Approximation-Aware Scheduling on Heterogeneous Multi-core Architectures

Cheng Tan[1], Thannirmalai Somu Muthukaruppan[1], Tulika Mitra[1], and Lei Ju[2]
[1]School of Computing, National University of Singapore
[2]School of Computer Science and Technology, Shandong University
Email: {tancheng,tsomu,tulika}@comp.nus.edu.sg, julei@sdu.edu.cn

Abstract

The high performance demand of embedded systems along with restrictive thermal design power (TDP) constraint have lead to the emergence of the heterogenous multi-core architectures, where cores with the same instruction-set architecture but different power-performance characteristics provide new opportunities for energy-efficient computing. Heterogeneity introduces challenges in scheduling the tasks to the appropriate cores and selecting the frequency assignment of each core. In this paper, we introduce an approximation-aware scheduling framework for soft real-time tasks on the heterogeneous multi-core architectures. We consider multiple versions of a task obtained by introducing approximation in the computation to provide different levels of quality of service (QoS) versus performance tradeoffs. The additional choice of approximation allows us more flexibility in meeting the performance and TDP constraints while maximizing QoS per unit of energy.

I. Introduction

Heterogeneous multi-core architectures, such as ARM big.LITTLE [5], combine power-efficient simple cores with power-hungry complex cores on the same chip in addition to multiple frequency levels per cluster of cores. Heterogeneity enables better match between application requirements and computation capabilities, and achieves substantially improved energy-efficiency [6, 1]. These architectures also provide a promising solution to the emerging dark silicon era where a chip can have many cores but power and thermal constraints demand that a significant fraction of them should be left unpowered (or dark) during execution [4, 2]. With heterogeneous architectures, only the cores most suitable for the current computation are switched on leading to energy-efficiency.

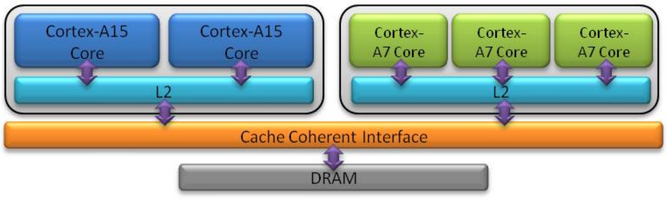

Fig. 1. ARM big.LITTLE heterogeneous multi-core architecture.

In this paper, we focus on scheduling soft real-time tasks on heterogenous multi-core architectures with strict TDP constraints. Apart from heterogeneity and power state manipulation (switching off unused cores), we exploit dynamic voltage and frequency scaling (DVFS) and approximate computing to meet the performance and TDP constraints while offering the best tradeoff between QoS and energy. While DVFS can help satisfy the TDP constraint through lower voltage-frequency operating point, the increased execution time of the tasks may lead to frequent deadline misses.

An alternative is to employ approximate computing where the execution time of a task is reduced at the cost of lower QoS. Traditionally, computing systems focus on delivering precise and accurate output. However, many modern workloads can easily accommodate minimal loss in accuracy and/or QoS. This inherent tolerance of applications for accuracy can be leveraged to improve the power-performance characteristics. For example, a video can be encoded at an expected frame rate (performance/deadline constraint) but at reduced image quality (QoS). The approximation enables execution of a task at lower frequency (low power) without impacting its performance but potentially diminishing the QoS.

Concretely, this paper integrates *computation approximation, task scheduling, and DVFS on heterogeneous multi-core architecture to minimize energy consumption and maximize QoS while satisfying the performance and TDP constraint.* Our key contributions are:

- We propose a framework to achieve energy-efficiency under the TDP constraint by prudently scheduling tasks, accommodating approximation, and managing DVFS.
- We propose an offline algorithm to search the complex design space consisting of task versions, DVFS, task schedule to obtain near-optimal solution.
- We propose a run-time scheduler to dynamically improve the QoS and energy efficiency further.
- We show the effectiveness of our approach with real applications on a real heterogeneous multi-core architecture.

