’ monitor updates, while device knobs are pushed from the RTM and monitor values pulled.

Co-optimisation: The proposed, flexible combination of cross-layer knobs and monitors provides a powerful mechanism to enable optimisation across applications and devices. To direct the runtime management process, applications and devices specify *bounds* in the form of *minima* and *maxima*. Knob bounds represent a range of *allowed* values, while they are a range of *desired* values for application monitors. The RTM’s primary objective is to ensure that the true values of all application monitors remain within the specified bounds. Beyond this, it is free to optimise any unbounded monitors in either the application or device layer in order to meet any secondary objective(s), for example to reduce power consumption.

Calling again upon the previous examples, we would envisage the available precisions shown in Fig. 2a being represented by indices, e.g. 0 for single- and 1 for double-precision, thus an application knob $p \in \{0, 1\}$ would control their use. If some minimum frame rate α were required, this would be indicated to the RTM with an application monitor bounded by $[\alpha, \infty)$. Device knobs would be similarly bounded, e.g. the frequency knob f shown in Fig. 2b must be set such that $f \in \{0, 1, \dots, 9\}$.

Weights: We envisage scenarios in which applications have multiple performance objectives with differing priorities. For example, an application that is aware of both its throughput and accuracy may wish to prioritise the optimisation of one of these metrics over the other. If a monitor were provided for each along with respective weights, the RTM would be guided such that effort was proportionally expended in order to optimise the monitor values. Thus, we propose that each application monitor be coupled with a numeric weight.

Types: For device knobs and monitors, we opted to augment each with a *type* selectable from a discrete set of options. This choice was made as a compromise between complete agnosticism and full provision of information to the RTM. Versus applications, there are relatively few conceivable device parameters—voltage, frequency, power, *etc.*—and so we considered the provision of ‘hints’ to the RTM about the functions or consequences of particular knob or monitor changes, e.g. “lower power is better,” to be worthwhile since it prevents these conclusions from having to be learnt empirically.

Spaces: We propose the provision of discrete- and continuous-valued versions of each knob and monitor. For their implementation, discrete versions will use signed integer values while their continuous counterparts will operate using floating-point data instead.

Adaptability: Finally, to afford application developers maximal flexibility, we consider all bounds and weights to be adjustable at runtime, and also propose placing no restrictions on when registration and deregistration events—of knobs, monitors and the applications themselves—can occur. Generally, it is envisaged that applications will register their knobs and monitors on start-up and deregister them prior to termination, although we consider that there should be no limitation to such events occurring partway through application execution.

III. FRAMEWORK SPECIFICATION

The PRiME Framework is realised through application-to-RTM and RTM-to-device APIs, which connect the system layers of Fig. 1 (dashed boxes) and enable the exposure of knobs and monitors between the three layers. Table I illustrates how the API functions are split into application (app) and device (dev) categories, with subcategories for knob (knob) and monitor (mon) control functions. Discrete- and continuous-valued (disc and cont) versions of each function exist across the API functions to signify the type of knob or monitor that is being targeted.

As an example, the functions `prime_app_knob_disc_get()` and `prime_app_mon_cont_set()` can be formed from the permutations of the layer, construct, space

TABLE I
COMPLETE SET OF PRiME FRAMEWORK API FUNCTIONS.

Layer	Construct	Space	Identifier	Input(s)	Output(s)	Description	
app	–	–	reg	pid	–	Register application	
	–	–	dereg	pid	–	Deregister application	
	knob	disc / cont	reg	min, max, init	knob	Register application knob	
			dereg	knob	–	Deregister application knob	
			min	knob, min	–	Update application knob's minimum allowed value	
			max	knob, max	–	Update application knob's maximum allowed value	
			get	knob	value	Pull application knob's current value	
	mon		reg	min, max, weight	mon	Register application monitor	
			dereg	mon	–	Deregister application monitor	
			min	mon, min	–	Update application monitor's minimum desired value	
			max	mon, max	–	Update application monitor's maximum desired value	
			weight	mon, weight	–	Update application monitor's relative importance	
			set	mon, value	–	Push application monitor's current value	
	dev		knob	reg	–	knobs	Register all device knobs
				dereg	–	–	Deregister all device knobs
				min	knob	min	Pull device knob's minimum allowed value
				max	knob	max	Pull device knob's maximum allowed value
				init	knob	init	Pull device knob's initial (default) value
			type	knob	type	Pull device knob's type	
				set	knob, value	–	Push device knob's current value
mon			reg	–	mons	Register all device monitors	
			dereg	–	–	Deregister all device monitors	
		type	mon	type	Pull device monitor's type		
	get	mon	value	Pull device monitor's current value			

and identifier columns. These functions enable the pulling and pushing of application knob and monitor values from and to the RTM layer. Detailed explanation of all the functions defined by the API and their intended uses is presented in the Appendix.

We developed an implementation of the framework in C++, contained within the Linux operating system. Interprocess communication was achieved by passing messages over local Unix domain sockets (UDSes). An open-source release of this implementation is available online², including the example applications, RTMs and device classes discussed in the following section.

IV. BENCHMARKS

The PRiME Framework and its API enable rapid experimentation with different runtime management algorithms across multiple platforms and applications. The framework promotes comparison between state-of-the-art runtime algorithms through the standard knobs and monitor constructs. We demonstrate these qualities through a series of experiments using interchangeable components in each layer that we call benchmarks. These are described for each layer in the following sections.

A. Applications

The application layer contains application software and the requisite API functions to allow interaction with the runtime

management layer. Any application used within the framework must provide at least one monitor in order for the runtime management layer to meaningfully optimise its execution.

Three applications, described in Sections IV-A1, IV-A2 and IV-A3, were chosen to illustrate how the framework can enhance any software program to support runtime management. The knobs and monitors exposed for each application are listed in Table II, including whether they are discrete or continuous. In addition, we explore the power, performance and accuracy tradeoffs of these applications and demonstrate their runtime control within the framework.

1) *Jacobi Iterative Method*: The Jacobi matrix solver is an application with an iterative pattern of behaviour. The application solves the system of N linear equations $\mathbf{Ax} = \mathbf{b}$, where \mathbf{A} is an $N \times N$ matrix and \mathbf{x} and \mathbf{b} are $N \times 1$ column vectors. If \mathbf{A} is decomposed into diagonal and remainder components \mathbf{D} and \mathbf{R} , under suitable conditions \mathbf{x} can be computed iteratively via (1), also shown in an element-wise fashion in (2), where k is the iteration index. Each element of $\mathbf{x}^{(k+1)}$ relies only upon the previously calculated value of \mathbf{x} . Therefore, (2) can be parallelised by computing each $x_i^{(k+1)}$ independently.

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1} (\mathbf{b} - \mathbf{R}\mathbf{x}^{(k)}) \quad (1)$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right) \quad (2)$$

²Available at: https://github.com/PRiME-project/PRiME_Framework

TABLE II
APPLICATION-LEVEL KNOBS AND MONITORS PRESENT IN JACOBI
(SECTION IV-A1), VIDEO DECODER (IV-A2) AND WHETSTONE (IV-A3).

Application	Name	Const.	Space	Allowed/target values
Jacobi	Iterations	knob	disc	$\mathbb{N} \in [1, \infty)$
	Data type	knob	disc	$\{\text{float}, \text{double}\}$
	Device type	knob	disc	$\{\text{CPU}, \text{GPU}/\text{FPGA}\}$
	Throughput	mon	cont	$\mathbb{R} \in [10, \infty)$
	Error	mon	cont	$\mathbb{R} \in (-\infty, 1e^{-12}]$
Video decoder	Throughput	mon	cont	$\mathbb{R} \in [25, \infty)$
Whetstone	Threads	knob	disc	$\mathbb{N} \in [1, \infty)$
	Throughput	mon	cont	$\mathbb{R} \in [2.5, \infty)$

The result of the application is evaluated against the convergence criterion $\|\mathbf{Ax} - \mathbf{b}\| < \epsilon$, which is used for establishing the final result's accuracy and is dependent on K , the number of iterations performed. Tuning K operates a trade-off between accuracy and computational speed. The application exposes three knobs and two monitors, presented in the first section of Table II. In the context of this experiment, $\|\mathbf{Ax} - \mathbf{b}\|$ is captured as an error monitor, with maximum bound ϵ . A throughput monitor reports the time taken to complete K iterations of the algorithm. The application was implemented as OpenCL kernel code to support execution on multiple devices; we experimented with an embedded GPU and FPGA for acceleration. K work-groups, each comprised of N work-items, one per element of \mathbf{x} , are launched for each iteration. Two distinct kernel implementations are provided for single- and double-precision floating-point operations. Along with a variable number of iterations, this enables tradeoffs to be made between precision and accuracy. The ARM CPU clusters we used are not OpenCL-compatible devices, therefore a multi-threaded implementation of the application was created using POSIX threads as well. The *device type* application knob is provided to support switching between implementations. In Section V-A, we profile these knobs and monitors to expose the Pareto-optimal operating points for a trade-off between error and performance.

2) *Video Decoder*: The video decoder is a multimedia application which uses the OpenCV VideoCapture API to open a video file and read in frames. The application exposes a single monitor, as shown in the second section of Table II. This performance monitor gives the time taken to decode each video frame. The target frame rate can be controlled using the bounds of the monitor.

3) *Whetstone Benchmark*: Whetstone is a benchmark that measures the performance of integer and floating-point arithmetic with a variety of numerical functions [38]. While it is a synthetic benchmark, Whetstone exercises many of the subsystems of a CPU used by numerical applications, making it a suitable proxy for assessing their performance within the framework. The benchmark is divided into ten modules representing a different arithmetic, data movement or control flow operation, in the patterns which they occur, for real-world numerical applications. The knobs and monitors exposed by the application are shown in last section of Table II. The

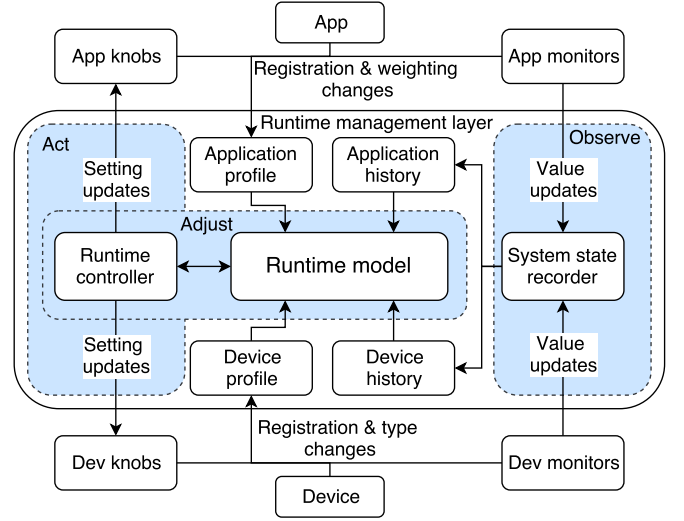


Fig. 3. Example structure of the runtime management layer with the act-observe-adjust phases highlighted.

knob controls the number of threads that are executed by the application simultaneously. Each thread executes an independent benchmark, rather than threading multiple benchmarks, to ensure that all cores in a multi-core system are equally exercised. The monitor represents the performance of the application measured in thousands of Whetstone instructions per second (KIPS). This is calculated as $\text{KIPS} = 100N_{\text{loop}}/t_{\text{exec}}$, where N_{loop} is the loop multiplier, a scaling factor applied to the number of repetitions of each module to increase the execution time, and t_{exec} is the time taken to execute all of the modules.

B. Runtime Managers

The runtime management layer is central to the operation of the framework as it is responsible for controlling the application and device layers through the knobs and observing the state of the system through the monitors, as illustrated in the central layer of Fig. 1. The contents of this layer can take any form, such as those discussed in Section I. We show an example structure in Fig. 3, which places an RTM in the context of the PRiME framework and shows the act-observe-adjust process. This RTM arrangement is centred around a runtime model of the system, which encodes information on the tradeoffs between all the application and device knobs and monitors exposed by the framework. Registrations, bound, weighting and type changes feed into the application and device profiles. Profiles contain information about the structure of applications and devices and are modified when registration or configuration updates occur. Data is fed into the RTM's state recorder from monitor value updates and is filtered into a cache of application and device data histories. This enables the layer to both observe and learn the system's behaviour.

The profiles and histories are used as inputs to construct the runtime model. The runtime controller interrogates the runtime model to predict optimal knob settings based on bounds set by the monitors and attempts to co-optimize any that are

unbounded. For example, gradient descent can be used to optimise power consumption under a performance constraint. This achieves the act and adjust elements of our methodology.

We demonstrate the operation of the runtime management layer within the framework with two state-of-the-art approaches developed by the PRiME Project. These are discussed in more detail in the following two sections and their operation within the framework is experimentally validated in Section V-B. In addition, we use a profiler to explore the runtime operating space by testing every combination of device and application knob values and recording the resulting monitor values. This exposes the relationships between knobs and monitors and the potential tradeoffs between monitors, which are demonstrated experimentally in Section V-A. This data forms the basis of an empirical model of the system which can be used by an RTM to make predictions and optimisations at runtime. Previous work, validated with several image processing applications, has used regression-based learning of energy-throughput tradeoffs to generate such a runtime model and perform prediction online [19].

1) *Q-Learning Runtime Manager*: This RTM uses a Q-Learning algorithm to ensure that application-specific performance requirements are met while reducing system energy by scaling the operating voltage and frequency [39]. In a standard Q-learning approach, a learning agent (the RTM) repeatedly observes the state of the system at a set rate in order to predict the next state with predictions updated and stored in a Q-table. Based on this prediction an appropriate action is selected to achieve the specified optimisation objective. The objective, for example application performance or power reduction, is quantified using a numeric payoff, with positive payoffs being classed as a reward and negative ones as a penalty.

Initially the RTM has no knowledge of how its actions will affect the state of the system or what rewards its actions will produce. As such, the RTM starts with an *exploration* phase where it experiments with various actions in different states to determine the predicted payoff for each action. Over time, the confidence in the selected action improves and the algorithm always selects the action corresponding to the highest payoff. This is known as the *exploitation* phase. The algorithm balances the two modes of operation with an epsilon-decreasing strategy, where the best action is selected for the proportion $(1 - \epsilon)$ of states and a random state is selected for the proportion ϵ . ϵ decreases over time in order to favour *exploitation* of known high rewards.

The Q-learning RTM used within the framework was automatically synthesised from a formally verified Event-B model using a code generation tool [40]. This form of runtime algorithm is best suited to frame-based applications. In this implementation, the RTM is updated every 10 frames or cycles of the application and the state of the system is determined using the CPU cycle count. The operation of this RTM with the video decoder application operating in the framework is validated in Section V-B.

2) *PMC-based Runtime Manager*: This RTM classifies workloads based on their memory intensiveness using the memory reads per instruction (MRPI) metric [41]. Workloads with a high MRPI are classified as memory intensive and the

$V-f$ level is scaled down to minimise power consumption as memory-intensive workloads can be run at lower frequency with little to no performance loss. MRPI is derived from two CPU performance counters, the number of last-level cache data read refills and instructions retrieved, and shows a high correlation with the memory intensiveness of a workload with little variation due to frequency scaling when compared to CPU cycle count and memory reads per cycle.

