Adaptive Energy Minimization of OpenMP Parallel Applications on Many-Core Systems

Rishad A. Shafik, Anup Das, Sheng Yang, Geoff V. Merrett & Bashir M. Al-Hashimi
School of ECS, University of Southampton, SO17 1BJ, UK, *e-mail:* {*ras1n09,akd1g13,gvm,sy2u12,bmah*}*@ecs.soton.ac.uk*

Abstract—**Energy minimization of parallel applications is an emerging challenge for current and future generations of many-core computing systems. In this paper, we propose a novel and scalable energy minimization approach that suitably applies DVFS in the sequential part and jointly considers DVFS and dynamic core allocations in the parallel part. Fundamental to this approach is an iterative learning based control algorithm that adapt the voltage/frequency scaling and core allocations dynamically based on workload predictions and is guided by the CPU performance counters at regular intervals. The adaptation is facilitated through performance annotations in the application codes, defined in a modified OpenMP runtime library. The proposed approach is validated on an Intel Xeon E5-2630 platform with up to 24 CPUs running NAS parallel benchmark applications. We show that our proposed approach can effectively adapt to different architecture and core allocations and minimize energy consumption by up to 17% compared to the existing approaches for a given performance requirement.**

Keywords*—*Many-core, OpenMP, Energy minimization.

I. Introduction

Silicon technology scaling has enabled the fabrication of many interconnected cores on a single chip for current and future generations of computing systems. The emergence of such systems has facilitated computing performance at unprecedented levels with application parallelization and architectural support. However, higher device-level integration and operating frequency in these systems have rendered exponentially increased power density and energy consumption [1]. Hence, minimizing the energy consumption, while delivering the required performance is a key design challenge for many-core applications [2], [5].

The continuing performance growth of many-core applications has been facilitated by parallel programming models. OpenMP is one such programming model, considered as the *de facto* standard of shared memory multiprocessing [3]. It features compiler-enabled annotations that can achieve data- or task-level parallelization. The parallelization is facilitated by runtime libraries that can allocate the number of computing nodes and suitably schedule the parallel tasks and threads during runtime to achieve high performance [9].

Over the years, there has been growing interest in OpenMP-based dynamic adaptation between energy and performance trade-offs of parallel applications [4]. Dynamic voltage/frequency scaling (DVFS) is a major runtime control knob to achieve such adaptation. The main principle of DVFS is to suitably lower the operating voltage/frequency to exponentially reduce the energy consumption at the cost of linear performance degradation [5]. Shirako *et al.* [10] proposed a DVFS-based energy minimization approach using a modified OpenMP compiler, named OSCAR. The compiler analyses the criticality of various parallel tasks and sections and identifies suitable DVFS for them. Cochran *et al.* [11] proposed another adaptive DVFS approach to control the peak power budget of an application. The approach benefits from restricting thread executions to a given number of processing cores to maximize the performance with a given power budget.

Another effective control knob for energy and performance adaptation during runtime is software dynamic concurrency throttling (DCT). DCT selects the number of concurrent processing cores and the threads running on them during runtime to manage application parallelism and trade-off performance for energy consumption. Porterfield *et al.* [6] showed a DCT control approach highlighting a study of the energy and performance variations (from 20% to 2X) for various core allocations. Based on the study an OpenMP-based adaptive runtime core allocation method was shown using CPU performance counters at regular intervals.

Researchers have also considered both DVFS and DCT control knobs synergestically to achieve minimized energy consumption, while maintaining a required performance target. Matthew *et al.* [7] presented one such runtime control approach, which benefits from offline training to learn the system architecture. The training is then followed by online performance prediction as a function of the system configuration and events to guide the runtime optimization and adaptation. Among others, Hwang and Chung [8] showed another runtime energy minimization approach considering joint DVFS and DCT control. Their approach is facilitated through statistical offline learning and implemented using runtime code insertion.

Existing OpenMP-based energy minimization approaches for parallel applications have the following limitations. Firstly, existing approaches [5]–[7] ignore energy minimization in the sequential computation part, which constitutes a major performance component in many-core applications. Secondly, these approaches [7], [8] employ DVFS and/or DCT using offline training processes to learn the system architecture and control parameters. As a result, the scalability is poor for applications with different many-core architectural allocations.

To address the above limitations and minimize energy consumption of many-core parallel applications effectively, this paper makes the following ***contributions***:

- We propose a novel energy minimization approach that considers DVFS control in the sequential part, and joint DVFS and DCT control in the parallel part of the applications, facilitated through OpenMP performance annotations in these parts.
- Fundamental to this approach are scalable and adaptive DVFS and DCT control algorithms that can iteratively learn the optimized control knobs, guided by the feedback from the CPU performance counters.
- Our approach is implemented in a modified OpenMP runtime library and is validated on a many-core platform [13] running NAS benchmark applications [12].

The remainder of this paper is organized as follows. Section II motivates the proposed approach, while Section III details the performance annotation and energy minimizations in the sequential and parallel parts of the application. Section IV validates the effectiveness and scalability of the approach. Finally, Section V concludes the paper.

