Reliable Mapping and Partitioning of Performance-constrained OpenCL Applications on CPU-GPU MPSoCs

Eduardo Weber Wachter, Geoff V. Merrett and Bashir M. Al-Hashimi
University of Southampton
Southampton, United Kingdom
eww1n17@soton.ac.uk,{gvm,bmah}@ecs.soton.ac.uk

Amit Kumar Singh
University of Essex
Colchester, United Kingdom
a.k.singh@essex.ac.uk

ABSTRACT

Heterogeneous Multi-Processor Systems-on-Chips (MPSoCs) containing CPU and GPU cores are typically required to execute applications concurrently. Existing approaches exploit applications executing in CPU and GPU cores at the same time taking into account performance and energy consumption for mapping and partitioning. This paper presents a proposal for mapping and partitioning of applications in CPU-GPU MPSoCs taking into account the temperature behavior of the system. We evaluate the temperature profiling to partition the applications between CPU and GPU. The profiling is done by measuring the temperature of the CPU and GPU cores while executing different applications at different partitions. Results shown up to 13% savings of average temperature of the chip while maintaining performance requirements. A lower thermal behavior represents a better long-term reliability (lifetime) of the SoC.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**;

KEYWORDS

Heterogeneous MPSoC, OpenCL applications, Mapping, Partitioning, Performance, Energy consumption, Temperature-aware

ACM Reference format:
Eduardo Weber Wachter, Geoff V. Merrett and Bashir M. Al-Hashimi and Amit Kumar Singh. 2017. Reliable Mapping and Partitioning of Performance-constrained OpenCL Applications on CPU-GPU MPSoCs. In *Proceedings of 15th IEEE/ACM Symposium on Embedded Systems for Real-Time Multimedia, , 2017 (ESTIMEDIA),* 6 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Modern embedded systems, e.g. mobile phones, rely on heterogeneous Multi-Processor Systems-on-Chips (MPSoCs) containing different types of cores. An example of a commercial heterogeneous MPSoC is the Samsung Exynos 5422 SoC [1]. This SoC contains 4 ARM Cortex-A15 (big) CPU, 4 ARM Cortex-A7 (LITTLE) CPU and 6 ARM Mali-T628 GPU cores. Such an architecture provides opportunities to exploit distinct features of different types of cores in order to meet end-user demands in terms of performance, energy consumption and thermal profile [22] [23] [21]. This platform also has five temperature monitors enabling to take decisions based on the current state of the chip. The GPU and each one of the four A15 cores have its own temperature monitor.

Platforms as this SoC became possible due to the technology node size reduction. But these integrated circuits become also more prone to aging phenomena jeopardizing their reliability. Scaling to new technology nodes leads to progressive degradation of the performance characteristics of devices and system components [7] induced by aging phenomena. Some works show that the impact of temperature-induced variability on circuit lifetime can be higher than that due to stress and exceed 50% over the value estimated considering the circuit average temperature [6].

For a given application, simultaneous exploitation of heterogeneous cores having different instruction set architectures (ISAs) such as CPU and GPU is challenging, as they handle instructions in different ways. Additionally, CPU cores typically handle task and thread level parallelisms, whereas GPU cores handle data level parallelism.

OpenCL [2] provides an opportunity to write programs that can execute across heterogeneous cores including CPUs and GPUs [13, 15, 17, 20]. However, depending upon the kind of parallelism dominant in the application, the performance, energy consumption and temperature will vary when it is allocated onto only CPU, only GPU, or both CPU and GPU cores.

This paper presents a proposal for mapping and partitioning of applications in CPU-GPU MPSoCs taking into account the temperature behavior of the system. We evaluate the temperature profiling to partition the applications between CPU and GPU. The profiling is done by measuring the temperature of the CPU and GPU cores while executing different applications at different partitions. With a temperature profiling, we can choose a better thermal behavior and consequently present a lower impact on long-term reliability (lifetime) of the SoC.