Unlike the Q-Learning RTM, the PMC-based RTM uses offline profiling to support workload classification and $V-f$ selection. Workload prediction is carried out using an exponentially weighted moving-average filter and, to minimise workload mispredictions, the predicted workload for the interval $t - 1$ to t is compared with the actual workload measured from hardware PMCs. The computed prediction error—the difference between actual and predicted workloads—is used to improve the workload prediction for t to $t + 1$.

This RTM is used to demonstrate that approaches targeted at non-frame-based applications or that rely on offline profiling can be used within the context of the framework. Validation of this RTM is presented in Section V-B.

C. Devices

The device layer is responsible for interfacing between the runtime management layer and the underlying hardware, facilitating the control of device parameters, such as operating frequency and voltage, and presentation of device data sources in a standardised manner across different platforms. One platform can incorporate multiple devices and, for the purposes of the framework, a device is defined as a single functional unit, such as a CPU cluster, graphics processing unit (GPU), digital signal processor (DSP) or field-programmable gate array (FPGA) fabric. Specific types are assigned to the knobs and monitors exposed by the device layer as summarised in Table III. Additionally, to allow the runtime management layer to be as device-agnostic as possible, the device layer communicates information about which knobs and monitors are available, and their relationships to each other, to the runtime management layer at runtime.

The runtime management layer interfaces directly to the operating system to control CPU thread-to-core affinity. This is preferable to the provision of a knob from the device layer as it avoids transferring application-specific information to the device.

Two heterogeneous multi-core platforms have been used to demonstrate knob and monitor support for devices within the PRiME Framework:

1) *Odroid XU3*: The Odroid XU3 is an embedded development platform built around the Samsung Exynos 5422 system-on-chip (SoC), containing a heterogeneous multiprocessor architecture with four ARM Cortex-A7 and four Cortex-A15 CPUs, arranged in two performance-asymmetric quad-core clusters, and a six-core Mali T624 GPU. The presence of multiple heterogeneous processing elements makes this an ideal platform for experimentation with runtime management techniques. The platform contains a rich set of parameters that are exposed through the framework as knobs and monitors;

TABLE III
AVAILABLE DEVICE-LEVEL KNOB AND MONITOR TYPES.

Construct	Type	Description
knob	EN	Enables device
	VOLT	Controls voltage
	GOVERNOR	Controls CPU frequency governor
	FREQ_EN	Enables <i>user space</i> CPU frequency control
	FREQ	Controls frequency
	PMC_CTRL	Controls performance counter
mon	POW	Reports power consumption
	TEMP	Reports temperature
	PMC	Reports performance counter data
	CYCLES	Reports cycle count data

TABLE IV
DEVICE-LEVEL KNOBS AND MONITORS FOR ODROID-XU3
(SECTION IV-C1) AND CYCLONE V (IV-C2) PLATFORMS.

Plat.	Const.	Space	Type	For	No.
Odroid-XU3	knob	disc	GOVERNOR	A7 cluster	1
		disc	GOVERNOR	A15 cluster	1
		disc	FREQ	A7 cluster	1
		disc	FREQ	A15 cluster	1
		disc	FREQ_EN	GPU DVFS	1
		disc	FREQ	GPU	1
		disc	PMC_CTRL	A7 cores	16
		disc	PMC_CTRL	A15 cores	24
	mon	cont	POW	Clusters, RAM, GPU, SoC	5
		cont	TEMP	A15 cores	4
		cont	TEMP	GPU	1
		disc	CYCLE	A7 cores	4
		disc	CYCLE	A15 cores	4
		disc	PMC	A7 cores	16
Cyclone V	knob	cont	VOLT	A9 cluster, peripherals	4
		cont	VOLT	FPGA, peripherals	3
	mon	cont	POW	A9 cluster, peripherals	5
		cont	POW	FPGA, peripherals	4
		cont	POW	SoC	1

these are summarised in the first half of Table IV. The six knobs come from frequency and governor control of each of the CPU clusters and frequency control of the GPU. The 58 monitors are derived from a variety of sources included in the platform: five power measurements come from hardware sensors on the board, each A7 core has a cycle counter and four PMCs, each A15 core has a cycle counter, six PMCs and a temperature sensor and the GPU has a temperature sensor.

2) *Cyclone V*: The Altera Cyclone V SoC development board tests the operation of the framework with a CPU-FPGA platform. The SoC features two ARM Cortex-A9 CPUs and an FPGA fabric with 42k look-up tables, tightly coupled and manufactured on a low-power 28nm process. Knobs and monitors exposed through the device's class are detailed in the lower half of Table IV. The two power regulators present on the development board facilitated the independent control of seven voltage rails, including CPU and FPGA core voltages, along

with the monitoring of their realtime power consumptions. OpenCL applications can be executed on the FPGA fabric by despatching kernel tasks to pre-compiled accelerators.

V. EXPERIMENTAL EVALUATION AND RESULTS

Using the application, RTM and device components described in the previous section, we have conducted a series of experiments to demonstrate that application- and platform-agnostic runtime management can be achieved through our framework. Firstly, in Section V-A, we profile the three applications described in Section IV-A across the two platforms described in Section IV-C to validate the application-agnostic properties of the framework. To demonstrate the flexibility of the runtime management layer in the framework, Section V-B presents validation of the operation of the two state-of-the-art approaches from Section IV-B, taking advantage of the cross-layer API to dynamically adjust and observe application and device knobs and monitors, respectively.

A. System Profiling and Tradeoff Analysis

Design space exploration was performed using the profiling approach described in Section IV-B as the runtime management layer on the three applications of Section IV-A and across both platforms for the Jacobi application. For Jacobi, the dimensions of the matrix and vectors, N , is set to 4096. We evaluated the error, performance and power characteristics of each application-platform configuration and extracted the Pareto-optimal points to create the tradeoffs shown in Figs 4, 5 and 6, with results for both platforms shown in Fig. 4. Optimal points are highlighted as blue crosses for the Odroid and green triangles for the Cyclone V, with sub-optimal points shown as red dots. Table V lists the application and device knob values for each of the labelled points in Fig. 4, for the Odroid platform. The optimal points of Fig. 4 are clustered in their error value, due to having the same number of iterations K , this confirms the behaviour predicted from (2). Furthermore, a performance decreases as number of iterations increases due to the additional latency in computing $x[k+1]$. The points within each cluster are distributed across a range of performances due to their different frequency settings. Power is not a dimension of importance in this trade-off therefore lower performance points are considered sub-optimal even if they lead to a better performance-per-watt. This trade-off is considered in 5 for the video decoder application. Device selection determines whether frequency control is activated. DVFS control is passed back to the default frequency governor for all devices that are not executing the application for that operating point, shown as GOV in Table V. The Cyclone V platform can achieve the lowest error levels (10^{-20} for 9 iterations at double precision using the FPGA-based kernel), however its performance is lower than the Odroid even at higher errors due to its limited frequency range and CPU cores with lower compute capabilities.

We demonstrate the runtime selection of Pareto-optimal operating points for the Jacobi application and Odroid platform by using a look-up table process in the runtime management layer. Fig. 7 gives insight into the RTM's behaviour

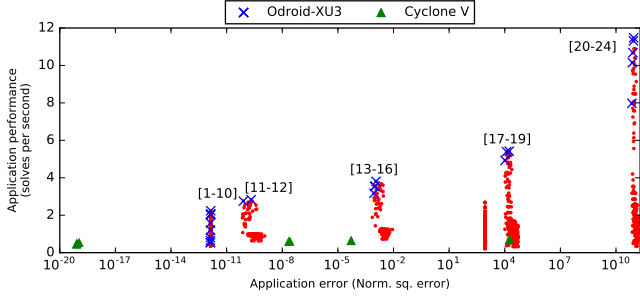


Fig. 4. Pareto-optimal states for the error and performance trade-off of the Jacobi application on the Odroid-XU3 and Cyclone V.

TABLE V

KNOB AND MONITOR VALUES FOR PARETO-OPTIMAL POINTS IN FIG. 4.

No.	Error $\ Ax - b\ $	Time (ms)	Iters K	Prec. (bits)	Device	Frequency (MHz)		
						A7	A15	GPU
1	1.32×10^{-12}	0.537	8	double	GPU	1400	1200	3
2	1.33×10^{-12}	0.537	8	double	GPU	1400	1200	2
3	1.34×10^{-12}	0.649	6	double	GPU	1400	600	4
4	1.34×10^{-12}	0.801	5	double	GPU	1400	1100	1
5	1.34×10^{-12}	1.21	6	double	CPU	1400	800	GOV
6	1.34×10^{-12}	1.58	5	double	CPU	600	1200	GOV
7	1.36×10^{-12}	2.02	5	double	CPU	1200	1600	GOV
8	1.37×10^{-12}	2.05	5	double	CPU	1000	1800	GOV
9	1.38×10^{-12}	2.07	5	double	CPU	1400	1800	GOV
10	1.39×10^{-12}	2.24	5	double	CPU	1400	200	GOV
11	7.97×10^{-11}	2.76	4	double	CPU	600	200	GOV
12	2.12×10^{-10}	2.83	4	double	CPU	1400	200	GOV
13	8.86×10^{-4}	3.18	3	double	CPU	800	1600	GOV
14	9.60×10^{-4}	3.54	3	double	CPU	600	1800	GOV
15	1.01×10^{-3}	3.56	3	double	CPU	1400	1800	GOV
16	1.15×10^{-3}	3.81	3	double	CPU	600	200	GOV
17	1.04×10^4	4.93	2	double	CPU	1400	1600	GOV
18	1.26×10^4	5.40	2	double	CPU	600	200	GOV
19	1.75×10^4	5.42	2	double	CPU	1400	200	GOV
20	7.21×10^{10}	7.98	1	double	CPU	1000	1000	GOV
21	7.61×10^{10}	10.1	1	double	CPU	1000	1800	GOV
22	8.16×10^{10}	10.7	1	double	CPU	600	1600	GOV
23	8.75×10^{10}	11.3	1	double	CPU	1400	200	GOV
24	8.99×10^{10}	11.5	1	double	CPU	600	200	GOV

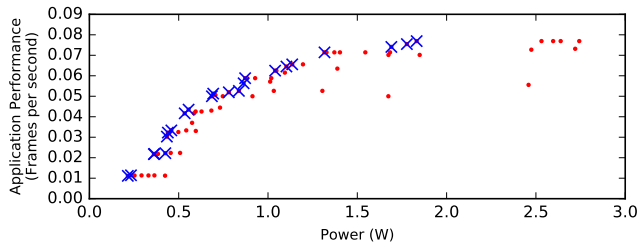


Fig. 5. Pareto-optimal states identified by the RTM for power and performance trade-off of the video decoder application on the Odroid-XU3.

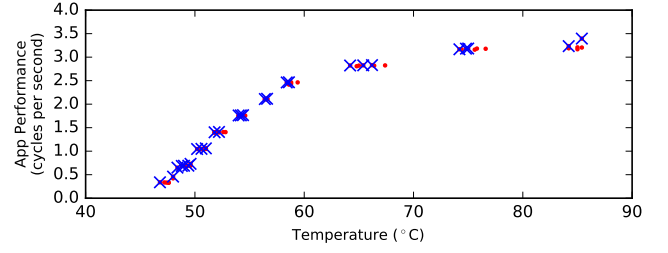


Fig. 6. Pareto-optimal states identified by the RTM for temperature and performance trade-off of the Whetstone application on the Odroid-XU3.

with the adjustment of knob settings over time. The knob settings at every operating point ensure that the predicted application monitor values were within their target ranges (*i.e.* $\text{perf}_{\min} \leq \text{perf}_{val|t} \leq \text{perf}_{\max}$) and device monitors were optimised (*i.e.* minimum power and temperature). In order to sweep across a range of points in the operating space, the minimum bound of the performance monitor of the Jacobi application incremented by 0.4 solves per second every 40 seconds, shown in the top sub-plot of Fig. 7. The value and upper bound of the error monitor are shown in the next sub-plot in the figure. The error value is free to fluctuate within the specified bounds if it allows other monitor values to be optimised (*i.e.* power). The next two sub-plots show the value of the application knobs, specifically iterations and device type. The data type knob remained set at double throughput the experiment and so it is not plotted. Iterations had a minor effect on performance and a value above 5 was sufficient to ensure that the error remained in the order of 10^{-12} , with 3 iterations being the least that ensured the maximum bound was met. The CPU and GPU frequencies (bottom sub-plot) had the strongest affect on performance and during periods of high performance requirement higher frequencies were used. A low performance bound allowed the reduction of the CPU or GPU frequency, leading to a reduction in power consumption. Finally, all the temperature monitors are plotted in the fifth sub-plot down in Fig. 7 which all show a strong correlation with frequency and power consumption.

B. Runtime Manager Validation

Cross-layer connections created through the knobs and monitors of the framework increase the separability of the runtime management layer from the device and application layers. This promotes the development of interchangeable runtime management approaches that can be enabled and disabled to suit the current workload. We validate that the two runtime management layers operate within the framework in the same way as in their source publications. The same platform and application (Odroid-XU3 and video decoder) are used to show that multiple runtime algorithms can provide energy savings. The experiment is repeated 50 times for each runtime algorithm, in order to evaluate the distribution of the energy consumption of the application under RTM control, as shown in Fig. 11, and establish the average energy saving.

1) *Q-Learning RTM*: We experimentally validated the operation of the Q-Learning algorithm described in Section IV-B1

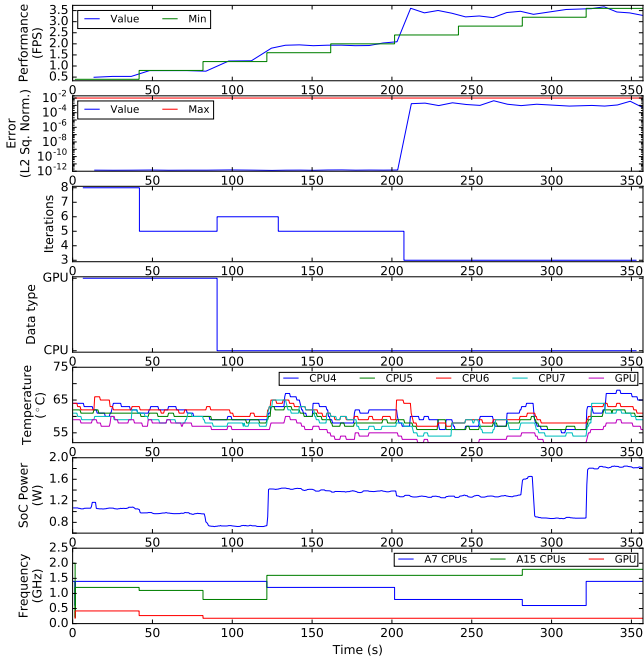


Fig. 7. Runtime management layer responding to changes in application requirements by updating application and device knob settings.

in Fig. 8 in terms of the power consumption over time while running the the video decoder application. Comparison was made against the Ondemand Linux frequency governor (red crosses). The two phases of the Q-Learning approach, exploration and exploitation, are highlighted, from 0 to 85 seconds and from 85 seconds onwards, respectively. A greater variation in power consumption was observed during the exploration phase due to a higher ϵ value as the Q-Table was being populated. In the exploitation phase, the variance in the system state reduced as ϵ decreased and the power consumption was more consistent as a result. Fluctuations during the exploitation phase were in response to changes in the application processing requirements for specific frames.