II. Motivation

Parallel applications usually consist of sequential and parallel parts with the performance contribution of the sequential parts varying significantly between applications [5]. Existing works [5]–[7] primarily focus on the performance and energy trade-offs in the parallel parts using DVFS and/or DCT controls. However, for effective energy minimization sequential parts also need to be carefully controlled through DVFS, as it can potentially create opportunities for further energy savings. Moreover, the energy saved by the sequential parts can also be used to increase the performance of the parallel part for a given energy budget.

To demonstrate the importance of energy minimization in both sequential and parallel parts, Fig. 1.(a) and (b) show the execution times and energy consumptions of *bt* application from NAS parallel benchmarks used as a case study [12] (with an input of small solution set size of (24x24x24)) for different number of parallel core allocations, showing comparative contributions of parallel and sequential parts. The execution times were recorded on an Intel Xeon E5-2630 platform [13] in Linux OS environment. The energy consumptions were measured using *LIKWID* [15], which is a performance and energy profiling utility based on x86 generations of Intel processors. The measurements were carried out at nominal voltage/frequency levels (2.6 GHz at 1.35V) and at applied voltage/frequency scaling (VFS) of 1.2 GHz (at 0.98V) for all CPU cores. From the figures, the following two observations can be made:

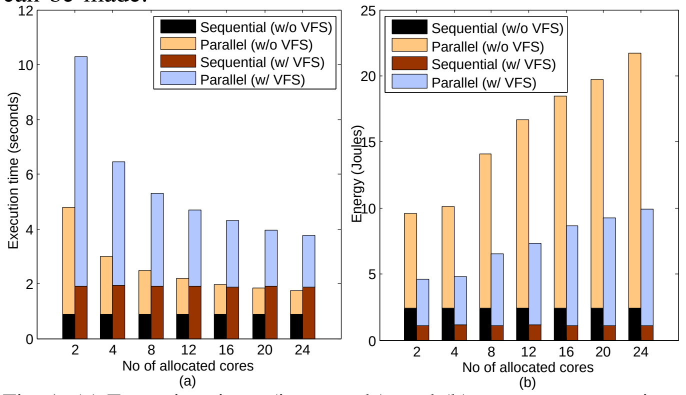


Fig. 1: (a) Execution times (in seconds), and (b) energy consumptions (in Joules) in sequential and parallel parts of *bt* application for varying core allocations (with and without VFS)

Observation 1: Referring to Fig. 1.(a), the sequential part of the parallel application *bt* cannot be ignored when overall execution time is considered. As can be seen, the relative contribution of the sequential part execution time increases as the number of parallel cores increases for both VFS scaling options (with VFS and without VFS). This is because the execution time of the parallel part decreases as concurrency increases with increased number of cores.

Observation 2: Referring to Fig. 1.(b), it can be seen that uncontrolled parallelism can cause the energy consumption of the parallel part to increase substantially without VFS. This is because with increased core allocations, the total sum of core energy consumptions (both dynamic and leakage) also increases compared to the energy consumption of the sequential part, which does not vary with core allocations. As can be seen, with VFS scaling this energy can be drastically reduced at the expense of reduced application performance in both sequential and parallel parts (Fig. 1.(a)).

From the above observations, it is evident that to achieve effective energy minimization, while maintaining a required performance level, DVFS control must be applied for both sequential and parallel parts of the application (Observation 1). Moreover, to control the increase in the energy consumption in the parallel part, concurrency must be managed dynamically by limiting the number of active cores for the given performance level (Observation 2). However, such energy minimization approach must be scalable for arbitrary number of architectural allocations and configurations. Underpinning the above observations this paper proposes a novel and scalable energy minimization approach for parallel applications, capable of reducing energy consumptions for both sequential and parallel parts under given performance requirements.

III. Proposed Energy Minimization Approach

Fig. 2 shows the proposed adaptive energy minimization approach organized in three steps, highlighting the interactions between application, runtime and hardware. In the first step, performance annotations are incorporated in the sequential and parallel parts of the application codes. These annotations, defined within and compiled by the modified OpenMP library (*libgomp*), communicate the approximate execution time requirements to the runtime. The runtime, which consists of the OpenMP library and OS routines, uses these times to guide the energy minimization steps for both sequential and parallel parts of the application through DVFS and DCT controls, guided by the monitored performance counters at regular intervals. The proposed energy minimization steps are further detailed in the following.

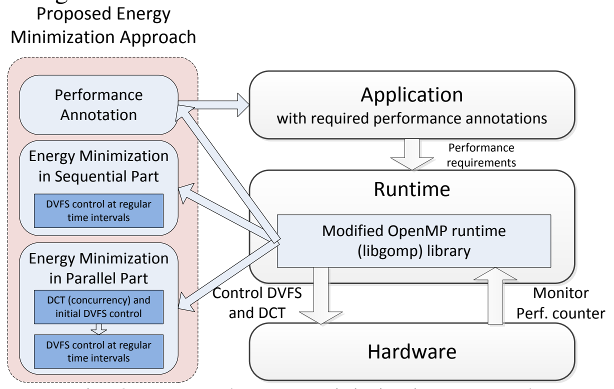

Proposed Energy Minimization Approach
Performance Annotation
Energy Minimization in Sequential Part
DVFS control at regular time intervals
Energy Minimization in Parallel Part
DCT (concurrency) and initial DVFS control
DVFS control at regular time intervals
Application
with required performance annotations
Performance requirements
Runtime
Modified OpenMP runtime (libgomp) library
Control DVFS and DCT
Monitor Perf. counter
Hardware