The data-parallel applications are potential candidates to concurrently exploit cores of a MPSoC as data can be processed in parallel on the cores. However, each application should be written in OpenCL to exploit cores of two different ISAs such as CPU and GPU. The GPU version of the popular Polybench benchmark suite [12] contains such data-parallel applications written in OpenCL and we use them. The application codes are slightly modified to launch them only on CPU cores, only on GPU cores, or on both

ESTIMEDIA, 2017,
© 2017 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CPU and GPU cores. Additionally, an appropriate work-group size for each application is selected as in [20].

2 MOTIVATIONAL EXAMPLE

Figures 1 presents four OpenCL applications (SYR2K, SYRK, CORRELATION, and COVARIANCE) from the Polybench benchmark [12] executing on the Exynos 5422 heterogeneous MPSoC while varying the fraction of application workload (threads) executed on CPU cores and remaining threads on GPU cores. Both CPU and GPU are set with the maximum possible voltage-frequency. A fraction of zero indicates that no threads are executed on CPU cores, i.e. all of them are executed only on the GPU cores. Similarly, when this value is 1, all the threads are executed only on CPU cores and none on GPU cores.

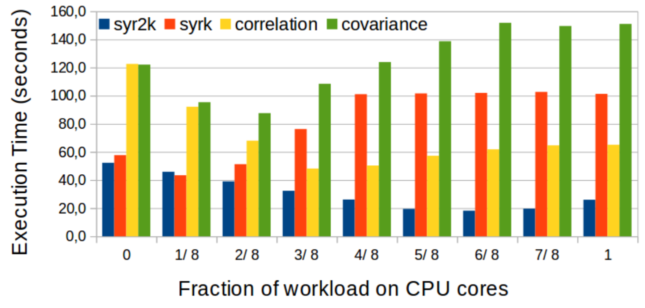

Figure 1: Execution time (ET) at varying fraction of application workload (threads) to be executed on CPU cores.

Figure 1 [24] shows that different applications have different behaviors in different cores. Some execute faster on CPUs (e.g., CORRELATION) and some on GPU (e.g., COVARIANCE). Further, all applications show a significant reduction in execution time when run on both the CPU and GPU cores, with the best partitioning of threads. These observations indicate the advantages of simultaneously exploiting both CPU and GPU cores for each application.

This work [24] evaluates performance and energy consumption, but does not take into account the temperature variations due to the partition between CPU and GPU. Figure 2 shows the same testcase from Figure 1 but evaluates the average temperature of the CPU and GPU cores. It also shows the average temperature has a different behavior for each application. Some applications shows a lower temperature with all workloads executing on GPU (e.g. SYRK) and some on CPUs (e.g. CORR).

Figure 3 illustrates the temperature behavior of one A15 core (the core 7 is used in these four testcases) temperature for the SYRK application when the big and LITTLE CPU cores are used in four different mapping combinations: one LITTLE and four big (1L+4B), two LITTLE and four big (2L+4B), three LITTLE and three big (3L+3B) and four LITTLE and two big (4L+2B). In this testcase we are only evaluating the mapping between different CPU cores, therefore the GPU is not used. The best temperature profile is achieved when using the mapping 4L+2B, which has an average of 66.2 °C, but it shows the longest execution time. Using three cores of each (3B+3L) leads to a higher temperature profile, but with a lower execution time. The best execution time is achieved with two

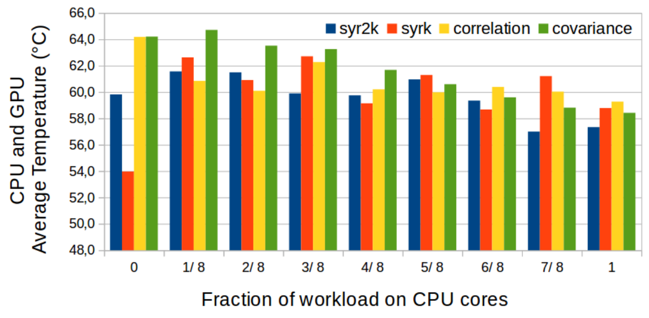

Figure 2: Average Temperature on CPU and GPU at varying fraction of application workload (threads) to be executed on CPU cores.

mappings (1L+4B and 2L+4B), with similar temperature profiles, which shows that there will be no gain executing the application with more than 1 LITTLE and four big cores.