The average energy per frame for the Q-learning RTM was 3.38mJ, lower than the Ondemand governor's 4.13mJ. Therefore, the RTM achieved an 18.2% energy saving across the program's whole execution. However, this came at a performance impact of 7.66% in terms of FPS.

2) *PMC-based RTM*: We use the PMC-based RTM described in Section IV-B2 as an alternative runtime management layer. The operation of the algorithm is demonstrated in the top part of Fig. 9 in terms of the power consumption required to decode each frame. Comparison is made against the Ondemand frequency governor, which showed consistently higher average energy per frame. The RTM achieved an average energy saving of 17.2%, however this came at the cost of a performance impact of 6.90% FPS.

The calculated MRPI and measured frequency are shown in the middle and lower parts of Fig. 9. This demonstrates the inverse relationship between MRPI and selected frequency, with a larger MRPI resulting in a lower core frequency being selected. This relationship comes from the knowledge that

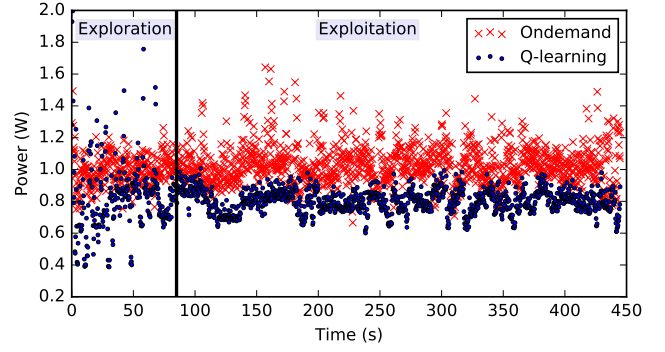


Fig. 8. Comparison between Q-learning RTM and Ondemand governor for the power consumption of the video decoder executing on the Odroid.

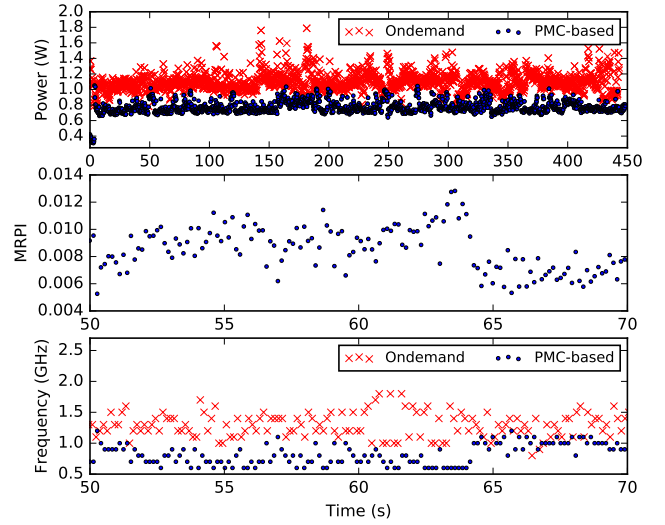


Fig. 9. Top: Comparison between PMC-based RTM and Ondemand governor for the power consumption of the video decoder executing on the Odroid. Middle and lower: the MRPI metric and the frequency choices of the PMC-based RTM and Ondemand governor.

a higher MRPI correlates with a memory-bound workload, therefore computational speed can be reduced to match the memory bottleneck, and power consumption is reduced as a result.

VI. FRAMEWORK OVERHEAD ANALYSIS

Introducing abstraction to achieve cross-platform and application-agnostic support means that the framework introduces latency and power overheads that reduce the overall efficiency of the system. The amount of overhead introduced is dependent on the the specific application, runtime management algorithm and device used. In this section we provide insight into the causes of these overheads theoretically and then experimentally quantify them for our implementation of the PRIME framework on the Odroid XU3.

A. Latency Overhead

Latency overheads capture the time required to perform cross-layer operations, *e.g.* reading monitors and setting knobs,

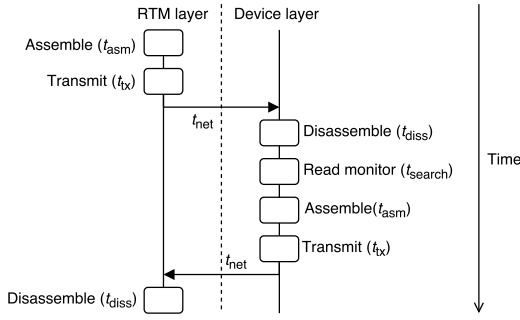


Fig. 10. Breakdown of the sources of latency introduced by the framework for communication between the RTM and device layers.

that impact on the responsiveness of the runtime algorithm. Fig. 10 visualises the steps involved in a monitor request from the runtime management layer to the device layer and shows that seven discrete components can be identified. This can be expressed as (3) where t_{asm} , t_{tx} and t_{diss} are the times to assemble, transmit and disassemble the message, t_{net} is the message-passing interface latency and t_{search} is the time to search for and read a monitor. $|_{rtm}$ denotes that the latency is evaluated in the context of the RTM layer, while $|_{dev}$ signifies a device layer-related term. The exact latency for each component is dependent on the length of the message and the processing required in each layer. These are platform-specific and quantified for our experimental setup and framework implementation by taking measurements of the time at different locations in the RTM and device classes.

$$t_{read_mon} = t_{asm}|_{rtm} + t_{tx}|_{rtm} + t_{diss}|_{dev} + t_{search} + t_{asm}|_{dev} + t_{tx}|_{dev} + t_{diss}|_{rtm} + 2t_{net} \quad (3)$$

We determine the latency in two scenarios: when the RTM requests a monitor from the device and when it sets a device knob. In the first experiment, two messages are involved (as in Fig. 10), while, in the second, a single message is needed. The overhead showed a variation of 400–800 μ s when reading a monitor and 80–200 μ s when setting a knob. Almost 40% of this overhead can be attributed to the UDS-based interface, t_{net} in Fig. 10, which was implemented with the boost C++ library for asynchronously notifying parallel processes.

B. Energy Overhead

Additional energy is consumed due to latency overheads and intermediate computations carried out by the API functions. Total energy usage can be aggregated with (4) from the energy cost of application, knob and monitor registration (E_{reg}) and deregistration (E_{dereg}) messages plus a sum of the product of the total number of messages m_i sent of each type i with the energy associated with each message (E_{msg}_i).

$$E_{total} = E_{reg} + \sum_i m_i (E_{msg}_i) + E_{dereg} \quad (4)$$

Moreover, the responsiveness of the RTM is bounded by the latency of API calls, *i.e.* the total time required to read

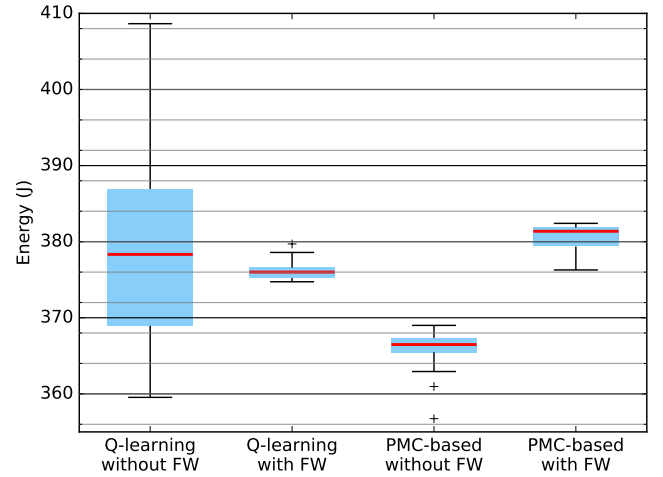


Fig. 11. Plot of the total energy consumed by the video decoder running under RTM control both with and without the framework.

a monitor and set a knob. This incurs an energy cost when a more energy-efficient system state s is available while $t < t_{read_mon} + t_{set_knob}$ and so such a potential optimisation is not visible to the RTM while the inequality holds.

We evaluated the energy overhead of the framework and API by measuring the total energy consumed by the video decoder application (Section IV-A2) when processing a 11125-frame video under the management of the Q-learning and PMC-based RTMs. We ran the experiment first using the PRiME Framework and then using a framework-less approach for both RTMs by temporarily merging the system layers. Fig. 11 presents the distribution of the total energy consumed to decode the video across 50 runs performed for each scenario. Whiskers reach to the last datum above and below 1.5 times the interquartile range (IQR), which is shown as a blue box centred on the mean. Beyond the whiskers, data are considered outliers and are plotted as individual points.

With the Q-learning RTM, the framework consumed 4.23% more energy in the worst case, shown as the lower whiskers in Fig. 11, which equate to a per-frame overhead of 1.37 mJ. However, the energy distribution with the framework was more consistent, using 0.836% less energy in the mean case. In contrast, the framework consumed 5.48% more energy for the PMC-based RTM in the worst case, with a per-frame overhead of 1.20 mJ, and 3.98% more in the mean case.

The on-board power sensors for the Odroid report the average power over 256ms intervals, and so are not able to capture the power contribution from individual messages, such as the registration and deregistration components in (4). The contribution of these terms to the total energy overhead diminishes as the application execution time increases, so we considered them to be negligible in our experimental setup. As a result, the per-message energy for the Q-learning RTM was estimated from the per-frame overhead with (5) and (6). One application and one device monitor was read each frame (costing energy: $E_{mon}|_{app} + E_{mon}|_{dev}$). Additionally, one RTM-device knob-setting ($E_{knob}|_{dev}$) occurred every tenth frame to

set the CPU frequency. Assuming that $2E_{\text{mon}}|_{\text{app}} \approx E_{\text{mon}}|_{\text{dev}} \approx 2E_{\text{kno}}|_{\text{dev}}$, the per-message energy E_{msg} can be estimated with (6). Taking the minimum-case, the per-message overhead was estimated to be 0.427mJ for the Q-Learning RTM.

$$10E_{\text{frame}} \approx 10E_{\text{mon}}|_{\text{app}} + 10E_{\text{mon}}|_{\text{dev}} + E_{\text{kno}} \quad (5)$$

$$E_{\text{msg}}|_{\text{Q-Learning}} \approx 5/16E_{\text{frame}} \quad (6)$$

The PMC-based RTM operates periodically every 200ms, during which it reads two device monitors and sets one device knob. Messages from the application layer still occur each frame. The per-message energy can be estimated by determining the average number of messages sent per frame with (7), where m_{frame} and m_{update} are the number of messages sent every frame and sent when the RTM updates, respectively. Taking the mean frame rate f of 24.8 FPS in the minimum case and the RTM update rate, t_{update} , of 200ms, the per-message overhead can be estimated to be 0.597mJ for the this RTM.

$$E_{\text{msg}}|_{\text{PMC}} \approx \frac{E_{\text{frame}}}{m_{\text{frame}} + \frac{m_{\text{update}}}{f \times t_{\text{update}}}} \quad (7)$$

VII. CONCLUSIONS

As heterogeneous embedded platforms increase in complexity, the need to meet application requirements and operate energy efficiently is the primary challenge in embedded system software design. In this paper, we presented the PRiME Framework, our novel methodology for the runtime management and co-optimisation of multiple objectives sought by concurrently executing applications, incorporating a cross-layer API which exposes knobs and monitors between the system layers. The framework increases the mobility of applications, runtime management software and hardware platforms by providing a standardised interface for communication and control. We conducted experimental validation of our framework using three applications from distinct domains on two heterogeneous platforms. We showed power, performance and accuracy tradeoffs for these application-platform scenarios to develop operating spaces which can be traversed at runtime.

We demonstrated two state-of-the-art runtime management algorithms operating through the framework, one based on reinforcement learning and the other on hardware performance metrics. The runtime managers achieved 18.2% and 17.2% energy savings, respectively, for a video decoding application compared to Ondemand Linux governor-controlled DVFS. Our framework has low latency and energy overheads arising from the introduced abstraction. In our implementation, the two runtime algorithms consume 4.23% and 5.48% more energy in the worst cases, and we estimated their per-message overheads to be approximately 0.43mJ and 0.60mJ, respectively.

We have released an open-source implementation of the framework to provide a standard approach to runtime management and encourage other researchers to develop additional applications, runtime algorithms and support more platforms using our API. The PRiME Framework enables this research to be conducted more efficiently and thoroughly, including facilitating comparison between competing runtime algorithms.

In the future, we will continue to integrate the framework API into additional applications, devices and runtime algorithms with the aim of increasing the framework's validation and providing greater demonstration of the contributions of application- and platform-agnostic runtime management. In addition, we will continue development of the implementation of the framework and its API to reduce the latency and energy overheads.

APPENDIX API SPECIFICATION

This appendix explains the purpose of the API functions listed in Table I that are defined in the PRiME Framework. This API is targeted at software developers designing applications, which expose knobs and monitors, as well as developers of runtime algorithms that control software programs and optimise hardware platforms.

As mentioned in Section III, the API functions are split into application (app) and device (dev) categories, with subcategories for knob (kno) and monitor (mon) control functions. The top half of Table I lists the application-to-RTM API functions that application developers must use to expose and update knobs and monitors. In the first instance, applications must register themselves through the API using `prime_app_reg()` to allow for their runtime management. This must be done before the registration of knobs and monitors. Applications can deregister using `prime_app_dereg()` when they no longer require runtime management, following knob and monitor deregistration.

Function `set_prime_app_(kno|mon)_(disc|cont)_(reg|dereg)()` facilitates the on-demand registration and deregistration of individual application knobs and monitors. It is envisaged that applications will register their knobs and monitors on start-up and deregister them prior to termination, although there is no limitation to such events occurring at arbitrary points throughout application execution instead. The number of knobs and monitors exposed is specific to the particular application.

Once registered, the RTM must be made aware of the allowable or desired values for knobs and monitors, respectively, in order to ensure that its optimisations are legal and have favourable effects. For knobs, functions `prime_app_kno_(disc|cont)_(min|max)()` facilitate this, letting the application indicate the range in which values can be chosen. Conversely, monitor functions `prime_app_mon_(disc|cont)_(min|max|weight)()` allow the setting of RTM objectives, with `*_min()` and `*_max()` functions indicating desired lower and upper bounds. Where an application demands only a maximum or minimum value, `PRIME_(DISC|CONT)_MIN` or `PRIME_(DISC|CONT)_MAX` may be used in place of the lower or upper bound, respectively. Intra-application weighting values between 0.0 and `PRIME_CONT_MAX` can be used to indicate relative monitor importance to the RTM using `*_weight()` functions, guiding its optimisations. All of these settings can be updated during application execution if required. To reduce bloat, the relevant settings are also arguments of the registration functions.

Listing 1

API FUNCTIONS REQUIRED TO REGISTER THE JACOBI APPLICATION (SECTION IV-A1) AND CREATE APPLICATION-LEVEL KNOBS AND MONITORS

```

1  // Register application with RTM
   prime_app_reg(getpid());

   // Set up application knob for controlling number of iterations
5  prime_app_knob_disc_t* iters_knob = prime_app_knob_disc_reg(1, PRIME_DISC_MAX, 10);

   // Set up application knob for controlling kernel precision. Two discrete options: 0 = float, 1 = double
   prime_app_knob_disc_t* prec_knob = prime_app_knob_disc_reg(0, 1, 0);

10  // Set up application knob for selecting device used to execute kernels. Two discrete options: 0 = CPU, 1 = GPU
   prime_app_knob_disc_t* dev_knob = prime_app_knob_disc_reg(0, 1, 1);

   // Set up application monitor for observing error. Required bound is (PRIME_CONT_MIN, CONVERGENCE_THRESHOLD)
   prime_app_mon_cont_t* error_mon = prime_app_mon_cont_reg(PRIME_CONT_MIN, CONVERGENCE_THRESHOLD, 1.0);

15  // Set up application monitor for observing throughput. Required bound is [THROUGHPUT_THRESHOLD, PRIME_CONT_MAX]
   prime_app_mon_cont_t* throughput_mon = prime_app_mon_cont_reg(THROUGHPUT_THRESHOLD, PRIME_CONT_MAX, 1.0);

```

Functions `prime_app_knob_(disc|cont)_get()` and `prime_app_mon_(disc|cont)_set()` enable the pulling and pushing of knob and monitor values from and to the RTM, respectively, by the applications that define them. The timing of these actions is entirely application-controlled.