Fig. 2: Proposed energy minimization approach

A. *Performance Annotation*

Performance annotation in the application codes is carried out to enable energy minimization with specified performance requirements (Fig. 2). This is done through annotating both sequential and parallel parts with approximate maximum execution times (AMETs) and required approximate execution times (AETs) as a pair. To evaluate the AMETs the application codes are instrumented using thread-safe OpenMP *omp_get_wtime()* function and executed on a single-core running at the lowest operating frequency (f_{min}). The required AETs (which is the second part in the pair) can be specified by the application developer arbitrarily using AMETs as a guiding principle (since AET $\leq$ AMET) or systematically through evaluation of the target parallel speedup using the Amdahl's law [16] as

$$S = \frac{1}{\sum_{i=1}^{A_{seq}} T_{AET_i}^{seq} + \frac{\sum_{j=1}^{A_{par}} T_{AET_j}^{par}}{N}}, \tag{1}$$

where $T_{AET_i}^{seq}$ is the AET of the i-th sequential part, $T_{AET_i}^{par}$ is the AET of the i-th parallel part, A_{seq} is the total number of sequential parts in the application, A_{par} is the total number of parallel parts in the application and N is the maximum number of allocated cores. Optimizing the AET values for the sequential and parallel parts to achieve the best possible speedup of a parallel application is beyond scope of this paper (interested readers are referred to [16] for further details).

With the given AETs, different parts of the application are then exercised differently by the DVFS and DCT trade-offs with an aim to minimize the overall energy consumption, while maintaining the application performance target, given by the total execution time (T_{req}), which can be expressed as

$$T_{req} = \sum_{i=1}^{A_{seq}} T_{AET_i}^{seq} + \sum_{j=1}^{A_{par}} T_{AET_j}^{par}. \tag{2}$$

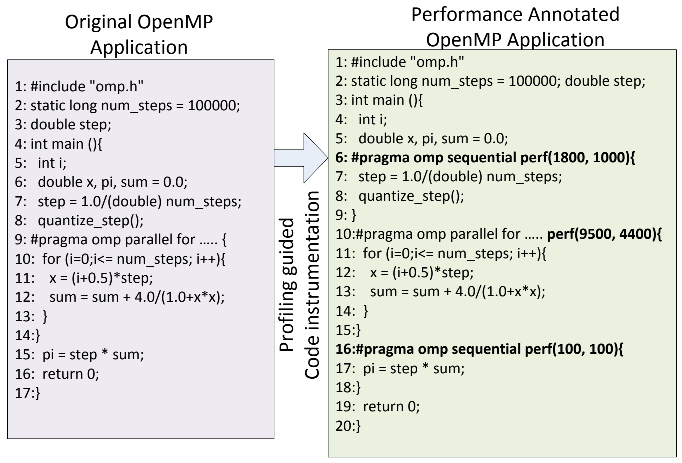
Following the evaluations of AMETs and AETs, the sequential and parallel parts are annotated as follows:

Sequential Part Annotation:

The sequential parts are enclosed by a newly incorporated "*#pragma omp sequential perf(AMET, AET)*" annotation. The "*#pragma omp sequential*" part of the annotation implies that the sequential part of the application codes will now be executed by the OpenMP runtime library, considering it as a special case of existing "*parallel*" annotation with the number of cores limited to 1 and affinity limited to thread 0 (i.e. master thread). The "*perf*(AMET, AET)" part of the annotation communicates the approximate maximum and required execution times (both in milliseconds) to the runtime for energy minimization with a specified performance requirement (see Section III-B).

Parallel Part Annotation:

The parallel parts are usually enclosed by "*#pragma omp parallel*" or "*#pragma omp task*" annotations etc. These annotations are further extended by adding "*perf*(AMET, AET)" with them. Similar to the sequential part, this annotation communicates the maximum and required execution times (both in milliseconds) to the runtime.


Original OpenMP Application
Performance Annotated OpenMP Application
Profiling guided Code instrumentation

```
1: #include "omp.h"
2: static long num_steps = 100000;
3: double step;
4: int main (){
5:   int i;
6:   double x, pi, sum = 0.0;
7:   step = 1.0/(double) num_steps;
8:   quantize_step();
9: #pragma omp parallel for ..... {
10:  for (i=0;i<= num_steps; i++){
11:    x = (i+0.5)*step;
12:    sum = sum + 4.0/(1.0+x*x);
13:  }
14:}
15:  pi = step * sum;
16:  return 0;
17:}
```

```
1: #include "omp.h"
2: static long num_steps = 100000; double step;
3: int main (){
4:   int i;
5:   double x, pi, sum = 0.0;
6: #pragma omp sequential perf(1800, 1000){
7:   step = 1.0/(double) num_steps;
8:   quantize_step();
9: }
10:#pragma omp parallel for ..... perf(9500, 4400){
11:  for (i=0;i<= num_steps; i++){
12:    x = (i+0.5)*step;
13:    sum = sum + 4.0/(1.0+x*x);
14:  }
15:}
16:#pragma omp sequential perf(100, 100){
17:  pi = step * sum;
18:}
19:  return 0;
20:}
```