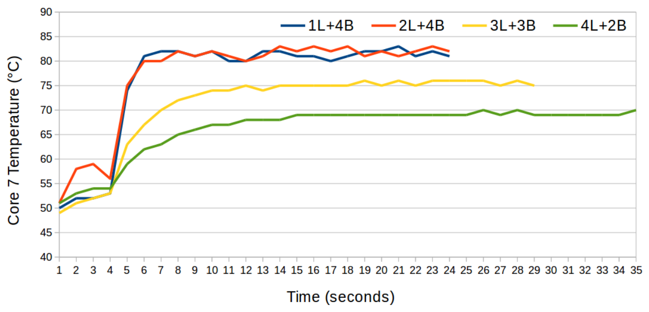

Figure 3: Temperature behavior in one A15 CPU core over time for four mappings of application SYRK.

Figure 4 shows the temperature behavior over time for three of the partitions in application SYRK. The temperature is measured each second in this scenario. In this scenario, a partition of 0 (all workload executing on the GPU) shows a better profile (lower temperature), but it shows a higher execution time (32 seconds). The partition 1/8 shows a lower execution time (24 seconds), but a higher temperature.

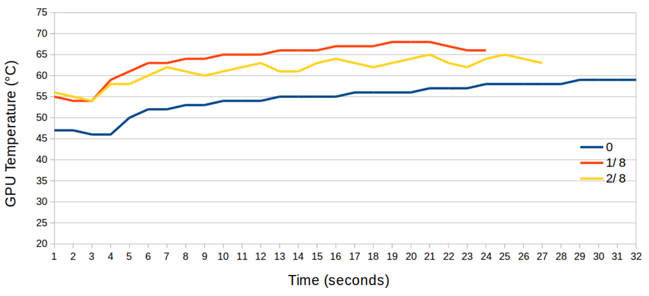

Figure 4: GPU Temperature behavior over time for three partitions of application SYRK. CPU executes 0, 1/8 and 2/8 of the applications workload.

The same behavior is observed in one of the A15 cores shown in Figure 5. It shows that the partition of 0 has a lower temperature with higher execution time, while the partition 1/8 has a lower execution time but a higher temperature. These observations indicate that there is design exploration space for choosing the best trade off between temperature profile and performance (execution time).

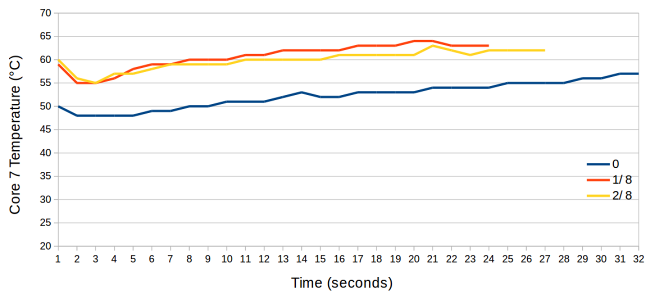

Figure 5: Temperature behavior in one A15 CPU core over time for three partitions of application SYRK. CPU executes 0, 1/8 and 2/8 of the applications workload.

3 STATE-OF-THE-ART

Mapping of multi-threaded applications on single-ISA heterogeneous MPSoCs has been a hot topic [4, 5, 8, 10, 18, 25, 26]. Most of these approaches consider Samsung Exynos 5422 SoC and utilize 4 big and/or 4 LITTLE cores that have the same ISA [4, 8, 25]. Further, for a given application, most of these approaches do not concurrently exploit more than one types of cores [10, 18, 25, 26]. Although there has been some effort to concurrently exploit both big and LITTLE cores [4], it cannot be applied to exploit cores having different ISAs such as CPU and GPU because they handle instructions in different ways.

There has been efforts to simultaneously exploit CPU and GPU cores in desktop platforms, but CPU and GPU cores are not situated within a single chip [14, 17]. In these works, CPU cores are used for general purpose tasks and GPU cores to accelerate data-parallel tasks. Such allocation of tasks to cores leads to improved throughput and energy efficiency. Further, since CPU and GPU cores are situated in different chips, these approaches cannot be efficiently applied to MPSoC due to different communication infrastructure.