Minimal modification is required to applications to expose dynamic knobs to the RTM through the framework, as illustrated in the application code snippet in Listing 1. Application registration is performed once and uses the process identifier (PID) to uniquely associate all of its knobs and monitors. In this example, a total of only 18 single-line function calls were added: 14 for registration and deregistration, four to pull knob setting updates and two to push monitor values.

Device-level knobs and monitors are exposed and updated via the RTM-to-device API functions, as shown in the lower half of Table I. Since device knobs and monitors are exposed to the RTM by the respective drivers, function `set_prime_dev_(knob|mon)_(disc|cont)_(reg|dereg)()` will register or deregister *all* knobs or monitors of a particular space (*i.e.* discrete or continuous) available within a device. We envisage the RTM querying for connected devices at regular intervals; those newly discovered will have their knobs and monitors registered automatically, while any that were disconnected will have them deregistered.

For each newly registered knob or monitor, the RTM must establish information about it to ensure that its proposed optimisations are sane. Knob-related functions `prime_dev_knob_(disc|cont)_(min|max)()` are parallels of their application-level counterparts, setting limits on the values that can be selected. Additional functions `prime_dev_knob_(disc|cont)_(type|init)()` return the type or initial—default from which the RTM should start its exploration—value. Monitors have type-indicating functions only, probed using `prime_dev_mon_(disc|cont)_(type)()`. Calls to `*_type()` functions return values from the sets presented in Table III.

At runtime, the RTM pushes device knob settings and pulls monitor values as it desires: functions `prime_dev_knob_(disc|cont)_set()` and `prime_dev_mon_(disc|cont)_get()` facilitate these actions.

ACKNOWLEDGEMENTS



This work was supported by the EPSRC-funded PRiME Project (grant number EP/K034448/1). <http://www.prime-project.org>

The authors would like to thank Joshua M. Levine and James R. B. Bantock for their role in the development of the PRiME Framework methodology and API. For contributions to the experimental results and the development of runtime algorithms, we would like to acknowledge Domenico Balsamo, Mohammad Sadegh Dalvandi and Basireddy Karunakar Reddy.

REFERENCES

- [1] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement Learning-based Inter- and Intra-application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *ACM/EDAC/IEEE Design Automation Conference*, 2014.
- [2] S. T. Fleming and D. B. Thomas, "Heterogeneous Heartbeats: A Framework for Dynamic Management of Autonomous SoCs," in *International Conference on Field Programmable Logic and Applications*, 2014.
- [3] E. Paone, D. Gadioli, G. Palermo, V. Zaccaria, and C. Silvano, "Evaluating Orthogonality between Application Auto-tuning and Run-time Resource Management for Adaptive OpenCL Applications," in *IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2014.
- [4] D. Gadioli, G. Palermo, and C. Silvano, "Application Autotuning to Support Runtime Adaptivity in Multicore Architectures," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2015.
- [5] S. Wildermann, T. Ziermann, and J. Teich, "Game-theoretic Analysis of Decentralized Core Allocation Schemes on Many-core Systems," in *Design, Automation Test in Europe*, 2013.
- [6] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting, *Invasive Computing: An Overview*. Springer, 2011.
- [7] A. Singh, C. Leech, K. R. Basireddy, B. Al-Hashimi, and G. Merrett, "Learning-based Run-time Power and Energy Management of Multi/Many-core Systems: Current and Future Trends," *Journal of Low Power Electronics*, vol. 13, no. 3, 2017.
- [8] H. Shen, J. Lu, and Q. Qiu, "Learning Based DVFS for Simultaneous Temperature, Performance and Energy Management," in *International Symposium on Quality Electronic Design*, 2012.
- [9] D.-C. Juan and D. Marculescu, "Power-aware Performance Increase via Core/Uncore Reinforcement Control for Chip-multiprocessors," in *ACM/IEEE International Symposium on Low Power Electronics and Design*, 2012.

- [10] H. Shen and Q. Qiu, "Contention Aware Frequency Scaling on CMPs with Guaranteed Quality of Service," in *Design, Automation Test in Europe*, 2014.
- [11] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, "Learning Transfer-based Adaptive Energy Minimization in Embedded Systems," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 35, no. 6, 2016.
- [12] Z. Wang, Z. Tian, J. Xu, R. K. V. Maeda, H. Li, P. Yang, Z. Wang, L. H. K. Duong, Z. Wang, and X. Chen, "Modular Reinforcement Learning for Self-adaptive Energy Efficiency Optimization in Multicore System," in *Asia and South Pacific Design Automation Conference*, 2017.
- [13] G.-Y. Pan, J.-Y. Jou, and B.-C. Lai, "Scalable Power Management Using Multilevel Reinforcement Learning for Multiprocessors," *ACM Transactions on Design Automation of Electronic Systems*, vol. 19, no. 4, 2014.
- [14] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps," in *IEEE/ACM International Symposium on Microarchitecture*, 2011.
- [15] H. Sasaki, S. Imamura, and K. Inoue, "Coordinated Power-performance Optimization in Manycores," in *International Conference on Parallel Architectures and Compilation Techniques*, 2013.
- [16] S. Sridharan, G. Gupta, and G. S. Sohi, "Adaptive, Efficient, Parallel Execution of Parallel Programs," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014.
- [17] R. Ye and Q. Xu, "Learning-based Power Management for Multi-core Processors via Idle Period Manipulation," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 33, no. 7, 2014.
- [18] A. Das, B. M. Al-Hashimi, and G. V. Merrett, "Adaptive and Hierarchical Runtime Manager for Energy-aware Thermal Management of Embedded Systems," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 2, 2016.
- [19] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi, "Adaptive Energy Minimization of Embedded Heterogeneous Systems using Regression-based Learning," in *International Workshop on Power and Timing Modeling, Optimization and Simulation*, 2015.
- [20] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE)," in *International Symposium on Computer Architecture*, 2012.
- [21] D. Gadioli, S. Libutti, G. Massari, E. Paone, M. Scandale, P. Bellasi, G. Palermo, V. Zaccaria, G. Agosta, W. Fornaciari, and C. Silvano, "OpenCL Application Auto-tuning and Run-time Resource Management for Multi-core Platforms," in *IEEE International Symposium on Parallel and Distributed Processing with Applications*, 2014.
- [22] Y. G. Kim, M. Kim, and S. W. Chung, "Enhancing Energy Efficiency of Multimedia Applications in Heterogeneous Mobile Multi-core Processors," *IEEE Transactions on Computers*, vol. 66, no. 11, 2017.
- [23] F. Gong, L. Ju, D. Zhang, M. Zhao, and Z. Jia, "Cooperative DVFS for Energy-efficient HEVC Decoding on Embedded CPU-GPU Architecture," in *ACM/EDAC/IEEE Design Automation Conference*, 2017.
- [24] P. Lama, Y. Guo, C. Jiang, and X. Zhou, "Autonomic Performance and Power Control for Co-located Web Applications in Virtualized Datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, 2016.
- [25] J. Ma, G. Yan, Y. Han, and X. Li, "An Analytical Framework for Estimating Scale-out and Scale-up Power Efficiency of Heterogeneous Manycores," *IEEE Transactions on Computers*, vol. 65, no. 2, 2016.
- [26] M. Otoom, P. Trancoso, H. Almasaeid, and M. Alzubaidi, "Scalable and Dynamic Global Power Management for Multicore Chips," in *Workshop on Parallel Programming and Run-time Management Techniques for Many-core Architectures*, 2015.
- [27] X. Lin, Y. Wang, and M. Pedram, "A Reinforcement Learning-based Power Management Framework for Green Computing Data Centers," in *IEEE International Conference on Cloud Engineering*, 2016.
- [28] Y. Wen, Z. Wang, and M. F. P. O'Boyle, "Smart Multi-task Scheduling for OpenCL Programs on CPU/GPU Heterogeneous Platforms," in *International Conference on High Performance Computing*, 2014.
- [29] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic Knobs for Responsive Power-aware Computing," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011.
- [30] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, "A Generalized Software Framework for Accurate and Efficient Management of Performance Goals," in *International Conference on Embedded Software*, 2013.
- [31] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, "Proactive Control of Approximate Programs," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.
- [32] V. Vassiliadis, C. Chaliou, K. Parasyris, C. D. Antonopoulos, S. Lalis, N. Bellas, H. Vandierendonck, and D. S. Nikolopoulos, "Exploiting Significance of Computations for Energy-constrained Approximate Computing," *International Journal of Parallel Programming*, vol. 44, no. 5, 2016.
- [33] H. Hoffmann, J. Eastepp, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application Heartbeats: A Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments," in *International Conference on Autonomic Computing*, 2010.
- [34] A. Baldassari, C. Bolchini, and A. Miele, "A Dynamic Reliability Management Framework for Heterogeneous Multicore Systems," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2017.
- [35] F. Gaspar, L. Taniça, P. Tomás, A. Ilic, and L. Sousa, "A Framework for Application-guided Task Management on Heterogeneous Embedded Systems," *ACM Transactions on Architecture and Code Optimization*, vol. 12, no. 4, 2015.
- [36] V. M. Weaver, D. Terpstra, H. McCraw, M. Johnson, K. Kasichayanula, J. Ralph, J. Nelson, P. Mucci, T. Mohan, and S. Moore, "Papi 5: Measuring Power, Energy, and the Cloud," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2013.
- [37] B. G. Pinto, L. M. de Souza Xavier, R. M. Favaretto, and G. G. H. Cavallheiro, "Um Framework para Monitoração do Consumo Energético em Arquiteturas Multicore," *Revista Brasileira de Computação Aplicada*, vol. 7, no. 3, 2015.
- [38] H. J. Curnow and B. A. Wichmann, "A Synthetic Benchmark," *The Computer Journal*, vol. 19, no. 1, 1976.
- [39] L. A. Maeda-Nunez, A. K. Das, R. A. Shafik, G. V. Merrett, and B. M. Al-Hashimi, "PoGo: An Application-specific Adaptive Energy Minimisation Approach for Embedded Systems," in *HiPEAC Workshop on Energy Efficiency with Heterogeneous Computing*, 2015.
- [40] A. S. Fathabadi, L. A. Maeda-Nunez, M. J. Butler, B. M. Al-Hashimi, and G. V. Merrett, "Towards Automatic Code Generation of Run-time Power Management for Embedded Systems Using Formal Methods," in *International Symposium on Embedded Multicore/Many-core Systems-on-chip*, 2015.
- [41] B. K. Reddy, A. K. Singh, G. V. Merrett, and B. M. Al-Hashimi, "ITMD: Run-time Management of Concurrent Multi-threaded Applications on Heterogeneous Multi-cores," in *Design, Automation and Test in Europe*, 2017.



Charles Leech (M16) is a Senior Research Assistant at the University of Southampton, where he is completing a PhD and received his BEng degree (1st class Hons) in Electronic Engineering degree in 2013. His work focuses on the optimisation of computer vision applications on heterogeneous embedded systems and the development of software frameworks for runtime management. He has interests in approximate computing, stereo vision and machine learning for embedded devices.



Graeme M. Bragg received an MEng (Hons) in Electronic Engineering in 2012 and a PhD in 2017 from the University of Southampton. He is a Research Fellow on the PRiME project focussing on applications and demonstrators. He has previously worked on the Glacsweb and Mountain Sensing projects investigating IoT environmental sensor networks. His research interests include environmental sensor networks, low-power networks and heterogeneous multi-core systems.



James J. Davis is a Research Associate at Imperial College London. His research concerns the runtime monitoring and adaption of digital electronic hardware for energy efficiency and reliability. James received his PhD in Electrical and Electronic Engineering from Imperial College London. He is a Member of the IEEE and ACM.



Geoff V. Merrett (GSM06-M09) received the BEng degree (Hons) in Electronic Engineering and the PhD degree from the University of Southampton, UK, in 2004 and 2009, respectively. He was appointed as a Lecturer in energy-efficient electronic systems at the University of Southampton in 2008, and was promoted to Associate Professor in 2014. He has published over 100 scientific papers in journals and refereed conference proceedings.



George A. Constantinides is the Royal Academy of Engineering/Imagination Technologies Research Professor of Digital Computation and Head of Imperial College London's Circuits and Systems research group. George received his PhD in Electrical and Electronic Engineering from Imperial College London. He is a Senior Member of the IEEE and a Fellow of the BCS.



Bashir M. Al-Hashimi (M99-SM01-F09) is a Professor of Computer Engineering and Dean of the Faculty of Physical Sciences and Engineering at University of Southampton, UK. He is the ARM Professor of Computer Engineering and Co-director of the ARM-ECS Research Centre. His research interests include methods and tools for low-power design and test of embedded computing systems. He has published over 300 technical papers, authored or co-authored five books and has graduated 33 PhD students.

Real-time Room Occupancy Estimation with Bayesian Machine Learning using a Single PIR Sensor and Microcontroller

Charles Leech
ARM, Cambridge
University of Southampton, UK

Yordan P. Raykov
NCRG, Aston University, UK

Emre Ozer
ARM
Cambridge, UK

Geoff V. Merrett
University of Southampton, UK

Abstract—This paper presents the implementation and deployment of a compute/memory intensive non-parametric Bayesian machine learning algorithm on a microcontroller unit (MCU) to estimate room occupancy in a Smart Room using a single analogue PIR sensor. We envisage an IoT device consisting of a resource-constrained MCU, PIR sensor and a battery running the occupancy estimation algorithm and operating over days or months without recharging or replacing the battery. Both hardware-independent and hardware-dependent optimizations are performed to reduce memory footprint and yet provide acceptable real-time performance while consuming less energy. We show a significant reduction in the on-chip memory usage in the MCUs by the algorithm through optimisation of the machine learning models and of the static memory footprint and dynamic memory usage. We also show that a low-end MCU does not meet the real-time requirements of the application without causing high average power consumption. However, a moderately high-performance MCU with a higher clock frequency and hardware floating-point unit provides 19x improvement in the execution time of the algorithm, better meeting the real-time specification of the application and reducing power consumption. Further, we estimate the battery lifetime of the IoT device if it operates continuously in a Smart Room. With a typical size battery, an IoT device consisting of a Cortex-M4F MCU and PIR sensor can operate for more than a month without replacement or recharging of the battery while running the compute-intensive Bayesian machine learning algorithm.

I. INTRODUCTION

Smart Building and Smart Workplace applications are becoming increasingly of interest as the focus of technological innovation shifts from mobile to IoT devices. For these systems to be effective they must be energy efficient, low cost and non-invasive, all whilst providing useful information. We present the implementation and the deployment analysis of a novel sensing system capable of occupancy estimation using only a single passive infrared (PIR) sensor.