II. Related Work

With the emergence of the multi-core processors, multi-core scheduling is a well-researched topic. In terms of heterogenous multi-core scheduling, Li et al. adapt the Linux scheduler to take into account the characteristics of the different cores in a heterogeneous architecture [11]. Both [8] and [9] modify the original scheduling policy to decide which core will be used for an incoming task. The energy cost will decline if a little core is selected while the execution time will be shortened if a big core

978-1-4799-7792-5/15/$31.00 ©2015 IEEE

is active. Such trade-offs provide an opportunity to improve the performance-per-watt behavior of the system. [6] proposes a scheduling framework for heterogeneous multi-core that satisfies performance demand under thermal design constraint and minimizes energy. In terms of real-time systems, [10] guarantees that the peak temperature of the chip remains under the threshold with a modified DVFS management. The peak temperature or power consumption is minimized in [12] through ILP or a heuristic algorithm. In this paper, we combine the task scheduling with the DVFS management together.

Earliest Deadline First (EDF) is a popular scheduling algorithm for hard real-time systems. [13] introduces a restricted migration-based scheduling strategy for the multi-cores, which permits the migration of tasks only at job boundary (defined as task-level migration in [14]). In this work, we modify the scheduling strategy in [13] to take into account heterogeneous multi-core architecture.

For graceful degradation of QoS while satisfying the performance and thermal constraints, we employ approximate computing. We create multiple versions of a task with the loop perforation [7] technique. It transforms loops by skipping some of the iterations such that the accuracy of the result is impacted but the execution time is reduced. None of the above energy-aware scheduling works consider approximation of the tasks that can potentially improve energy-efficiency significantly while having minimal impact on QoS.

III. Problem Formulation

Architecture and Application Our architectural model is based on a generalization of the ARM big.LITTLE heterogeneous multi-core [5]. The architecture consists of C clusters and E cores. Each cluster consists of a set of homogeneous cores. But different clusters have different core types. Given a core $e \in E$, let $ctype(e) \in C$ represent the cluster that core e belongs to. Each cluster c has $\mathcal{F}_c = \{f_c^1, \dots f_c^{F_c}\}$ discrete frequency levels where f_c^1 is the lowest frequency level and $f_c^{F_c}$ is the highest frequency level. All the cores in a cluster need to run at the same frequency level. We observe that the speedup at different frequency levels compared to the lowest frequency on a cluster is mostly agnostic to the workload. So we define $\{sf_c^1, \dots sf_c^{F_c}\}$ as the speedup at different frequency levels on cluster c normalized to the lowest frequency level, i.e., $sf_c^1 = 1$ and $sf_c^m \geq 1$ for $m = 1 \dots F_c$. Similarly, we assume that the active and idle power of a core is constant at a particular frequency level and is defined as AP_e^m and IP_e^m, respectively at frequency level m on core e. Note that active and idle power of all the cores within a cluster are identical.

Our application model consists of a set of independent, periodic, soft real-time tasks T. Each task has multiple versions where each version provides different tradeoffs in terms of QoS and execution time. Let us define $\mathcal{V}_t = \{v_t^1 \dots v_t^{V_t}\}$ as the V_t different versions of task $t \in T$. The first version v_t^1 represents the original task without any approximations, i.e., it has the best QoS and longest execution time among all the versions. Let sv_t^n be the speedup of the n^{th} version of task t compared to the original version. Clearly, $sv_t^1 = 1$ and $sv_t^n \geq 1$ for $n = 1 \dots V_t$. Note that we assume (and experimentally validated) that the speedup of a task version compared to the original is independent of the core type on which it is running. We also define q_t^n as the relative QoS of the n^{th} version of task t compared to the original version. That is $q_t^1 = 1$ and $q_t^n \leq 1$ for $n = 1 \dots V_t$.