For desktop platforms, there has also been efforts to exploit CPU and GPU cores present within a single chip [19, 27, 28]. In these platforms, coordination of CPU and GPU cores needs more consideration. In [27], an algorithm is proposed to partition the workload and power budget between CPU and GPU cores of an AMD Trinity single chip heterogeneous platform to improve throughput. In [28], similar AMD platform is used to perform coordinated CPU-GPU executions, but memory contention occurs due to access of the same bank in different patterns by the CPU and GPU. In [19], the problem of shared resources in AMD platforms is addressed. However, these efforts do not consider limited power budget that is available for embedded systems operating from batteries.

For mobile platforms used in embedded systems and containing CPU and GPU cores within a single chip, there has been some works to partition the application threads between CPU and GPU cores. In [11], HPC workloads are executed on Mali GPU to achieve energy efficiency, but the possible collaboration with CPU is not considered. In [9], the threads are partitioned by considering shared resources and synchronization. However, these works do not use GPU for OpenCL kernel execution. OpenCL framework for ARM processors was introduced in [16]. In [20], a similar open source framework, FreeOCL [3] is used for the ARM CPU that acts as both the host processor and an OpenCL device. This enables concurrent use of CPU and GPU to execute an application threads, but in [20], a static partitioning is performed by using all the CPU and GPU cores.

The Authors in [6] show that the impact of temperature-induced variability on circuit lifetime can be higher due to stress and exceed over the value estimated considering the circuit average temperature. They propose a simulation framework for the BTI degradation analysis of DVFS designs that considers thermal profiles under the influence of a Dynamic Thermal Management (DTM) system. Using the proposed framework the work explores the expected lifetime and performance circuits from a 32nm CMOS technology library, for various thermal management constraints. Results shown that the proposed framework can tradeoff long-term reliability (lifetime) and performance with higher accuracy when taking into account the temperature of the chip.

A close observation of approaches to map and partition application threads between CPU and GPU cores of a mobile MPSoC indicates that they cannot be efficiently applied. Further, while doing such partitioning, existing approaches do not consider the temperature behavior of the applications. In contrast, our proposed approach performs energy-efficient and temperature-aware mapping and partitioning of applications' threads of each application.

4 PROPOSED MAPPING AND PARTITIONING

An overview of the proposed thread mapping and partitioning approach is illustrated in Figure 6. Similar approach is followed for all the applications, one after another. Our approach extends the work in [24] taking into account the temperature behavior of the applications to execute the mapping and partitioning. The main steps of the approach are as follows (See Figure 6): (1) Generating mappings using CPU/GPU cores. (2) Evaluation of each mapping for execution time, energy consumption and temperature profile at various partitions. (3) Finding performance and energy optimized set of design points meeting performance requirements. (4) Selection of the point having the best temperature and energy profile.

The main aspects of the mapping and partitioning approach are as follows.

- Consideration of CPU and GPU cores processing capability to identify the partitioning of work-groups.
- Consideration of CPU and GPU cores temperature behavior.

Algorithm 1 provides more details of the proposed methodology to perform energy and thermal aware partitioning. The details of Algorithm 1 are as follows.

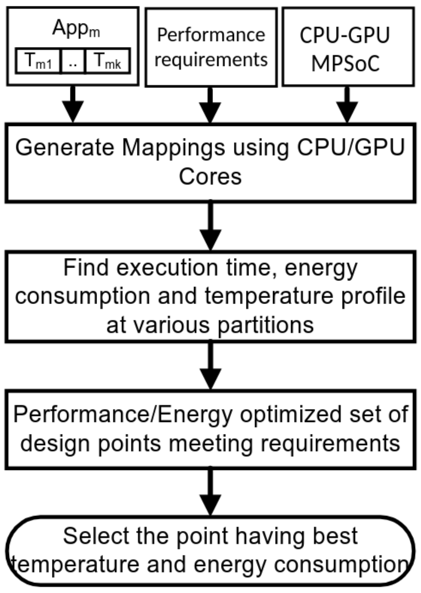
App$_m$
T$_{m1}$ .. T$_{mk}$
Performance requirements
CPU-GPU MPSoC
Generate Mappings using CPU/GPU Cores
Find execution time, energy consumption and temperature profile at various partitions
Performance/Energy optimized set of design points meeting requirements
Select the point having best temperature and energy consumption