This novel system was first introduced in [1] where we discussed in more detail the evaluation, the training and the statistical modeling involved in the problem of occupancy estimation with a single PIR. However, as [1] points out the novel system relies on sophisticated probabilistic modeling and therefore deployment on energy constrained IoT hardware can be a great challenge. Towards that end, we extend [1] and here we develop a framework of both hardware-dependent and algorithmic optimization steps to efficiently utilize the

resources available on a typical microcontroller (MCU). The undertaken optimization steps hardly affect the accuracy of the original system from [1] and we can estimate occupancy within one individual error bar with more than 80% accuracy.

The optimized model is deployed on two MCUs to estimate room occupancy natively on the devices in real time and we measure its performance, memory usage, energy consumption and battery life. To enable real-time estimations we build upon fast inference methods from [2] to derive online method for fast inference in probabilistic models. Furthermore, we demonstrate that the use of a floating point unit and hardware peripherals on the MCU leads to an almost 10 \times reduction in execution time of the algorithm. We also show that the memory optimization that we perform enable the system to operate within 12 KB of SRAM without impacting speed.

The paper is organized as follows: Section II briefly discusses the related work. Section III provides a brief overview of the iHMM model developed in our early work. Section IV describes the hardware-independent iHMM model resource optimizations. Section V describes the algorithm implementation and porting. Section VI presents the experimental setup, and Section VII describes the hardware-dependent optimizations. Section VIII presents the memory usage, performance, power consumption and battery lifetime estimation results, and finally Section IX concludes the paper.

II. RELATED WORK

Occupancy estimation is one of the five main tasks in human sensing and is an essential one to consider when building self-aware environments and smart building management systems [3]. There are methods that rely on data inference from cameras coupled with image processing algorithms ([4], [5], [6] and [7]), methods using multiple motion sensors at all entry and exit locations of a closed environments ([8], [9], [10], [11], [12] and [13]) and methods relying on historical patterns of movement across the environment based on data from motion or environmental sensor networks ([14], [15], [16]). PIR sensors have been dominantly used in the second of those categories. For example ([10] and [11]) showed how by placing three PIR sensors in a hallway, we can identify direction of movement and relative location of passing individuals. [12] used PIR sensors in combination with reed

switch door sensors placed at each doorway and [13] presented a similar approach but using only PIR sensors at all entries and exits.

In contrast to these approaches, [1] proposed using a single analogue PIR sensor as a monitoring device rather than simply counting entries or exits. The analogue PIR output is segmented using a flexible probability model and the patterns of motion which are the best descriptor of actual occupancy are identified. In effect, we have traded the complexity of image data with simpler data coupled with more complex and flexible modelling. In this way, simple single dimensional time series data generated from the PIR sensor can be used to learn some motion behaviours of interest from the data. This can be used for accurate occupancy estimation on its own, hence why [1] is able to estimate occupancy using a single sensor. However, while such an approach can reduce the cost and also the invasiveness of the system, it faces the challenge of learning a flexible Bayesian nonparametric model online, which is non-trivial on a resource-constrained MCU. To address this problem, we present several hardware-independent model optimizations in Section IV.

III. OVERVIEW OF OCCUPANCY ESTIMATION ALGORITHM

In this section we present a high level overview of the pre- and post-deployment stages of our algorithm for occupancy estimation. The first stage is performed before deployment to train the model and is described in more detail in [1]. We have collected PIR data for approximately 50 hours from various office meetings. We segment the raw analogue PIR data using an infinite hidden Markov model (iHMM) [17] with Laplace components and separate the data that is most descriptive of the room occupancy. This filtered data is then used to estimate a Laplace diversity parameter which is good indicator of the levels of motion and occupancy. A regression model is fitted to the different diversity parameters for each time window of 30 seconds and the regression is used to predict occupancy.

After training the iHMM and inferring the regression parameters, we implement a modified prediction process on the MCU. The trained iHMM parameters are condensed and transferred to the MCU. On the MCU a modified online learning version of the MAP-iHMM [2] is used to fit the iHMM to incoming streams of unseen PIR data, where the method also incorporates the effect of the previously trained parameters. Once we segment the new data, we update the training iHMM parameters to incorporate the effect of the last seen data. The regression parameters are robust so they rarely need to be re-calculated.

IV. HARDWARE-INDEPENDENT MODEL RESOURCE OPTIMIZATIONS

The iHMM used in [1] is a powerful probabilistic model that can capture arbitrarily complex time dependent patterns in entirely unsupervised way. Applications of models such as the iHMM has been limited thanks to the computationally expensive inference methods they typically require: exhaustive

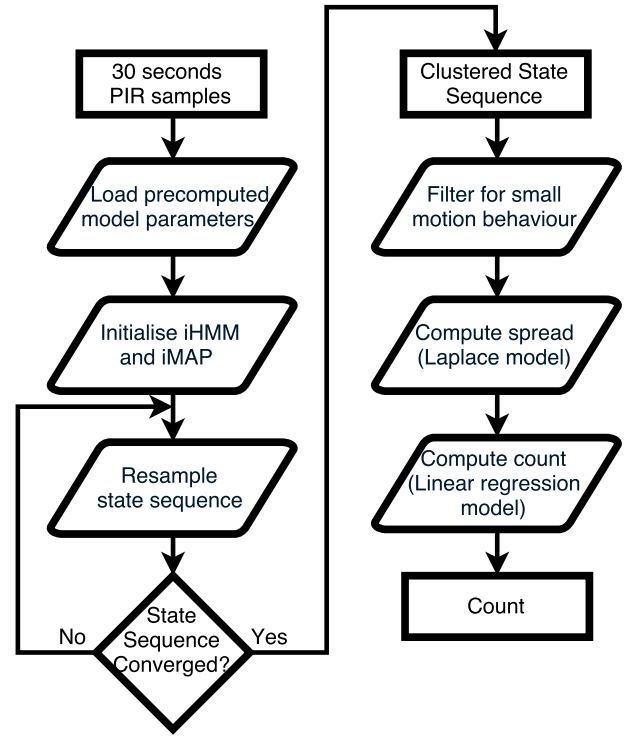


Fig. 1. Block diagram of the stages of the room occupancy estimation algorithm performed during deployment on the MCU.

Markov Chain Monte Carlo (MCMC) sampling. MCMC methods are probabilistic therefore aim to find the complete likelihood distribution of the fitted model. This involves storing orders of magnitude more parameters than often needed. For example, fitting an iHMM on 5 minutes of PIR output using MCMC would require storing approximately $N * K * I + 2K$ parameters, where: N is large (11500 for 5 minutes of data) as is the number of data points; K is the number of states found in the time series (usually 3–6); I is the length of the chain (can vary between 120 – 1000 iterations). Approximate Variational Bayes inference methods have been proposed ([18], [19]) which improve mixing drastically, but stochastic Variational methods [19] require knowledge of the size of the data to be processed a priori and storing data dependent number of variational distributions. [18] reduces the memory overhead when used on a batch of data, but in streaming applications suffers from the same problems as [19].

To fit the iHMM to streams of PIR data we build upon a recent learning algorithm called MAP-iHMM from [2], [20]. Iterative MAP methods are convenient as they converge orders of magnitude faster than MCMC methods and return only the most likely segmentation of the data rather than a full posterior distribution; this makes them quite memory efficient. Using the example from above for fitting an iHMM on 5 minutes of PIR output data, this time using iterative MAP, the number of parameters to store would be approximately $N + 2 * K$. However if the system is deployed, as we monitor more and

more PIR data, N becomes too large to store. Therefore, we derive from the MAP-iHMM an online streaming algorithm that processes batches of sensor data as it is sampled. Once converged, the method only updates the small number of parameters and discards the raw data before receiving the next batch. One challenge is that [20] and a lot of the efficient inference algorithms for Bayesian nonparametric models use a collapsed (Rao-Blackwellized) representation of the iHMM for faster and more robust convergence (standard practice in Bayesian modelling). This representation introduces dependencies between the model parameters requiring us to keep some explicit parameters for each sensor output point that influences the segmentation. To overcome this issue, after processing each window of data, we recover an approximation of the complete representation of the iHMM with an explicit distribution available for the whole parameter space. For example, the memory footprint of the collapsed representation of the model after training on 50 hours of data is approximately 228 MB, compared to few hundred KB when recovering a non-collapsed model representation.

This is possible because future data is independent of historic data given the complete model representation, while this is not the case for the collapsed representation. Therefore the complete trained model rigorously and unbiasedly can be used as a starting point for the clustering on the next window of sensor data. The trained form of the model involves updating only $(K+1)*(K+2)+2$ parameters after processing each time window, which is sufficiently more compact. Note that those few parameters still incorporate the knowledge gained by observing and segmenting many hours of PIR data and so the model is still capable of segmenting behaviours of interest despite the reduced memory footprint. This will allow the model to dynamically update the model parameters after its deployment to the microcontroller unit (MCU).

V. ALGORITHM IMPLEMENTATION AND PORTING

Initially, a series of Matlab functions characterise the iHMM and iMAP processes, such as the clustering of data samples and the Bayesian resampling of hyperparameters. A top-level encapsulation function connects these ML functions with the Laplace model and regression parameters to apply the algorithm to occupancy estimation.

Matlab was used to experiment with the theoretical concepts of the algorithm without exposing hardware constraints. However, many microprocessor compilers cannot interpret such high-level languages and therefore the Matlab code must be translated to C and C++ such that it can be compiled into a binary and executed directly by the MCU.

Porting can be achieved through manual programming or automatic compiler-style translation using Matlab Coder [21]. Manual programming is the simpler option however it does not scale well with increasing code size and requires detailed knowledge of the syntax and constructs of both languages. Automatic translation avoids both of these barriers, however in the case of Matlab Coder, extensive code preparation is required involving the addition of common syntactic elements

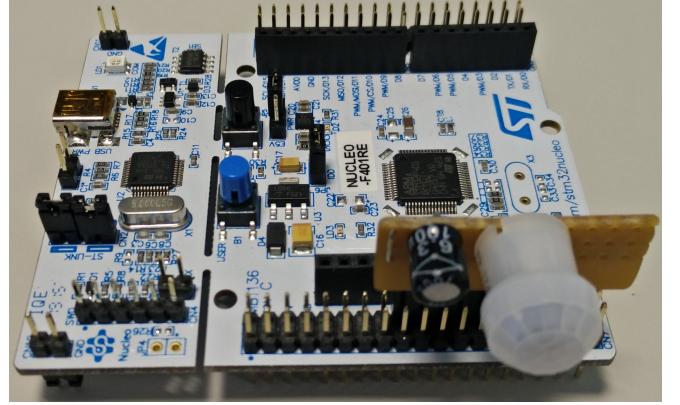


Fig. 2. Photo of the experimental setup consisting of the ST Nucleo-401RE MCU board and PIR sensor connected to the ADC through the Arduino header.

from both languages in order to guide the translation tool. This negates much of the benefit that would be afforded by the process. Furthermore, due to the algorithmic optimisations described in the previous section, the algorithm occupies a relatively small code-base and as a result the porting of each function was realistic through manual programming.

Whenever possible, Matlab standard toolbox functions have been replaced by equivalents from C and C++ standard libraries, with functional testing performed to ensure that the algorithm results remain the same for the same input dataset. Moreover, there are no dependencies on libraries introduced with the C++11 standard to improve the portability of the code as support for newer C++ standards varies across toolchains and development environments. In the case where no standard library function exists as the replacement for a Matlab function, a custom function was developed from a theoretical basis or by combining other standard functions. Only one case for this was required in the algorithm in the generation of the gamma random distribution when re-sampling the values of Beta from the iHMM. Here the Ziggurat method is used, developed by Marsaglia and Tsang [22], based on the cube of scaled normal variates from a normal distribution, which is a standard library function.

VI. EXPERIMENTAL SETUP

We begin our experimentation with the ARM Cortex-M0 based ST Nucleo-F070RB, a highly resource-constrained MCU, to test the memory and computational boundaries of the algorithm. The specification of the MCU is shown in Table I where we highlight the critical features, including clock frequency, memory capacity and the peripherals that we will utilise. A Panasonic NaPiOn AMN21111 PIR sensor is connected to one of the ADC inputs on the Arduino header of the MCU, as shown in Figure 2. The PIR sensor continuously produces a single dimensional analogue signal which is sampled at 50 Hz by the 12-bit ADC, this produces 1500 samples per 30 second recording interval, which the algorithm then processes. Samples are recorded as integers

TABLE I
SPECIFICATIONS OF THE CORTEX-M0 AND M4 MCUS USED FOR EXPERIMENTATION. MEMORY NUMBERS SHOW SRAM SIZE WITH FLASH SIZE IN SQUARE BRACKETS.

Board name	CPU	Clock Modes (MHz)	Memory (KB)	Peripherals Used	Technology Node (nm) [23]	Operating Voltage (V)
Nucleo-F070RB	Cortex-M0	8, 24, 48	16 [128]	ADC, DMA, TIM3	180	2.4
Nucleo-F401RE	Cortex-M4F	84	96 [512]	ADC1, DMA2, TIM2	90	1.7

but converted to floating point values in the range 0.0 to 1.0 before processing.

We make extensive use of the hardware peripherals of the MCU, through the STM hardware abstraction library (HAL). A timer with a 50Hz period triggers conversion events in the ADC via an interrupt. We use DMA to transfer sampled data from the ADC to SRAM without the involvement of the CPU, allowing the CPU to sleep or perform other tasks during sampling periods.

The completion of sampling is signalled by a DMA transfer complete interrupt when the data buffer in SRAM is filled. A DMA interrupt service routine (ISR) links the interrupt to an ADC conversion complete callback function from where the count estimation algorithm is called. The configuration and interaction of the HAL components is illustrated in Figure 3.

The board is connected to a laptop via a USB cable solely for power supply and programming reasons. All data collection and processing for the algorithm is performed locally on the MCU. Furthermore, the board can be configured to receive power from a battery source connected to it and, when deployed in a Smart Room context, the program will be automatically loaded from the flash memory on start-up, removing all need for an external connection.

An LCD display is mounted to the board via the Arduino header, allowing the occupancy count estimation from the algorithm to be displayed on the device. During development, debugging information was communicated back to a PC via the USB cable and displayed via a serial terminal.

Despite the particular choice of MCU, in principle the C/C++ code for the algorithm can be compiled and executed on any microprocessor with a similar specification.

VII. HARDWARE-DEPENDENT OPTIMIZATIONS

The occupancy estimation algorithm is required to have low computational complexity and memory requirements to allow real-time processing and deployment on the MCU.

Strategies to manage the memory requirements of the algorithm from a theoretical perspective are described in detail in Section IV. These primarily consist of the pre-deployment construction of a non-collapsed iHMM representation to reduce the memory footprint of the algorithm from the order of MB to KB and the use of an iterative MAP inference method due to the fact that it is an order of magnitude faster than MCMC methods when fitting the iHMM to streams of PIR data, also greatly reducing the expected computation time.