We define $w_{t,c}$ as the WCET of the original version v_t^1 of task t running on cluster c at the lowest frequency level f_c^1 whereas $w_{t,c}^{m,n}$ is the execution time of the n^{th} version of task t on cluster c at the m^{th} frequency level. So $w_{t,c} = w_{t,c}^{1,1}$ and $w_{t,c}^{m,n} = \frac{w_{t,c}}{sv_t^n \times sf_c^m}$.

Let p_t be the period of task t and let HP be the hyper-period corresponding to the task set T. A task t has $J_t = \frac{HP}{p_t}$ jobs (task instances) denoted as $\mathcal{J}_t = j_t^1 \dots j_t^{J_t}$ within HP and let $J = \bigcup_{t \in T} \mathcal{J}_t$ be the set of all jobs within HP.

Problem Definition The notations used in the paper are summarized in Table I. Given a set of independent, periodic soft real-time tasks and a heterogeneous multi-core architecture with per-cluster DVFS management (all the identical cores within a cluster run at the same voltage-frequency level), our objective is to select the version for each job (task instance) and derive the task schedule and frequency assignment for the hyper-period that minimize energy consumption and maximize QoS while satisfying the thermal design power (TDP) constraint and the performance demand. Our solution consists of (a) an offline scheduling strategy based on the WCET values of each task and (b) an online strategy that improves upon the offline solution by exploiting the difference between the WCET and the actual execution time.

We use the following variables to define a concrete schedule and frequency assignment S. First the hyper-period HP is divided into time intervals such that the job executing on each core and the frequency assignment of each cluster remain unchanged throughout each interval. Given a time interval x, we define $task(e, x)$ as the task running on core e and $freq(c, x)$ as the frequency of cluster c during time interval x. We also let $ver(j_t^k)$ denote the version selected for the k^{th} instance of task t denoted by j_t^k. The variables $task(e, x)$ and $freq(c, x)$ for all time intervals and $ver(j_t^k)$ for all jobs concretely define a solution S with task schedule and frequency assignment.

In order for a solution to be valid, it has to satisfy the performance and the TDP constraint. The performance constraint requires that a job should complete execution before its deadline. For the k^{th} instance of a task t, we have to ensure the following: (a) the job can execute only during its own period, (b) the job can only be assigned to one core, and (c) the cumulative execution time of the job within its period should be equal to $\frac{w_{t,c}}{ver(j_t^k)}$. The cumulative execution time for the job within the period can be computed as

$$\sum_{\substack{\forall x \ within \ [(k-1)\times p_t, k\times p_t] \\ s.t. \ task(e,x)=t}} length(x) \times sf_c^m \qquad (1)$$

where x represents a time interval within the period for the k^{th} task instance ($[(k-1) \times p_t, k \times p_t]$) where for some core e, we have $task(e, x) = t$ and $c = ctype(e)$ is the cluster type of core e, $m = freq(c, x)$ is the frequency level assigned to cluster c during the time interval x, and $length(x)$ is the length of the time interval x.

To ensure the TDP constraint, we first compute the power consumption for the entire chip $P(x)$ for time interval x.

$$P(x) = \sum_{e\in E} \beta(e,x) \le TDP \tag{2}$$

$$\beta(e,x) = \begin{cases} AP_e^m & \text{if } task(e,x) \neq \bot \;\&\; freq(c,x) = m \\ IP_e^m & \text{if } task(e,x) = \bot \;\&\; freq(c,x) = m \\ 0 & \text{if } task(e,x) = \bot \;\&\; freq(c,x) = 0 \end{cases}$$

where $c = ctype(e)$ is the cluster containing core e. Note that $freq(c,x) = 0$ implies that the cluster c is shutdown in the time interval x. Similarly, $task(e,x) = \bot$ implies that the core c does not have a task mapped to it.