Figure 6: Methodology

4.1 Generating mappings using CPU/GPU cores

For each available application (App_1 to App_m), The total number of mappings (design points) considering both the CPU and GPU cores of the considered MPSoC are:

$$NumMappings = \{N_b + N_L + (N_b \times N_L)\} + 1 \tag{1}$$

Where, N_b and N_L are the number of big and LITTLE cores, respectively. For GPU cores, since all the cores are used by the application, there is only one mapping. For the considered MPSoC, there are 4 big and 4 LITTLE CPU cores.

4.2 Evaluation of mappings at various partitions

For each mapping (line 1), the fraction of work-items to be executed on CPU and GPU cores are varied and execution time (*ET*), energy consumption (*EC*) and temperature profile (*TEMP*) is captured. The *ET* is determined by the device taking maximum time, CPU or GPU. *EC* is computed by using the available on-board power sensors by getting power samples at every 100ms. The *TEMP* is captured by using the available on-board temperature sensors. Although the *TEMP* is captured over time, we take average over all the temperature samples as it affects reliability the most. Evaluation at each partition represents a design point. For facilitating application execution as OpenCL kernels, the workload on both CPU and GPU should be multiples of work-group size. Therefore, the partitioning point has been calculated as the number of work-groups that is nearest to the desired fraction of the CPU workload. Similar steps are followed for each mapping.

Algorithm 1 Thread-to-core mapping and repartitioning of threads

```
1: for each mapping do
2:     for each partition do
3:         Find execution time $ET$;
4:         Find energy consumption $EC$;
5:         Find temperature profile $TEMP$;
6:     end for
7: end for
8: Create a set ($minEC$) of design points having low energy consumption and satisfying $Apps_{Prfr}$;
9: Find the $minTEMP\_EC$ design point, return partition and number of used cores, their types and frequencies;
```

4.3 Finding performance and energy optimized set

For each mapping, among all the evaluated design points in the previous step, the ones satisfying the performance requirements and having low energy consumption are selected as set *minEC*. Towards this, we select a limited number of design points at each mapping option such that their number is in control. This whole process gives energy optimized design options at various combinations of used CPU and GPU cores and designer can choose one of them depending upon the resource availability.

4.4 Energy and temperature optimized design point

Among the performance and energy optimized set of design points, one appropriate partitioning of work-items between CPU and GPU leading to optimized temperature profile is possible, which will give energy and temperature optimized design point. Towards this, we select the point having minimum energy_consumption × average_temperature. Such selection leads the a design point having optimized energy consumption and temperature as we want to minimize both of these metrics. For this selected design point, Algorithm 1 returns number of used cores, their types and frequencies as the mapping and fraction of work-items as the partition.

5 RESULTS

This section evaluates the mapping and partitioning of threads for all applications of the Polybench benchmark. The evaluation is executed on an Odroid-XU3 platform that runs a modified Ubuntu Linux Kernel 3.10.96. The experiments were validated on the board with the heat sink and fan. For this evaluation we take into account each application executing individually at the platform with all processing cores set to the maximum frequency: 2 GHz for A15, 1.4 GHz for A7 and 600 MHz for GPU. The first column of Table 1 shows considered applications with its abbreviations and work groups. Some of these applications, e.g., 2 dimensional convolution (2DCONV) and 2 dimensional matrix multiplication (2MM) are representative set of kernels used in multimedia processing.

Each one of these applications is executed with different partitions and mappings to profile energy consumption, temperature profile and performance (application execution time). These experiments are repeated to ensure correctness and we report the average

Table 1: Selected applications from Polybench [12], their number of work-groups

App Name	Abbreviation	# work-groups
CORRELATION	CR	2048
SYR2K	S2	512
SYRK	SR	512
COVARIANCE	CV	2048
2MM	2M	128
2DCONV	2D	2048
GEMM	GE	512
MVT	MV	4096

of these runs. Then, the proposed method is applied to perform a trade-off between temperature and energy while maintaining the performance requirements. To put the results in perspective, we compare our approach with the work in [24] which does not take into account the temperature profiling.