In deployment of the algorithm, additional optimisations are made to further reduce the memory and computational requirements from an implementation perspective. Smaller

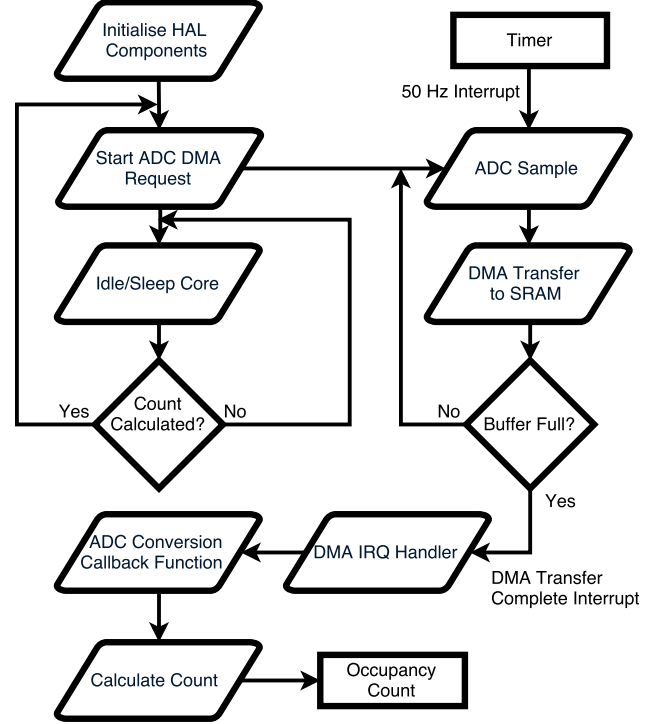


Fig. 3. The configuration of HAL components on the MCU

data type sizes were used throughout the algorithm to reduce the size of memory required to hold intermediate variables. The double data type offers the greatest precision to represent real numbers but requires twice as many bytes for storage. As a result, all floating-point numbers are represented in single-precision, with negligible loss in accuracy, reducing the size of stored data from 12 to 6 KB for 1500 samples. In addition, for integers that are known to be within the range -128 to $+127$, their type is reduced to characters, reducing memory allocation by a factor of four with no effect on data values. This has the greatest affect in the storage of the cluster assignments for the hidden state sequence; from 6 to 1.5 KB.

Memory tracing was used to more accurately analyse the memory behaviour of the algorithm throughout its execution. Tracing was performed using an mbed OS API whereby calls to standard C memory management functions were wrapped and intercepted to identify when (de-)allocations were made. In addition, heap statistics were recorded online to monitor the maximum heap allocation reached in the algorithm, determining the minimum SRAM size required by the MCU. Memory reduction techniques have been used throughout the algorithm

such as lowering the scope of intermediate variables to discard temporary data and pruning data structures to pass only the minimum amount of data between functions. Updates are made directly to the hidden state sequence for each iteration so that only the final clustered states are returned by the model.

Sampling of the PIR sensor and execution of the algorithm is performed simultaneously through additional configuration of the hardware peripherals on the MCU, as outlined in the previous section, to enable the transfer of sensor data from the ADC to SRAM using the DMA module. This allows us to overlay each data processing operation with the beginning of data collection for the following estimation period, meaning that a count estimate can be obtained at regular intervals and still be based on a full 30 second sampling period. In addition, after sampling, the CPU can enter a lower power mode if has no other tasks to perform, whilst waiting for the next sampling period to complete.

VIII. EXPERIMENTAL RESULTS

A. Memory Usage and Real-time Performance

To evaluate the performance of the algorithm on the Cortex-M0 MCU, we record new PIR sensor data and apply the algorithm to 30 second segments. This directly emulates the operation expected when the MCU is placed in a meeting environment. The accuracy of the translated version of the algorithm has been verified against the original Matlab program by testing with the same data sets collected from meetings of known occupancy. The percentage of time windows where the predicted number of occupants was within ± 1 and ± 2 matches those presented in the evaluation of [1]. There was no impact on the estimation accuracy of the algorithm from losses in precision due to the conversion of real numbers from double to single precision. This rounding rarely causes cluster assignments to change and the regression parameters are robust after training. All other computation and memory optimizations do not impact data values.

To analyse memory consumption, we run the program with memory tracing and heap statistic recording enabled. The peak heap allocation was recorded as 0.7 KB. To find the total memory consumption we must include statically allocated global data, including the memory blocks outlined in the previous section, which accounts for 9.63 KB in our program, giving a combined total memory consumption of 10.33 KB. This is below the 16 KB available on the Cortex-M0 MCU.

To evaluate computational demand, we use execution time as a metric, and test how long the Cortex-M0 based MCU takes to calculate the estimated occupancy count, with the algorithmic optimisations described, across the three frequency modes in table I. We measure execution time as approximately 22 seconds. The main reason for low performance is because the iHMM model uses a significant number of floating-point operations, which are emulated by software since the Cortex-M0 CPU does not have a hardware floating-point unit (FPU), and also the clock frequency of the Cortex-M0 is comparatively low, i.e. 48 MHz.

TABLE II
RESULTS FOR COMPUTATION TIME, MEMORY CONSUMPTION AND CURRENT CONSUMPTION OF THE ALGORITHM ON THE CORTEx-M0 AND M4 BASED MCUs. MEMORY CONSUMPTION IS DIVIDED INTO SRAM AND FLASH, WITH THE LATER SHOWN IN SQUARE BRACKETS.

ARM Platform	Execution Time (s)	Memory Requirement (KB)
Cortex-M0	22	10.33 [11.70]
Cortex-M4	9.55	10.36 [12.03]
Cortex-M4 + FPU	1.15	10.36 [11.24]

In order to improve the execution time, we experiment with a higher performance Cortex-M4 based MCU. The Cortex-M4 CPU has a single-precision FPU that can perform floating-point calculations in hardware, dramatically increasing the speed at which our algorithm executes. This speed-up is compounded by the higher clock frequency of the Cortex-M4 (1.75x faster than Cortex-M0) which accelerates all instructions. The execution time is reduced from 22 s to 1.15 s. We disable the FPU in the Cortex-M4 to evaluate how the FPU improves execution time. We recompile the code, with the floating-point instructions being emulated, and observe that the algorithm executes in 9.55 s, which reflects the increase in clock frequency. The memory usage and execution time results are summarized in Table II. The same hardware peripherals (ADC, DMA and timer) are used in both Cortex-M0 and Cortex-M4 experiments, and the code for the algorithm remains unchanged.

B. Power Consumption and Battery Lifetime

We estimate the power consumption and battery lifetime of the platform (MCU and PIR sensor) based on current consumption and operating voltage numbers provided in the MCU and PIR datasheets and using the STM32CubeMX tool by STMicroelectronics [24]. While data is being sampled, the CPU is put into sleep mode which clock-gates the CPU and reduces the current consumption. For example, it is 9.58 mA for the Cortex-M4F MCU at 84 MHz and 7.53 mA for the Cortex-M0 MCU at 48 MHz. When data sampling is complete, the CPU is returned to run mode and, together with the continued use of the peripheral components, the current consumption of the Cortex-M4F and M0 MCUs rise to 17.18 mA and 11.50 mA, respectively. With operating voltages of 2.4 and 1.7 V, the average power consumption of the Cortex-M4F and Cortex-M0 platforms are 46.36 and 18.50 mW, respectively.

Finally, we estimate the battery lifetime of the platform if it were battery-powered and deployed in a meeting room environment to run continuously. Battery lifetimes are calculated from the average power consumption over the sampling and execution periods. We use a Lithium Polymer (LiPo) battery with a 2200 mAh capacity. The estimated lifetimes of the platform are shown in Figure 4, for the Cortex-M0 operating at three frequencies and the Cortex-M4 with its FPU enabled and disabled. Battery lifetime and execution time decrease for the Cortex-M0 platform as frequency increases. The Cortex-M4 MCU increases battery lifetime by 2.51x due to the

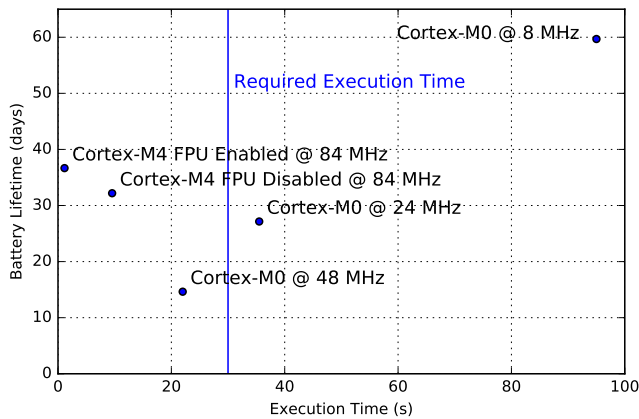


Fig. 4. Battery lifetime and execution time over a range of frequencies on the Cortex-M0 and for the FPU enabled and disabled on the Cortex-M4F. The vertical line shows the 30 second algorithm execution time requirement.

lower average power consumption whilst further reducing the execution time. The reduction in power consumption is due to the lower technology node of the Cortex-M4F MCU, which reduces dynamic current consumption, and a lower operating voltage.

IX. CONCLUSIONS

In this paper, we have investigated the deployment of a non-parametric Bayesian machine learning algorithm on a resource-constrained MCU for a room occupancy estimation application using a single analogue PIR sensor. Optimisations in several dimensions were performed to accommodate for reduced memory and computational capacity to produce an energy efficient and non-invasive solution with reduced hardware complexity. We have demonstrated a three order reduction in the memory requirement of the algorithm through hardware-independent optimisation of the machine learning models and an 8x reduction in memory utilisation on two MCUs, through analysis of the static memory footprint and dynamic memory behaviour of the implemented algorithm. We have shown that a low-end MCU did not meet the real-time service requirements for the application but a moderately high-performance MCU with a higher clock frequency and hardware floating-point unit provided 19.13x improvement in the execution time of the algorithm, meeting the real-time specification of the application. Further, we have estimated the power consumption of the IoT platform and battery lifetime if it operates indefinitely in a Smart Room application. With a typical battery size, the IoT platform consisting of a Cortex-M4F MCU and PIR sensor can operate for over 36 days without replacement or recharging the battery while running a compute-intensive self-learning algorithm.

REFERENCES

- [1] Y. P. Raykov *et al.*, "Predicting Room Occupancy with a Single Passive Infrared (PIR) Sensor Through Behavior Extraction," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '16. New York, NY, USA: ACM, 2016, pp. 1016–1027.
- [2] Y. P. Raykov, A. Boukoulalas, and M. A. Little, "Simple approximate MAP inference for Dirichlet processes mixtures," *Electron. J. Statist.*, vol. 10, no. 2, pp. 3548–3578, 2016.
- [3] T. Teixeira, G. Dublon, and A. Savvides, "A survey of human-sensing: Methods for detecting presence, count, location, track," and Identity. Technical report, ENALAB, Yale University, Tech. Rep., 2010.
- [4] V. Lempitsky and A. Zisserman, "Learning to count objects in images," in *Advances in Neural Information Processing Systems*, 2010, pp. 1324–1332.
- [5] T. Van Oosterhout, S. Bakkes, and B. J. Kröse, "Head Detection in Stereo Data for People Counting and Segmentation," in *VISAPP*, 2011, pp. 620–625.
- [6] A. B. Chan and N. Vasconcelos, "Counting People with Low-level Features and Bayesian Regression," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 2160–2177, 2012.
- [7] D. B. Yang, H. H. González-Baños, and L. J. Guibas, "Counting people in crowds with a real-time network of simple image sensors," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE, 2003, pp. 122–129.
- [8] K. Hashimoto *et al.*, "People count system using multi-sensing application," in *Solid State Sensors and Actuators, 1997. TRANSDUCERS'97 Chicago., 1997 International Conference on*, vol. 2. IEEE, 1997, pp. 1291–1294.
- [9] P. Zappi, E. Farella, and L. Benini, "Enhancing the spatial resolution of presence detection in a PIR based wireless surveillance network," in *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*. IEEE, 2007, pp. 295–300.
- [10] J. Yun and S.-S. Lee, "Human movement detection and identification using pyroelectric infrared sensors," *Sensors*, vol. 14, no. 5, pp. 8057–8081, 2014.
- [11] P. Zappi, E. Farella, and L. Benini, "Tracking motion direction and distance with pyroelectric IR sensors," *IEEE Sensors Journal*, vol. 10, no. 9, pp. 1486–1494, 2010.
- [12] Y. Agarwal *et al.*, "Occupancy-driven Energy Management for Smart Building Automation," in *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*. ACM, 2010, pp. 1–6.
- [13] F. Wahl, M. Milenkovic, and O. Amft, "A distributed PIR-based approach for estimating people count in office environments," in *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*. IEEE, 2012, pp. 640–647.
- [14] K. P. Lam *et al.*, "Occupancy detection through an extensive environmental sensor network in an open-plan office building," *IBPSA Building Simulation*, vol. 145, pp. 1452–1459, 2009.
- [15] R. H. Dodier, G. P. Henze, D. K. Tiller, and X. Guo, "Building Occupancy Detection Through Sensor Belief Networks," *Energy and buildings*, vol. 38, no. 9, pp. 1033–1043, 2006.
- [16] A. Khan *et al.*, "Occupancy Monitoring using Environmental & Context Sensors and a Hierarchical Analysis Framework," in *BuildSys@ SenSys*, 2014, pp. 90–99.
- [17] M. J. Beal, Z. Ghahramani, and C. E. Rasmussen, "The Infinite Hidden Markov Model," in *Advances in neural information processing systems*, 2001, pp. 577–584.
- [18] M. C. Hughes, W. T. Stephenson, and E. Sudderth, "Scalable Adaptation of State Complexity for Nonparametric Hidden Markov Models," in *Advances in Neural Information Processing Systems*, 2015, pp. 1198–1206.
- [19] N. Foti, J. Xu, D. Laird, and E. Fox, "Stochastic variational inference for hidden Markov models," in *Advances in Neural Information Processing Systems*, 2014, pp. 3599–3607.
- [20] Y. P. Raykov, A. Boukoulalas, and M. A. Little, "Iterative collapsed MAP inference for Bayesian nonparametrics," *NIPS 2015 Workshop on Bayesian Nonparametrics: The Next Generation*, 2015.
- [21] T. M. Inc. Matlab coder - generate c and c++ code from matlab code. Online. The MathWorks Inc. Natick, Massachusetts.
- [22] G. Marsaglia and W. W. Tsang, "A Simple Method for Generating Gamma Variables," *ACM Trans. Math. Softw.*, vol. 26, no. 3, pp. 363–372, Sep. 2000.
- [23] AN4435 Application note, STMicroelectronics, March 2016, rev 3.
- [24] Stm32cubemx - stm32cube initialization code generator. Online. STMicroelectronics. V4.18.0.