Fig. 3: Example of performance annotated OpenMP application code (in C), showing the original OpenMP application code on the left

Fig. 3 shows example application codes (in C) showing such performance annotation in both sequential and parallel parts. The original OpenMP application code is shown on the left, while the performance annotated application code is shown on the right. As can be seen, the original sequential statements in lines 7-8 (left) are grouped as the first sequential part and enclosed using "*#pragma omp sequential*" annotation in lines 6-9 in the annotated application (right). The annotation is followed by "*perf(1800, 1000)*", which indicates the sequential part is expected to have the maximum execution time of about 1800 ms and required execution time is 1000 ms. Similar sequential performance annotation is also carried out in lines 16-18 (right), which has equal AMET and AET of 100 ms. The parallel part, enclosed by "*#pragma omp parallel for*", is annotated using additional "*perf*" annotation, with AMET of 9500 ms, followed by the required AET of 4400 ms. With the given performance annotations, the application is expected to incur approximate execution time (1000+4400+100) = 5500 ms (given by (2)).

B. Energy Minimization in the Sequential Parts

With the specified AETs in the sequential parts, the energy minimization in the sequential parts is carried out through DVFS based on the predicted workloads at regular intervals (Fig. 2). To effectively predict the time-varying workload at each interval, exponentially weighted moving average (EWMA) is used as the prediction scheme, similar to [18]. Using this scheme, the predicted workload at the t^{th} interval, $\hat{\mathcal{C}}_t$ (in CPU cycles), is given by [18] as

$$\hat{\mathcal{C}}_t = \omega \mathcal{C}_{t-1} + \sum_{i=1}^{D} (1-\omega)^i \, \mathcal{C}_{t-i} \quad , \tag{3}$$

where $\mathcal{C}_t$ and $\mathcal{C}_i$ are the previous observed workloads (in CPU cycles) at the t^{th} and i^{th} decision epochs, $1 \leq i \leq D$, ω is the moving average coefficient and D is the window size (ω and D are evaluated empirically for higher prediction accuracy). Based on the predicted workload $\hat{\mathcal{C}}_t$ in (3) the operating frequency at the t-th interval (f_t^{seq}) is determined by the iterative learning control (ILC) function as

$$f_t^{seq} = f_{min} + \Delta f k_t K_1; \; if \; \mathcal{E}_{t-1}^{seq} \approx 0 \tag{4}$$

$$f_t^{seq} = f_{t-1}^{seq} - \Delta f K_2 \mathcal{E}_{t-1}^{seq}, \text{ otherwise} \tag{5}$$

where f_{t-1}^{seq} is the previous operating frequency, Δf is the frequency differential, K_1, K_2 are constants defining scaling steps, f_{min} is the minimum operating frequency of the system, $\mathcal{E}_{t-1}^{seq}$ is the performance error incurred due to previous control actions and k_t is workload dependent variable, given by

$$k_t = \exp\left[K_2\left(\frac{\hat{\mathcal{C}}_t}{f_{max} \times \Delta t} - 1\right)\right], \tag{6}$$

where Δt is the time interval and f_{max} is the maximum processor core frequency in the system. The performance error ($\mathcal{E}_{t-1}^{seq}$) in (4) is evaluated as

$$\mathcal{E}_{t-1}^{seq} = \sum_{i=1}^{t-1} \left(\frac{\mathcal{C}_i}{f_{ref}^{seq}} - \frac{\mathcal{C}_i}{f_i^{seq}} \right), \tag{7}$$

where $\mathcal{C}_i$ is the actual workload and f_i^{seq} is the chosen operating frequency at the i-th interval, f_{ref}^{seq} is the static reference frequency for the sequential execution determined using the performance annotations provided as

$$f_{ref}^{seq} = f_{min} \times \frac{T_{AMET}^{seq}}{T_{AET}^{seq}}. \tag{8}$$

The Δf, f_{max} and f_{min} values can be obtained from the OS (in the case of Linux, from *sysfs* variable). The ILC functions in (4) and (5) achieve the following:

- in the case of over-performance (positive $\mathcal{E}_{t-1}^{seq}$) or under-performance (negative $\mathcal{E}_{t-1}^{seq}$), it decreases or increases the frequency in steps; however, if $\mathcal{E}_{t-1}^{seq} \approx 0$, the clock frequency does not change, thus reducing the energy consumption,
- Due to predicted workload based k_t formulation, it suitably scales and decreases or increases the operating frequency to optimize for the CPU utilization for positive or negative $\mathcal{E}_{t-1}^{seq}$ values, respectively, and
- $\mathcal{E}_{t-1}^{seq}$, given by (7), accounts for the performance errors caused by the workload mispredictions.

Energy minimization through ILC is implemented as an additional timer-based thread in the OpenMP runtime library with a period of Δt. This thread is statically assigned the same affinity as the main master thread (i.e. thread 0).