Figures 7 and 8 show some design exploration points for application S2 evaluating the temperature on GPU and CPU. Figure 7 shows the temperature behavior for mapping on two LITTLE and four big cores while Figure 8 on four LITTLE and two big cores. Both Figures shows the exploration of partitions for fractions 6/8, 7/8 and 1 threads executing on CPU. Both Figures show the different behavior on temperature for each mapping and partition. On Figure 8 the behavior of the A15 core 4 shows a bigger variation according to the desired fraction, while in Figure 7 there is less variation.

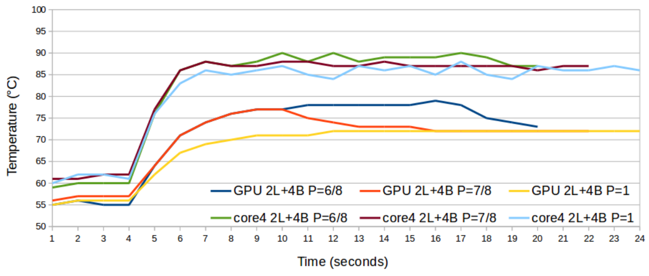

Figure 7: Temperature behavior exploration of fractions 6/8, 7/8 and 1 for application S2 mapped on two LITTLE and four big cores.

Basically, the difference between these two approaches is that we evaluate design points at various partitions with temperature profiles, whereas in [24] it is only performance-energy optimized.

Table 2 shows the comparison for some applications from Polybench benchmark with the best mapping/partition chosen by each one of the proposals. The column *Best Partition* shows an approximate value of the workload to be executed on the CPU for the work in [24] (*Perf-energy* column) compared to this work (*Temp.* column). A value of zero represents that all workload is executed on the GPU, while a value in a value of one all tasks are executed on the GPU.

The column *Average Temperature* shows the differences comparing our approach with [24]. Three overheads are shown, Performance (Application execution time), the average temperature

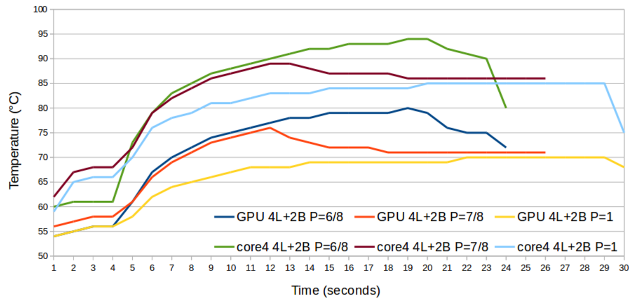

Figure 8: Temperature behavior exploration of fractions 6/8, 7/8 and 1 for application S2 mapped on four LITTLE and two big cores.

Table 2: Performance and Temperature differences for applications from Polybench. Comparison between our proposal and [24]

APP	Best Partition		Perf.	Average Temperature		
	[24] Perf-energy	Temp.		A15+ GPU	A15	GPU
2D	1/8	0	8.88%	-10.73%	-7.97%	-13.27%
2M	3/8	4/8	38.20%	-1.97%	-0.81%	-3.22%
CR	3/8	4/8	4.44%	-3.32%	-2.01%	-4.71%
CV	2/8	3/8	23.80%	-0.40%	0.19%	-0.99%
GE	1/8	0	13.17%	-11.55%	-13.44%	-9.54%
MV	2/8	2/8	0.00%	0.00%	0.00%	0.00%
S2	6/8	7/8	8.03%	-3.96%	-2.69%	-5.37%
SR	1/8	0	33.05%	-13.83%	-13.14%	-14.49%

on the GPU, on the A15 core and the average between this two values (*A15+GPU* column). For the application 2D, for example, our partition adds an overhead of 8.88% in performance, but it saves 10.73% of temperature. The savings in temperature can go up to 13.83% in some cases. A saving of 13% represents a lower average temperature of 8 °C.