Bibliography

- [1] D. Firesmith. (2017, Aug.) Multicore processing. Online. Software Engineering Institute, Carnegie Mellon University. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2017/08/multicore-processing.html
- [2] ARM, *ARM Cortex-A15 MPCore Processor Technical Reference Manual*, ARM, June 2013, pages 53 - 63. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0438i/DDI0438I.cortex_a15_r4p0_trm.pdf
- [3] A. Sinkar, H. Ghasemi, M. Schulte, U. Karpuzcu, and N. Kim, “Low-Cost Per-Core Voltage Domain Support for Power-Constrained High-Performance Processors,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 99, pp. 1–1, 2013.
- [4] P. Greenhalgh, “big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7,” ARM, Tech. Rep., September 2011.
- [5] A. Lukefahr, S. Padmanabha, R. Das, F. Sleiman, R. Dreslinski, T. Wenisch, and S. Mahlke, “Composite Cores: Pushing Heterogeneity Into a Core,” in *45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012, pp. 317–328.
- [6] R. Rodrigues, A. Annamalai, I. Koren, S. Kundu, and O. Khan, “Performance Per Watt Benefits of Dynamic Core Morphing in Asymmetric Multicores,” in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, 2011, pp. 121–130.
- [7] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, “The Landscape of Parallel Computing Research: A View from Berkeley,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006.
- [8] Y.-K. Chen, J. Chhugani, P. Dubey, C. Hughes, D. Kim, S. Kumar, V. Lee, A. Nguyen, and M. Smelyanskiy, “Convergence of Recognition, Mining, and Synthesis Workloads and Its Implications,” *Proceedings of the IEEE*, vol. 96, no. 5, pp. 790–807, May 2008.
- [9] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiawicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, “A View of the Parallel Computing Landscape,” *Communications of the ACM*, vol. 52, no. 10, pp. 56–67, Oct. 2009.

- [10] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, “Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps,” in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 175–185.
- [11] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, “Dynamic Knobs for Responsive Power-aware Computing,” in *Proceedings of 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2011.
- [12] D. Gadioli, G. Palermo, and C. Silvano, “Application Autotuning to Support Runtime Adaptivity in Multicore Architectures,” in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2015.
- [13] S. T. Fleming and D. B. Thomas, “Heterogeneous Heartbeats: A framework for dynamic management of Autonomous SoCs,” in *24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014.
- [14] E. Paone, D. Gadioli, G. Palermo, V. Zaccaria, and C. Silvano, “Evaluating orthogonality between application auto-tuning and run-time resource management for adaptive OpenCL applications,” in *IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, Jun. 2014, pp. 161–168.
- [15] G. Chrysos. (2012, Nov.) Intel Xeon Phi X100 Family Coprocessor - the Architecture. Online. Intel Corporation. [Online]. Available: <https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner>
- [16] L. Q. Nguyen, *Intel Xeon Phi™ Coprocessor Developer’s Quick Start Guide*, 1st ed., Intel, 2014. [Online]. Available: <https://software.intel.com/sites/default/files/managed/26/d6/Intel.Xeon.Phi.Quick.Start.Developers.Guide-MPSS-3.4.pdf>
- [17] Hardkernel. (2013) Odroid-XU3 Board Detail. [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=g140448267127&tab_idx=2
- [18] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
- [19] Itseez. Camera Calibration and 3D Reconstruction. Online. Itseez. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
- [20] G. Bradski and A. Kaehler, *Learning OpenCV 3*. O’Reilly Media, 2016.
- [21] D. Brodowski, *CPU frequency and voltage scaling code in the Linux kernel*, Linux Kernel Organization, 4 2017. [Online]. Available: <https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt>
- [22] D. Scharstein and R. Szeliski, “A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms,” *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, Apr. 2002.
- [23] H. Sutter, “The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software,” *Dr. Dobbs’s Journal*, vol. 30, no. 3, March 2005.

- [24] H. Esmailzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011, pp. 365–376.
- [25] C. Leech, C. K. Vala, A. Acharyya, S. Yang, G. V. Merrett, and B. Al-Hashimi, "Run-time Performance and Power Optimization of Parallel Disparity Estimation on Many-core Platforms," *ACM Transactions on Embedded Computing Systems*, August 2017. [Online]. Available: <https://eprints.soton.ac.uk/413390/>
- [26] V. Tenentes, C. Leech, G. Bragg, G. Merrett, B. Al-Hashimi, H. Amrouch, J. Henkel, and S. Das, "Hardware and Software Innovations in Energy-efficient System-reliability Monitoring," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*. IEEE, August 2017. [Online]. Available: <https://eprints.soton.ac.uk/413034/>
- [27] A. Singh, C. Leech, K. R. Basireddy, B. Al-Hashimi, and G. V. Merrett, "Learning-based Run-time Power and Energy Management of Multi/Many-core Systems: Current and Future Trends," *Journal of Low Power Electronics*, vol. 13, no. 3, 2017.
- [28] C. K. Vala, K. Immadisetty, A. Acharyya, C. Leech, V. Balagopal, G. V. Merrett, and B. M. Al-Hashimi, "High-Speed Low-Complexity Guided Image Filtering-Based Disparity Estimation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–12, 2017.
- [29] C. Leech and T. J. Kazmierski, "Energy Efficient Multi-Core Processing," *ELECTRONICS*, vol. 18, no. 1, pp. 3–10, June 2014. [Online]. Available: <https://eprints.soton.ac.uk/366664/>
- [30] C. Leech, Y. P. Raykov, E. Ozer, and G. V. Merrett, "Real-time Room Occupancy Estimation with Bayesian Machine learning using a Single PIR Sensor and Microcontroller," in *IEEE Sensors Applications Symposium (SAS)*, March 2017, pp. 1–6.
- [31] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 26–37, November 2009.
- [32] C. Isci, A. Buyuktosunoglu, C.-Y. Chen, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, 2006, pp. 347–358.
- [33] V. Hanumaiah and S. Vrudhula, "Energy-efficient Operation of Multi-core Processors by DVFS, Task Migration and Active Cooling," *Computers, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2012.
- [34] B. de Abreu Silva and V. Bonato, "Power/performance optimization in FPGA-based asymmetric multi-core systems," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 473–474.
- [35] K. Wonyoung, M. Gupta, G.-Y. Wei, and D. Brooks, "System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators," in *IEEE 14th International Symposium on High Performance Computer Architecture (HPCA)*, 2008, pp. 123–134.

- [36] H. Ghasemi, A. Sinkar, M. Schulte, and N. S. Kim, "Cost-effective Power Delivery to Support Per-core Voltage Domains for Power-constrained Processors," in *49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2012, pp. 56–61.
- [37] R. Kumar, V. Zyuban, and D. Tullsen, "Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling," in *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, 2005, pp. 408–419.
- [38] H. Zeng, J. Wang, G. Zhang, and W. Hu, "An Interconnect-Aware Power Efficient Cache Coherence Protocol for CMPs," in *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2008, pp. 1–11.
- [39] R. Basmadjian and H. De Meer, "Evaluating and modeling power consumption of multi-core processors," in *Third International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy)*, 2012, pp. 1–10.
- [40] D. . Woo and H.-H. Lee, "Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era," *Computer*, vol. 41, no. 12, pp. 24–31, 2008.
- [41] H. M. Waidyasooriya, Y. Takei, M. Hariyama, and M. Kameyama, "FPGA Implementation of Heterogeneous Multicore Platform with SIMD/MIMD Custom Accelerators," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, 2012, pp. 1339–1342.
- [42] B. de Abreu Silva, L. Cuminato, and V. Bonato, "Reducing the Overall Cache Miss Rate Using Different Cache Sizes for Heterogeneous Multi-core Processors," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2012, pp. 1–6.
- [43] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas, "Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance," in *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, 2004, pp. 64–75.
- [44] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen, "Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction," in *Microarchitecture (MICRO-36). Proceedings. 36th Annual IEEE/ACM International Symposium on*, 2003, pp. 81–92.
- [45] B. Jeff, "Advances in big.LITTLE Technology for Power and Energy Savings," ARM, Tech. Rep., September 2012.
- [46] A. Das, R. Rodrigues, I. Koren, and S. Kundu, "A Study on Performance Benefits of Core Morphing in an Asymmetric Multicore Processor," in *Computer Design (ICCD), IEEE International Conference on*, 2010, pp. 17–22.
- [47] Energy Aware Scheduling. Online. ARM. [Online]. Available: <https://developer.arm.com/open-source/energy-aware-scheduling>
- [48] A. Kucheria. (2015) Energy-Aware Scheduling (EAS) Project. [Online]. Available: <https://www.linaro.org/blog/core-dump/energy-aware-scheduling-eas-project/>
- [49] N. Pitre. (2014, Jun.) Teaching the scheduler about power management. [Online]. Available: <https://lwn.net/Articles/602479/>

- [50] M. Awan and S. Petters, "Energy-Aware Partitioning of Tasks onto a Heterogeneous Multi-core Platform," in *IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013, pp. 205–214.
- [51] M. Zhang, S. Tu, and Z. Chai, "PDSDL: A Dynamic System Description Language," in *International SoC Design Conference (ISOCC)*, vol. 01, 2008, pp. I–204–I–209.
- [52] A. Z. Jooya and M. Analoui, "Program Phase Detection in Heterogeneous Multi-Core Processors," in *14th International CSI Computer Conference (CSICC)*, 2009, pp. 219–224.
- [53] L. Sawalha and R. Barnes, "Energy-Efficient Phase-Aware Scheduling for Heterogeneous Multicore Processors," in *IEEE Green Technologies Conference*, 2012, pp. 1–6.
- [54] S. Saha, J. Deogun, and Y. Lu, "Adaptive Energy-Efficient Task Partitioning for Heterogeneous Multi-core Multiprocessor Real-Time Systems," in *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, 2012, pp. 147–153.
- [55] F. Calcado, S. Louise, V. David, and A. Merigot, "Efficient Use of Processing Cores on Heterogeneous Multicore Architecture," in *International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, 2009, pp. 669–674.
- [56] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schroder-Preikschat, and G. Snelting, "Invasive Computing: An Overview," in *Multiprocessor System-on-Chip*, M. Hubner and J. Becker, Eds. Springer New York, 2011, pp. 241–268.
- [57] J. Teich, A. Weichslgartner, B. Oechslein, and W. Schroder-Preikschat, "Invasive Computing - Concepts and Overheads," in *Forum on Specification and Design Languages (FDL)*, Sept 2012, pp. 217–224.
- [58] D. Newell, "Workloads, Scalability, and QoS Considerations in CMP Platforms," in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, April 2007, pp. xiii–xiii.
- [59] S. Huang and W. Feng, "Energy-Efficient Cluster Computing via Accurate Workload Characterization," in *9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, May 2009, pp. 68–75.
- [60] K. Manakul, P. Siripongwutikorn, S. See, and T. Achalakul, "Modeling Dwarfs for Workload Characterization," in *IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2012, pp. 776–781.
- [61] A. Maxiaguine, Y. Liu, S. Chakraborty, and W. T. Ooi, "Identifying "Representative" Workloads in Designing MpSoC Platforms for Media Processing," in *2nd Workshop on Embedded Systems for Real-Time Multimedia*, Sept 2004, pp. 41–46.
- [62] W. Alkohlani and J. Cook, "Towards Performance Predictive Application-Dependent Workload Characterization," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, Nov 2012, pp. 426–436.
- [63] Y. Shao and D. Brooks, "ISA-Independent Workload Characterization and its Implications for Specialized Architectures," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2013, pp. 245–255.

- [64] J. Held, J. Bautista, and S. Koehl, "From a Few Cores to Many: A Tera-scale Computing Research Overview," Intel White Paper, Tech. Rep., 2006. [Online]. Available: https://software.intel.com/sites/default/files/m/1/f/9/terascale_overview_paper.pdf
- [65] J. Lin, Y. Chen, E. Li, A. Jaleel, and Z. Tang, "Understanding the Memory Behavior of Emerging Multi-core Workloads," in *Eighth International Symposium on Parallel and Distributed Computing*, June 2009, pp. 153–160.
- [66] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. D. Kubiatowicz, E. A. Lee, N. Morgan, G. Necula, D. A. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. A. Yelick, "The Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View," EECS Department, University of California, Berkeley, Tech. Rep., 2008.
- [67] A. Burbano, S. Bouaziz, and M. Vasiliu, "3D-sensing Distributed Embedded System for People Tracking and Counting," in *International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec 2015, pp. 470–475.
- [68] C. C. T. Mendes and D. F. Wolf, "Real Time Autonomous Navigation and Obstacle Avoidance Using a Semi-global Stereo Method," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: ACM, 2013, pp. 235–236.
- [69] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive Avoidance Using Embedded Stereo Vision for MAV Flight," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 50–56.
- [70] S. Karakaya, G. Küçükyildiz, C. Toprak, and H. Ocak, "Development of a Human Tracking Indoor Mobile Robot Platform," in *Proceedings of the 16th International Conference on Mechatronics - Mechatronika (ME)*, Dec 2014, pp. 683–687.
- [71] S. Solak and E. D. Bolat, "Distance Estimation using Stereo Vision for Indoor Mobile Robot Applications," in *9th International Conference on Electrical and Electronics Engineering (ELECO)*, Nov 2015, pp. 685–688.
- [72] S. K. Gehrig, F. Eberli, and T. Meyer, *A Real-Time Low-Power Stereo Vision Engine Using Semi-Global Matching*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 134–143. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04667-4_14
- [73] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, "Real-Time Stereo Vision System using Semi-Global Matching Disparity Estimation: Architecture and FPGA-Implementation," in *International Conference on Embedded Computer Systems (SAMOS)*, July 2010, pp. 93–101.
- [74] J. Ding, J. Liu, W. Zhou, H. Yu, Y. Wang, and X. Gong, "Real-time stereo vision system using adaptive weight cost aggregation approach," *EURASIP Journal on Image and Video Processing*, vol. 2011, no. 1, pp. 1–19, 2011.
- [75] S. Perri, P. Corsonello, and G. Cocorullo, "Adaptive Census Transform: A Novel Hardware-oriented Stereovision Algorithm," *Computer Vision and Image Understanding*, vol. 117, no. 1, pp. 29–41, Jan. 2013.

- [76] C. Ttofis, C. Kyrkou, and T. Theocharides, "A Low-Cost Real-Time Embedded Stereo Vision System for Accurate Disparity Estimation Based on Guided Image Filtering," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2678–2693, Sept 2016.
- [77] G. C. S. L. Nalpantidis and A. Gasteratos, "Review of Stereo Vision Algorithms: From Software to Hardware," *International Journal of Optomechatronics*, vol. 2, no. 4, pp. 435–462, Jan 2008.
- [78] K.-J. Yoon and I. S. Kweon, "Adaptive Support-Weight Approach for Correspondence Search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 650–656, April 2006.
- [79] K. Ambrosch and W. Kubinger, "Accurate Hardware-based Stereo Vision," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1303–1316, Nov 2010. [Online]. Available: <http://dx.doi.org/10.1016/j.cviu.2010.07.008>
- [80] N. Baha and S. Larabi, "Accurate Real-time Neural Disparity MAP Estimation with FPGA," *Pattern Recognition*, vol. 45, no. 3, pp. 1195–1204, Mar. 2012. [Online]. Available: <http://dx.doi.org/10.1016/j.patcog.2011.08.005>
- [81] K. Zhang, J. Lu, and G. Lafruit, "Cross-Based Local Stereo Matching Using Orthogonal Integral Images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 7, pp. 1073–1079, July 2009.
- [82] K. He, J. Sun, and X. Tang, "Guided Image Filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 6, pp. 1397–1409, June 2013.
- [83] A. Hosni, M. Bleyer, C. Rhemann, M. Gelautz, and C. Rother, "Real-time Local Stereo Matching using Guided Image Filtering," in *IEEE International Conference on Multimedia and Expo*, July 2011, pp. 1–6.
- [84] P. Usachokcharoen, Y. Washizawa, and K. Pasupa, "Sign Language Recognition with Microsoft Kinect's Depth and Colour Sensors," in *IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, 2015, pp. 186–190.
- [85] J. Stowers, M. Hayes, and A. Bainbridge-Smith, "Altitude Control of a Quadrotor Helicopter Using Depth Map from Microsoft Kinect Sensor," in *IEEE International Conference on Mechatronics*, 2011, pp. 358–362.
- [86] M. G. Park, J. Park, Y. Shin, E. G. Lim, and K. J. Yoon, "Stereo vision with image-guided structured-light pattern matching," *Electronics Letters*, vol. 51, no. 3, pp. 238–239, 2015.
- [87] T. Tahara, R. Kawahara, S. Nobuhara, and T. Matsuyama, "Interference-Free Epipole-Centered Structured Light Pattern for Mirror-Based Multi-view Active Stereo," in *International Conference on 3D Vision*, 2015, pp. 153–161.
- [88] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *Proceedings of the 51st Annual Design Automation Conference*, 2014.