Algorithm 1 DCT and initial DVFS control algorithm for energy minimization in the parallel part

```
Require: $T^{par}_{AMET}$ and $T^{par}_{AET}$
 1: Reference parallel clock frequency $f^{par}_{ref} = f_{min} \times \frac{T^{par}_{AMET}}{T^{par}_{AET}}$
 2: for each new thread joining or forking do
 3:     for $n$ (number of cores): 2 to $N_{max}$ do
 4:         for $f$ (clock frequency): $f_{min}$ to $f_{max}$ do in $\Delta f$ steps
 5:             Evaluate approximate parallel speedup, $\hat{S}$
 6:             Evaluate expected parallel clock frequency $f^{par}_{exp}$
 7:             if $f^{par}_{exp}$ is closest to $f^{par}_{ref}$ then
 8:                 Current $f^{par}_{t}$ = $f$, core allocation, $\hat{N}$=$n$
 9:             end if
10:         end for
11:     end for
12: end for
```

C. Energy Minimization in the Parallel Parts

Energy minimization in the parallel parts is carried out using both DCT and DVFS controls (Fig. 2). These controls are established in two stages. In the first stage, the DCT and DVFS control decisions are taken at the beginning of the parallel part, which is further updated every time threads join or leave. Algorithm 1 shows the DCT and DVFS control algorithm used in the parallel part. As can be seen, initially the total reference parallel frequency (f^{par}_{ref}) is determined (line 1) using the given performance annotations as

$$f^{par}_{ref} = f_{min} \times \frac{T^{par}_{AMET}}{T^{par}_{AET}}. \tag{9}$$

Then, to evaluate the equivalent core allocation with per core operating frequencies, the algorithm iterates from the minimum number of cores 2 to N_{max} (lines 3–11). Also, for each core allocation the operating frequency (f) is iterated in Δf steps starting from the lowest operating frequency, f_{min}, to the maximum operating frequency, f_{max} (lines 4–10). For a given core allocation and operating frequency, the approximate parallel speedup ($\hat{S}$) is evaluated using (1) as

$$\hat{S} = \min\left(\frac{d}{d+\frac{1-d}{n}}, \frac{d}{d+\frac{1-d}{M}}\right), \tag{10}$$

where d is a constant related to data dependency among parallel threads arising from shared memory multiprocessing, and M is the number of currently participating threads (the maximum number of threads is limited by *OMP_NUM_THREADS* environment variable). In this work, d is empirically evaluated as 0.1 as it was found to model parallelism well for most NAS benchmark applications. The evaluated $\hat{S}$ is then multiplied with the current f to calculate the expected parallel frequency, i.e. f^{par}_{exp}=$\hat{S}f$. The minimum n and f, for which the f^{par}_{exp} is found to be the closest to f^{par}_{ref}, are chosen as the current concurrent core allocation ($\hat{N}$) and operating frequency (f^{par}_{t}).

In the second stage, the DVFS controls are further refined at every time interval based on the predicted workloads at regular intervals (Fig. 2). The workload prediction guided DVFS controls in the parallel part is carried out using the ILC function, similar to (3), (4) and (5). However, as the target platform supports common operating voltage and frequency per socket, the control is simplified through determination of a single operating frequency for all parallel cores. Such control of the operating frequencies is guided by the performance counters, which monitor the actual workloads after each interval. Using the actual workloads per core, the performance error in the parallel execution ($\mathcal{E}^{par}_{t-1}$) is evaluated as

$$\mathcal{E}^{par}_{t-1} = \sum_{n=1}^{N}\sum_{i=1}^{t-1}\left(\frac{\mathcal{C}_{i,n}}{f^{par}_{ref}} - \frac{\mathcal{C}_{i,n}}{f^{par}_{i,n}}\right), \tag{11}$$

where $\mathcal{C}_{i,n}$ is the actual workload and $f^{par}_{i,n}$ is the chosen operating frequency at the i-th interval of the n-th core.

Similar to the sequential part, energy minimization through ILC principles is implemented as a parallel thread, which is executed in the same core as the master thread through thread affinity control.

IV. Experimental Results

To validate the effectiveness of the proposed approach, a number of experiments are carried out. The experimental setup is further detailed below, followed by the results highlighting comparative evaluations and scalability of the proposed approach.

A. Experimental Setup

The proposed approach is experimented on Intel Xeon E5-2630 [13] platform, which has a total of 24 cores, organized in two sockets with 12 cores each. Each pair of cores within the platform shares 1536kB of L2 cache, six pairs within each socket share a 15MB L3 cache, and all cores have a shared memory of 32 GB. Each core operates at a minimum frequency of 1.2 GHz (at $V_{dd} = 0.98V$) and a maximum frequency of 2.6 GHz (at $V_{dd} = 1.35V$); there are also thirteen other intermediate frequencies increasing in 0.1 GHz steps. NAS application benchmarks of class B (medium) and C (large) are executed on Linux kernel version 2.6.32. These benchmarks feature wide variations in several execution properties, including parallelization annotations, varying sequential parts and compute- and memory-boundedness of threads [14]. The applications were initially profiled and performance annotated with execution times (Section III-A). The performance counters at 100 ms regular intervals and and energy measurements of these applications were carried out using the *LIKWID* [15] library. For the ILC functions in (4) and (5) K=15 and K=3 were used for sequential and parallel parts. All measurements are averaged over three executions.