6 CONCLUSIONS

This paper presented a temperature-aware mapping and partitioning for CPU-GPU MPSoCs. Taking the temperature behavior of the applications from the Polybench benchmark allows to reduce the average temperature of the system by 13% while mantaining performance when comparing to other state of the art approaches. This reduction on average temperature represents a significant impact on long-term reliability (lifetime) of the SoC.

This mapping and partitioning can be extended to multiple applications executing at run-time. The method can also be extended to apply DVFS for each computing module separately (A15, A7 and the GPU cores).

ACKNOWLEDGMENTS

This work was supported in parts by the EPSRC Grant EP/L000563/1 and the Grant EP/K034448/1, PRiME Programme (www.prime-project.org). Experimental data used in this paper can be found at DOI: http://doi.org/10.5258/SOTON/D0258.

REFERENCES

[1] 2016. Exynos 5 Octa (5422). (2016). www.samsung.com/exynos/

[2] 2016. The open standard for parallel programming of heterogeneous systems. (2016). https://goo.gl/A9wXRJ

[3] 2017. FreeOCL: Multi-platform implementation of OpenCL 1.2 targeting CPUs. (2017). Retrieved 2017 from https://github.com/zuzuf/freeocl

[4] Ali Aalsaud, Rishad Shafik, Ashur Rafiev, Fie Xia, Sheng Yang, and Alex Yakovlev. 2016. Power–Aware Performance Adaptation of Concurrent Applications in Heterogeneous Many-Core Systems. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design (ISLPED '16)*. ACM, New York, NY, USA, 368–373. https://doi.org/10.1145/2934583.2934612

[5] Karunakar Reddy Basireddy, Amit Singh, Geoff V. Merrett, and Bashir M. Al-Hashimi. 2017. ITMD: run-time management of concurrent multi-threaded applications on heterogeneous multi-cores. In *Conference on Design, Automation and Test in Europe 2017 (DATE'17)*. https://eprints.soton.ac.uk/406291/

[6] H. Chahal, V. Tenentes, D. Rossi, and B. M. Al-Hashimi. 2016. BTI aware thermal management for reliable DVFS designs. In *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. 1–6. https://doi.org/10.1109/DFT.2016.7684059

[7] V. Chandra. 2014. Monitoring reliability in embedded processors - A multi-layer view. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2593069.2596682

[8] Kiran Chandramohan and Michael F.P. O'Boyle. 2014. Partitioning Data-parallel Programs for Heterogeneous MPSoCs: Time and Energy Design Space Exploration. In *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '14)*. ACM, New York, NY, USA, 73–82. https://doi.org/10.1145/2597809.2597822

[9] Kiran Chandramohan and Michael F.P. O'Boyle. 2014. Partitioning Data-parallel Programs for Heterogeneous MPSoCs: Time and Energy Design Space Exploration. In *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '14)*. ACM, New York, NY, USA, 73–82. https://doi.org/10.1145/2597809.2597822

[10] B. Donyanavard, T. MÃijck, S. Sarma, and N. Dutt. 2016. SPARTA: Runtime task allocation for energy efficient heterogeneous manycores. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 1–10.

[11] I. Grasso, P. Radojkovic, N. Rajovic, I. Gelado, and A. Ramirez. 2014. Energy Efficient HPC on Embedded SoCs: Optimization Techniques for Mali GPU. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 123–132. https://doi.org/10.1109/IPDPS.2014.24

[12] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos. 2012. Autotuning a high-level language targeted to GPU codes. In *2012 Innovative Parallel Computing (InPar)*. 1–10. https://doi.org/10.1109/InPar.2012.6339595

[13] Dominik Grewe and Michael F. P. O'Boyle. 2011. A Static Task Partitioning Approach for Heterogeneous Systems Using OpenCL. In *Proceedings of the 20th International Conference on Compiler Construction: Part of the Joint European Conferences on Theory and Practice of Software (CC'11/ETAPS'11)*. Springer-Verlag, Berlin, Heidelberg, 286–305. http://dl.acm.org/citation.cfm?id=1987237.1987259