- [89] A. Das, B. M. Al-Hashimi, and G. V. Merrett, "Adaptive and Hierarchical Runtime Manager for Energy-aware Thermal Management of Embedded Systems," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 2, 2016.
- [90] S. Wildermann, T. Ziermann, and J. Teich, "Game-theoretic Analysis of Decentralized Core Allocation Schemes on Many-core Systems," in *Design, Automation Test in Europe*, 2013.
- [91] K. Bhatti, C. Belleudy, and M. Auguin, "Power Management in Real Time Embedded Systems through Online and Adaptive Interplay of DPM and DVFS Policies," in *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Dec 2010, pp. 184–191.
- [92] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz, "Prediction Models for Multi-dimensional Power-performance Optimization on Many Cores," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 250–259.
- [93] S. Tzilis, I. Sourdis, V. Vasilikos, D. Rodopoulos, and D. Soudris, "Runtime Management of Adaptive MPSoCs for Graceful Degradation," in *International Conference on Compilers, Architectures, and Synthesis of Embedded Systems (CASES)*, 2016, pp. 1–10.
- [94] Z. Wang, Z. Tian, J. Xu, R. K. V. Maeda, H. Li, P. Yang, Z. Wang, L. H. K. Duong, Z. Wang, and X. Chen, "Modular Reinforcement Learning for Self-Adaptive Energy Efficiency Optimization in Multicore System," in *22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 684–689.
- [95] P. Lama, Y. Guo, C. Jiang, and X. Zhou, "Autonomic Performance and Power Control for Co-Located Web Applications in Virtualized Datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1289–1302, May 2016.
- [96] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, "Learning Transfer-based Adaptive Energy Minimization in Embedded Systems," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 35, no. 6, 2016.
- [97] Y. Wang and M. Pedram, "Model-Free Reinforcement Learning and Bayesian Classification in System-Level Power Management," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3713–3726, Dec 2016.
- [98] S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi, "Adaptive energy minimization of embedded heterogeneous systems using regression-based learning," in *25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sept 2015, pp. 103–110.
- [99] J. C. Salinas-Hilburg, M. Zapater, J. L. Risco-Martin, J. M. Moya, and J. L. Ayala, "Unsupervised Power Modeling of Co-Allocated Workloads for Energy Efficiency in Data Centers," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2016, pp. 1345–1350.
- [100] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE)," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 2012, pp. 213–224.

- [101] H. Shen, J. Lu, and Q. Qiu, "Learning based DVFS for Simultaneous Temperature, Performance and Energy Management," in *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, March 2012, pp. 747–754.
- [102] D.-C. Juan and D. Marculescu, "Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2012, pp. 97–102.
- [103] H. Shen and Q. Qiu, "Contention Aware Frequency Scaling on CMPs with Guaranteed Quality of Service," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–6.
- [104] G.-Y. Pan, J.-Y. Jou, and B.-C. Lai, "Scalable Power Management Using Multilevel Reinforcement Learning for Multiprocessors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 4, pp. 33:1–33:23, Aug. 2014. [Online]. Available: <http://doi.acm.org/10.1145/2629486>
- [105] H. Sasaki, S. Imamura, and K. Inoue, "Coordinated Power-Performance Optimization in Manycores," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, Sept 2013, pp. 51–61.
- [106] S. Sridharan, G. Gupta, and G. S. Sohi, "Adaptive, Efficient, Parallel Execution of Parallel Programs," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014, pp. 169–180.
- [107] R. Ye and Q. Xu, "Learning-based Power Management for Multicore Processors via Idle Period Manipulation," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 33, no. 7, 2014.
- [108] D. Gadioli, S. Libutti, G. Massari, E. Paone, M. Scandale, P. Bellasi, G. Palermo, V. Zaccaria, G. Agosta, W. Fornaciari, and C. Silvano, "OpenCL Application Auto-tuning and Run-Time Resource Management for Multi-core Platforms," in *IEEE International Symposium on Parallel and Distributed Processing with Applications*, Aug. 2014, pp. 127–133.
- [109] A. J. Smola and B. Schölkopf, "A Tutorial on Support Vector Regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, Aug. 2004.
- [110] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated Management of Multiple Interacting Resources in Chip Multiprocessors: A Machine Learning Approach," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 318–329.
- [111] D.-C. Juan, S. Garg, J. Park, and D. Marculescu, "Learning the Optimal Operating Point for Many-core Systems with Extended Range Voltage/Frequency Scaling," in *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, 2013.
- [112] Y. Wen, Z. Wang, and M. F. P. O'Boyle, "Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms," in *21st International Conference on High Performance Computing (HiPC)*, Dec 2014, pp. 1–10.

- [113] H. Jung and M. Pedram, "Improving the Efficiency of Power Management Techniques by Using Bayesian Classification," in *9th International Symposium on Quality Electronic Design (isqed 2008)*, 2008, pp. 178–183.
- [114] —, "Supervised Learning Based Power Management for Multicore Processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 9, pp. 1395–1408, 2010.
- [115] J. Ma, G. Yan, Y. Han, and X. Li, "An Analytical Framework for Estimating Scale-Out and Scale-Up Power Efficiency of Heterogeneous Manycores," *IEEE Transactions on Computers*, vol. 65, no. 2, pp. 367–381, Feb 2016.
- [116] Y. Wu, D. S. Nikolopoulos, and R. Woods, "Runtime Support for Adaptive Power Capping on Heterogeneous SoCs," in *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, July 2016, pp. 71–78.
- [117] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIII. New York, NY, USA: ACM, 2008, pp. 48–59. [Online]. Available: <http://doi.acm.org/10.1145/1346281.1346289>
- [118] M. A. H. Monil, R. Qasim, and R. M. Rahman, "Energy-aware VM Consolidation Approach Using Combination of Heuristics and Migration Control," in *Ninth International Conference on Digital Information Management (ICDIM 2014)*, Sept 2014, pp. 74–79.
- [119] M. A. Suleman, M. K. Qureshi, and Y. N. Patt, "Feedback-driven Threading: Power-efficient and High-performance Execution of Multi-threaded Workloads on CMPs," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIII. New York, NY, USA: ACM, 2008, pp. 277–286. [Online]. Available: <http://doi.acm.org/10.1145/1346281.1346317>
- [120] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-out processors," in *39th Annual International Symposium on Computer Architecture (ISCA)*, June 2012, pp. 500–511.
- [121] Y. G. Kim, M. Kim, and S. W. Chung, "Enhancing Energy Efficiency of Multimedia Applications in Heterogeneous Mobile Multi-core Processors," *IEEE Transactions on Computers*, vol. 66, no. 11, 2017.
- [122] F. Gong, L. Ju, D. Zhang, M. Zhao, and Z. Jia, "Cooperative DVFS for Energy-efficient HEVC Decoding on Embedded CPU-GPU Architecture," in *ACM/EDAC/IEEE Design Automation Conference*, 2017.
- [123] M. Otoom, P. Trancoso, H. Almasaeid, and M. Alzubaidi, "Scalable and Dynamic Global Power Management for Multicore Chips," in *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures*, ser. PARMA-DITAM '15. New York, NY, USA: ACM, 2015, pp. 25–30. [Online]. Available: <http://doi.acm.org/10.1145/2701310.2701312>

- [124] X. Lin, Y. Wang, and M. Pedram, “A Reinforcement Learning-Based Power Management Framework for Green Computing Data Centers,” in *IEEE International Conference on Cloud Engineering (IC2E)*, April 2016, pp. 135–138.
- [125] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal, “A Generalized Software Framework for Accurate and Efficient Management of Performance Goals,” in *Proceedings of the 11th ACM International Conference on Embedded Software*, 2013.
- [126] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, “Proactive Control of Approximate Programs,” *ACM SIGOPS Operating Systems Review*, 2016.
- [127] V. Vassiliadis, C. Chaliros, K. Parasyris, C. D. Antonopoulos, S. Lalis, N. Bellas, H. Vandierendonck, and D. S. Nikolopoulos, “Exploiting Significance of Computations for Energy-constrained Approximate Computing,” *International Journal of Parallel Programming*, 2016.
- [128] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, “Application Heartbeats: A Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments,” in *Proceedings of 7th International Conference on Autonomic Computing*, 2010.
- [129] A. Baldassari, C. Bolchini, and A. Miele, “A Dynamic Reliability Management Framework for Heterogeneous Multicore Systems,” in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2017.
- [130] F. Gaspar, L. Taniça, P. Tomás, A. Ilic, and L. Sousa, “A Framework for Application-guided Task Management on Heterogeneous Embedded Systems,” *ACM Transactions on Architecture and Code Optimization*, vol. 12, no. 4, 2015.
- [131] V. M. Weaver, D. Terpstra, H. McCraw, M. Johnson, K. Kasichayanula, J. Ralph, J. Nelson, P. Mucci, T. Mohan, and S. Moore, “Papi 5: Measuring Power, Energy, and the Cloud,” in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2013.
- [132] B. G. Pinto, L. M. de Souza Xavier, R. M. Favaretto, and G. G. H. Cavaleiro, “Um Framework para Monitoração do Consumo Energético em Arquiteturas Multicore,” *Revista Brasileira de Computação Aplicada*, vol. 7, no. 3, 2015.
- [133] T. Kidd, *Measuring Power on Intel Xeon Phi™ Product Family Devices*, Intel, 2015. [Online]. Available: <https://software.intel.com/en-us/articles/measuring-power-on-intel-xeon-phi-product-family-devices>
- [134] J. Cohen, P. Cohen, S. West, and L. Aiken, *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. Taylor & Francis, 2013.
- [135] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. [Online]. Available: <http://www.R-project.org/>
- [136] N. R. Draper and H. Smith, *Applied Regression Analysis*, 3rd ed. Wiley-Blackwell, 1998.

- [137] F. Rego, *Quick Guide: Interpreting Simple Linear Model Output in R*, Oct. 2015. [Online]. Available: <https://feliperego.github.io/blog/2015/10/23/Interpreting-Model-Output-In-R>
- [138] B. Kim. (2015, Sep.) Understanding diagnostic plots for linear regression analysis. Online. University of Virginia Library. [Online]. Available: <http://data.library.virginia.edu/diagnostic-plots/>
- [139] MathWorks. Robust Regression - Reduce Outlier Effects. MathWorks. [Online]. Available: <https://uk.mathworks.com/help/stats/robust-regression-reduce-outlier-effects.html>
- [140] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.
- [141] Y. Morvan, "Acquisition, Compression and Rendering of Depth and Texture for Multi-View Video," Ph.D. dissertation, Eindhoven University of Technology, The Netherlands, 2009.
- [142] A. Bhatti, *Advances in Theory and Applications of Stereo Vision*. InTech, Jan. 2011.
- [143] H.-S. Son, K.-r. Bae, S.-H. Ok, Y.-H. Lee, and B. Moon, *A Rectification Hardware Architecture for an Adaptive Multiple-Baseline Stereo Vision System*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 147–155. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-27192-2_19
- [144] Q. Zhang, L. Xu, and J. Jia, "100+ Times Faster Weighted Median Filter (WMF)," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 2830–2837.
- [145] D. Scharstein and R. Szeliski, "High-Accuracy Stereo Depth Maps Using Structured Light," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 1, June 2003, pp. I–195–I–202 vol.1.
- [146] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On Building an Accurate Stereo Matching System on Graphics Hardware," in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, Nov 2011, pp. 467–474.
- [147] M. Bleyer, C. Rhemann, and C. Rother, "PatchMatch Stereo - Stereo Matching with Slanted Support Windows," in *British Machine Vision Conference, 2011*, pp. 1–11, vortrag: British Machine Vision Conference BMVC 2011, Dundee; 2011-08-29 – 2011-09-02.
- [148] W. Wang, J. Yan, N. Xu, Y. Wang, and F. H. Hsu, "Real-time High-quality Stereo Vision System in FPGA," in *Field-Programmable Technology (FPT), 2013 International Conference on*, Dec 2013, pp. 358–361.
- [149] M. Jin and T. Maruyama, "Fast and Accurate Stereo Vision System on FPGA," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 1, 2014.
- [150] H. Hirschmuller, "Stereo Processing by Semiglobal Matching and Mutual Information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, Feb 2008.

- [151] Y. Shan, Y. Hao, W. Wang, Y. Wang, X. Chen, H. Yang, and W. Luk, "Hardware Acceleration for an Accurate Stereo Vision System Using Mini-Census Adaptive Support Region," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4s, pp. 132:1–132:24, Apr. 2014.
- [152] L. Zhang, K. Zhang, T. S. Chang, G. Lafruit, G. K. Kuzmanov, and D. Verkest, "Real-Time High-Definition Stereo Matching on FPGA," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2011, pp. 55–64.
- [153] M. Jin and T. Maruyama, "A Real-time Stereo Vision System Using a Tree-structured Dynamic Programming on FPGA," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 21–24. [Online]. Available: <http://doi.acm.org/10.1145/2145694.2145698>
- [154] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S. K. Park, M. Kim, and J. W. Jeon, "FPGA Design and Implementation of a Real-Time Stereo Vision System," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15–26, 2010.
- [155] Y. Shan, Z. Wang, W. Wang, Y. Hao, Y. Wang, K. Tsoi, W. Luk, and H. Yang, "FPGA based Memory Efficient High Resolution Stereo Vision System for Video Tolling," in *Field-Programmable Technology (FPT), 2012 International Conference on*, Dec 2012, pp. 29–32.
- [156] N. Y. C. Chang, T. H. Tsai, B. H. Hsu, Y. C. Chen, and T. S. Chang, "Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 6, pp. 792–805, June 2010.
- [157] Itseez. (2015) Open Source Computer Vision Library. <https://github.com/itseez/opencv>.
- [158] C. Leech, G. Bragg, J. Davis, G. Constantinides, G. V. Merrett, and B. Al-Hashimi, "The PRiME Framework: Application- and Platform-agnostic Runtime Management," Under Review.
- [159] H. J. Curnow and B. A. Wichmann, "A Synthetic Benchmark," *The Computer Journal*, vol. 19, no. 1, 1976.
- [160] L. A. Maeda-Nunez, A. K. Das, R. A. Shafik, G. V. Merrett, and B. Al-Hashimi, "PoGo: An Application-specific Adaptive Energy Minimisation Approach for Embedded Systems," in *HiPEAC Workshop on Energy Efficiency with Heterogenous Computing*, 2015.
- [161] B. K. Reddy, A. K. Singh, G. V. Merrett, and B. M. Al-Hashimi, "ITMD: Run-time Management of Concurrent Multi-threaded Applications on Heterogeneous Multi-cores," in *Design, Automation and Test in Europe*, 2017.