B. Case Study

To illustrate how the energy minimization is carried out in the proposed approach, Fig. 4 shows the different runtime scenarios in the sequential (on the left) and parallel (on the right) computation parts of the *mg* application as a case study. The application was executed with 12 cores (maximum core allocation N=12, maximum concurrent threads=N). Fig. 4.(a) shows the predicted and actual CPU workloads, while Fig. 4.(b) and (c) show the correponding operating frequencies (f^{seq}_{t}) and performance errors ($\mathcal{E}^{seq}_{t-1}$) incurred during energy minimization in the sequential part. As can be seen, for the given predicted workloads the operating frequencies are chosen by the iterative learning control function with an aim to minimize the energy consumption (Fig. 4.(b)), while also reducing the performance error given by (4) and (5).

Fig. 4.(d) shows the result of DCT control decisions in the parallel part as a result of Algorithm 1, while Fig. 4.(e) and (f) show the correponding operating frequencies (f^{par}_{t}) and performance errors ($\mathcal{E}^{par}_{t-1}$). As can be seen, when two threads finish within the parallel part at the 39-th interval the algorithm dynamically chooses the throttle concurrency by reducing the core allocations from 7 cores to 5 cores. This is because with such allocation, the best speedup versus energy trade-off is achieved (see Section III-C). However, when a thread is forked out at the 93-rd interval, the DCT algorithm increases the core allocations to 6. It is to be noted that during such DCT controls, the DVFS controls are perturbed. As a result, the ILC starts to react by changing the operating frequency. For example, after the 39-th interval, due to reduced concurrency initially the parallel section starts to under-perform. However,

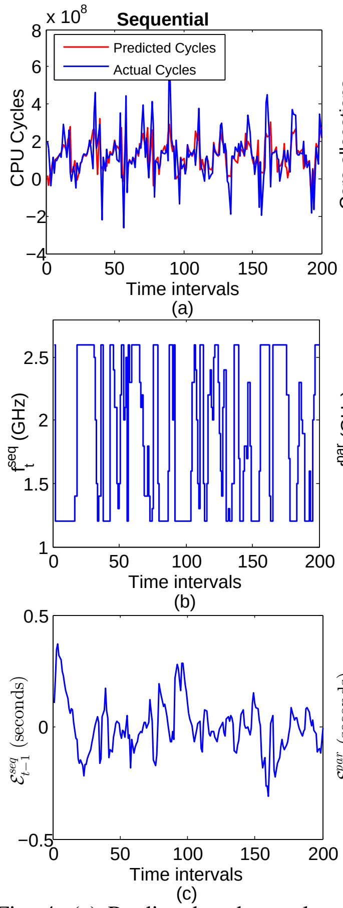


Fig. 4: (a) Predicted and actual workloads (b) operating frequencies (f_i^{seq}), (c) performance error ($\mathcal{E}_{t-1}^{seq}$), and (d) core allocations (Algorithm 1) (e) operating frequencies (f_i^{seq}), and (f) performance error ($\mathcal{E}_{t-1}^{par}$), for the *mg* application (class B)

as the operating frequencies are scaled up, the performance error decreases. On the other hand, when a thread joins in at the 93-rd interval, the parallel part starts to over-perform. At this time, the ILC decreases the operating frequency (Fig. 5.(b) and (e)).

C. Performance and Energy Trade-offs

Fig. 5 shows the performance and energy consumption trade-offs using the proposed energy minimization approach for various benchmark applications with the maximum number of threads N=24. Fig. 5.(a) and (b) show the execution times and energy consumptions for $\frac{AET}{AMET}$ ratios of 0.75 and 0.3 (on average) for the sequential and parallel parts, respectively. Fig. 5.(c) and (d) show the same when energy minimization is carried out for $\frac{AET}{AMET}$ ratios of 0.75 and 0.45 for the sequential and parallel parts. These two cases demonstrate the impact of varying the performances of the parallel parts. As can be seen, due to 15% increase in the execution time of the parallel parts energy consumption is reduced by up to 11% in the case of *sp* (class C) (Fig. 5.(c) and (d)). However, such decrease in the performance of the parallel parts reduces the number of cores allocated in the DCT control (Algorithm 1), which in turn, also reduces the thread synchronization and interrupt times [17]. As a result, the overall execution time does not increase substantially; up to 8% increase is noticed in the case of *is* (class B and C).