[14] Dominik Grewe and Michael F. P. O'Boyle. 2011. A Static Task Partitioning Approach for Heterogeneous Systems Using OpenCL. In *Proceedings of the 20th International Conference on Compiler Construction: Part of the Joint European Conferences on Theory and Practice of Software (CC'11/ETAPS'11)*. Springer-Verlag, Berlin, Heidelberg, 286–305. http://dl.acm.org/citation.cfm?id=1987237.1987259

[15] Dominik Grewe, Zheng Wang, and Michael F. P. O'Boyle. 2014. *OpenCL Task Partitioning in the Presence of GPU Contention.* Springer International Publishing, Cham, 87–101. https://doi.org/10.1007/978-3-319-09967-5_5

[16] Gangwon Jo, Won Jong Jeon, Wookeun Jung, Gordon Taft, and Jaejin Lee. 2014. OpenCL Framework for ARM Processors with NEON Support. In *Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing (WPMVP '14)*. ACM, New York, NY, USA, 33–40. https://doi.org/10.1145/2568058.2568064

[17] C. K. Luk, S. Hong, and H. Kim. 2009. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 45–55.

[18] J. Ma, G. Yan, Y. Han, and X. Li. 2016. An Analytical Framework for Estimating Scale-Out and Scale-Up Power Efficiency of Heterogeneous Manycores. *IEEE Trans. Comput.* 65, 2 (Feb 2016), 367–381. https://doi.org/10.1109/TC.2015.2419655

[19] Indrani Paul, Vignesh Ravi, Srilatha Manne, Manish Arora, and Sudhakar Yalamanchili. 2013. Coordinated Energy Management in Heterogeneous Processors. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*. ACM, New York, NY, USA, Article 59, 12 pages. https://doi.org/10.1145/2503210.2503227

[20] A. Prakash, S. Wang, A. E. Irimiea, and T. Mitra. 2015. Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*. 208–215. https://doi.org/10.1109/ICCD.2015.7357105

[21] Karunakar Basireddy Reddy, Amit Kumar Singh, Dwaipayan Biswas, Geoff V. Merrett, and Bashir M. Al-Hashimi. 2017. Inter-cluster Thread-to-core Mapping and DVFS on Heterogeneous Multi-cores. *IEEE Transactions on Multi-Scale Computing Systems* (2017).

[22] Amit Kumar Singh, Piotr Dziurzanski, Hashan Roshantha Mendis, and Leandro Soares Indrusiak. 2017. A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems. *ACM Comput. Surv.* 50, 2, Article 24 (April 2017), 40 pages. https://doi.org/10.1145/3057267

[23] Amit Kumar Singh, Charles Leech, Karunakar Reddy Basireddy, Bashir M Al-Hashimi, and Geoff V Merrett. 2017. Learning-based Run-time Power and Energy Management of Multi/Many-core Systems: Current and Future Trends. In *Journal of Low Power Electronics (JOLPE)*. 26.

[24] Amit Kumar Singh, Alok Prakash, Karunakar Reddy Basireddy, Geoff V. Merrett, and Bashir M. Al-Hashimi. 2017. Energy-Efficient Run-time Mapping and Thread Partitioning of Concurrent OpenCL Applications on CPU-GPU MPSoCs. *ACM Trans. Embedd. Comput. Syst.* (2017), 22p.

[25] E. Del Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini. 2016. Workload-aware power optimization strategy for asymmetric multiprocessors. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. 531–534.

[26] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. 2012. Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE). In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12)*. IEEE Computer Society, Washington, DC, USA, 213–224. http://dl.acm.org/citation.cfm?id=2337159.2337184

[27] Hao Wang, Vijay Sathish, Ripudaman Singh, Michael J. Schulte, and Nam Sung Kim. 2012. Workload and Power Budget Partitioning for Single-chip Heterogeneous Processors. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT '12)*. ACM, New York, NY, USA, 401–410. https://doi.org/10.1145/2370816.2370873

[28] Hao Wang, Ripudaman Singh, Michael J. Schulte, and Nam Sung Kim. 2014. Memory Scheduling Towards High-throughput Cooperative Heterogeneous Computing. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation (PACT '14)*. ACM, New York, NY, USA, 331–342. https://doi.org/10.1145/2628071.2628096