Fig. 5.(e) and (f) compares the execution times and energy consumptions for $\frac{AET}{AMET}$ ratios of 0.1 and 0.45 for the sequential and parallel parts. Together with Fig. 5.(c) and (d) the two cases demonstrate the impact of increasing the performances

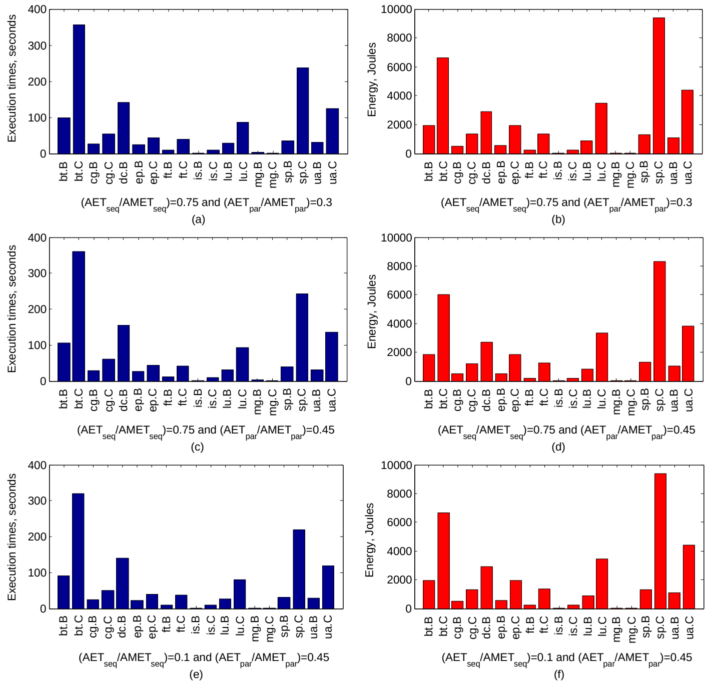


Fig. 5: Performance and energy trade-offs using the proposed approach

of the sequential parts by 65%. Such increase in the sequential performance requires using higher DVFS (see Section III-B), which is reflected by increased energy consumption by up to 15% in the case of *dc* (class B) and *sp* (class C). It is to be noted that the increased performance in sequential part in these applications is observed through 9% and 10% higher execution times in these applications.

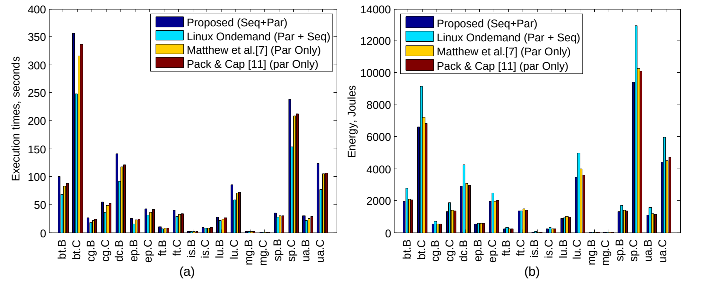


Fig. 6: Comparative evaluation of approaches in terms of (a) performance (execution times, in seconds) and (b) energy consumptions

D. Comparative Evaluations

To comparatively evaluate the proposed approach with the existing approaches, Fig. 6 shows the performances and energy consumptions of different benchmark applications using four different energy minimization approaches: the proposed approach, Linux's ondemand governor [19] running on all processor cores (as an example of energy minimization in both sequential and parallel), and approaches proposed in [7] and [11] (as examples of energy minimization in parallel parts only). Fig. 6.(a) shows the performances in terms of execution times (in seconds), while Fig. 6.(b) shows the energy consumptions (in Joules). The energy minimization approaches proposed in [7] and [11] were implemented using similar performance requirements in the parallel parts as the proposed approach (equivalent to $\frac{AET_{par}}{AMET_{par}} = 0.5$) with given parameter values. In the case of Pack & Cap approach [11] the performance requirements were adjusted with power budget control. The energy minimization approach using the ondemand governor could not be performance constrained, as it does not allow such provisions. As can be seen, the proposed approach outperforms the existing approaches in terms of energy minimization, reducing the energy by up to 17% (on average) when compared with the ondemand governor running on the processor cores, and by up to 10% and 7% when compared with the approach proposed in [7] and [11]. The reduction in energy minimization is achieved due to the following two reasons. Firstly, with higher required execution times the proposed approach can effectively relax the processor clock frequencies using ILC at regular time intervals, thus reducing the energy and meeting the required performance. Secondly, the proposed approach benefits from energy minimization in both sequential and and parallel parts, effectively reducing the overall energy consumption. The energy minimization approaches [7] and [11] do not have such capabilities.

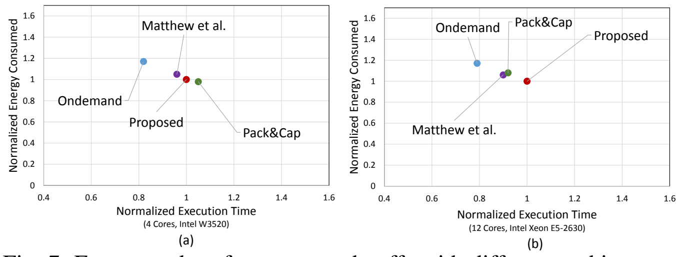


Fig. 7: Energy and performance trade-offs with different architectures and core allocations

E. Architectural Scalability

To validate the scalability of our proposed approach, we further experimented running *sp* application (class C) on two different architectures: a 4 cores system with Intel W3520 processors and a 12 cores system with Intel E5-2630 processors for similar performance requirements in the parallel parts for the approaches: [7], [11] and the proposed one. Both approaches [7] and [11] required extensive architecture-specific offline training using multinomial logistic regression based classification to establish the relationships between required performance and performance counters. The energy minimization in the proposed approach was carried out without any extensive training (only initial performance profiling was carried out to enable the performance annotations, Fig. 2). Fig. 7(a) and (b) show the energy versus performance trade-offs of the approaches showing normalized energy consumption and execution times. The energy consumption and execution times are normalized with respect to the same by the proposed approach. With such normalization, the execution time of lower than 1 means over-performance and higher than 1 means under-performance, while the energy of lower than 1 means less energy consumption, higher than 1 means more energy consumption compared to the reference; close to 1 for both means effective energy minimization, provided the performance is also on par.

As can be seen, the proposed approach continues to minimize energy for change of architecture and also with increased number of maximum core allocations. The approaches [7], [11], however, does not minimize energy effectively with such change of architecture allocation. This is because, as the number of cores increase the offline training based relationships in these approaches become harder to adapt due to runtime variations of the performance counters. The proposed approach continues to minimize energy effectively as the DVFS control considers higher CPU utilization with reduction of the performance errors. Moreover, based on the system architectures frequency controls, appropriate concurrent allocations are carried out at each thread creation and exit (See Section III).

V. Conclusions

An adaptive energy minimization approach for OpenMP parallel applications is proposed. The adaptation is facilitated through performance annotations in sequential and parallel parts of the applications, defined in the modified OpenMP runtime library. Using these performance annotations, the proposed approach suitably applies iterative learning control based DVFS using predicted workloads and feedback from the CPU performance counters. Moreover, dynamic concurrency is controlled by a DCT control algoroithm to limit the number of active cores and achieve energy minimization. The proposed approach is validated on a many-core platform running various NAS parallel benchmark applications, showing up to 17% reduced energy compared to the existing approaches. Energy-delay product based adaptive optimization and impact of overheads are currently being considered for future research.

References

[1] H. Esmaeilzadeh *et al.*. Dark silicon and the end of multicore scaling. *38th ISCA*, pp.365–376, 4-8 June, 2011.

[2] R. Zamani *et al.*. A feasibility analysis of power-awareness and energy minimization in modern interconnects for high-performance computing. *Cluster Computing, IEEE Intl. Conf. on*, pp.118–128. 2007.

[3] OpenMP. Open Multi-Processing [Online]: http://www.openmp.org/ [Accessed]: 18 July. 2014.

[4] M. Etinski *et al.*. Understanding the future of energy-performance trade-off via DVFS in HPC environments. *Journal of Parallel and Distributed Computing*, vol. 72, no. 4, pp.579-590, 2012.

[5] J. Li, and J.F. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. *in HPCA*, pp.77–87. 2006.

[6] A.K. Porterfield *et al.*. Power Measurement and Concurrency Throttling for Energy Reduction in OpenMP Programs. *Parallel and Distributed Processing Symposium*, pp.884–891. 2013.

[7] C-M. Matthew *et al.*. Prediction models for multi-dimensional power-performance optimization on many cores. *Proc. of the Intl. Conf. on Parallel architectures and compilation techniques*, pp.250–259. ACM, 2008.

[8] Y. Hwang, and K. Chung. Dynamic power management technique for multicore based embedded mobile devices. *IEEE Trans. on Industrial Informatics*, vol.9, no.3 pp.1601–1612, 2013.

[9] Y. Dong *et al.*. Energy-oriented OpenMP parallel loop scheduling. *in ISPA*, pp.162–169. 2008.

[10] J. Shirako *et al.*. Compiler control power saving scheme for multi core processors. Ch. in *Languages and Compilers for Parallel Computing*, pp.362–376. Springer Berlin Heidelberg, 2006.

[11] R. Cochran *et al.*. Pack & Cap: Adaptive DVFS and thread packing under power caps. *44th MICRO*, pp.175–185. ACM, 2011.

[12] NAS Parallel Benchmarks. NASA Advanced Supercomputing Division. [Online]: www.nas.nasa.gov [Accessed]: 18 July. 2014.

[13] Intel Xeon E5-2630. Intel® Xeon® Processor E5-2630 Family (15M Cache, 2.3GHz) [Online]: http://ark.intel.com/products/64593/Intel-Xeon-Processor-E5-2630-15M-Cache-2_30-GHz-7_20-GTs-Intel-QPI

[14] A. Ramachandran *et al.*. Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi. *in 42nd ICPP*, pp.736–743. 2013.

[15] J. Treibig *et al.*. LIKWID: Lightweight Performance Tools. Ch. in *Competence in High Performance Computing*, pp.165–175. Springer Berlin Heidelberg, 2012.

[16] M. Hill, M.R. Marty. Amdahl's Law in Multicore Era. *IEEE Computer*, Vol. 41, no. 7, pp.33–38, 2008.

[17] 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 *SIGARCH Comput. Archit. News*, Vol.36(1), , pp.277–286, Mar, 2008.

[18] S. Sinha, J. Suh, B. Bakkaloglu and Y. Cao. Workload-Aware Neuromorphic Design of the Power Controller. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol.1, no.3, pp.381–390, 2011.

[19] V. Pallipadi and A. Starikovskiy. The ondemand governor. *Proceedings of the Linux Symposium*, pp. 215–229, 2006.