Objectives Our objective is to maximize the QoS per unit energy. First, we compute WE that represents the worst-case energy consumption under the scenario that we execute the original versions of all the tasks at the highest frequency level without any consideration to the TDP constraint. The total energy consumption within the hyper-period HP is the summation of the energy consumption for each interval x. Then the normalized energy consumption $ne(S)$ of a solution S is

$$ne(S) = \frac{\sum_x (length(x) \times P(x))}{WE} \tag{3}$$

We compute the normalized QoS of a solution $nq(S)$ w.r.t. the minimal QoS for each task t (WQ_t) expected by the user. We compute the QoS of a solution as the summation of the QoS of the version assigned to each job for each task.

$$nq(S) = \Big(\sum_{t\in T} \sum_{\substack{j_t^k \in \mathcal{J}_t \\ n=ver(j_t^k)}} \frac{q_t^n - WQ_t}{1 - WQ_t}\Big)/|J| \tag{4}$$

Our objective is to choose the solution S that maximizes the normalized QoS divided by the normalized energy $\frac{nq(S)}{ne(S)}$.

IV. Proposed Framework

An overview of our proposed framework is shown in Figure 2. The framework consists of two components (a) an off-line strategy to generate a near-optimal schedule and frequency assignment based on the worst-case execution time, and (2) a run-time scheduler to improve the QoS and conserve energy further based on the actual execution time of the tasks.

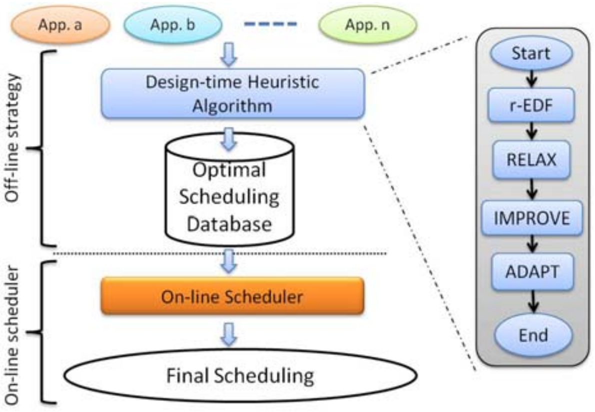

Fig. 2. An overview of the proposed framework.

TABLE I
Summary of Notations

Symbol	Meaning
C	the set of all the clusters
E	the set of all the cores
$ctype(e)$	the cluster that core e belongs to
$\mathcal{F}_c$	a set of discrete frequency levels of cluster c
f_c^n	the n_{th} frequency level of cluster c
sf_c^n	the speedup of the n_{th} frequency level of cluster c
AP_e^m	the active power of frequency level m on core e
IP_e^m	the idle power of frequency level m on core e
T	a set of soft real-time the tasks
p_t	the period of task t
$\mathcal{V}_t$	a set of different versions of task t
v_t^n	the n_{th} version of task t
sv_t^n	the speedup of the n_{th} version of task t
q_t^n	the relative QoS of the n_{th} version of task t
$w_{t,c}^{m,n}$	the execution time of the n^{th} version of task t on cluster c at the m^{th} frequency level
HP	the hyper period corresponding to the task set T
J	the set of all the jobs generated from tasks within HP
JC_c	the set of all the jobs running on cluster c
$\mathcal{J}_t$	the set of jobs generated from task t
$task(e,x)$	the task running on core e during time interval x
$freq(c,x)$	the frequency of cluster c during time interval x
$ver(j_t^k)$	the version selected for the k_{th} instance of task t
$length(x)$	the length of the time interval x
$P(x)$	the power of the entire chip for time interval x
WE	worst case energy corresponding to the schedule with best QoS and highest frequency
WQ_t	minimal expected QoS for task t

A. Off-line Schedule Generation

The mapping of the jobs to the cores, scheduling them along the time axis, choosing the version for each job, and finally frequency assignment for each cluster is a complex optimization problem. While it might be possible to employ search based heuristic algorithm such as genetic algorithm (GA) for this problem, from our experimental evaluation, the search using GA often returned solutions that are local optimal. Integer Linear Programming (ILP) formulation, on the other hand, was computationally infeasible with the large design space. Therefore, we design a customized algorithm that can return good quality solution within reasonable runtime.

The flow of the off-line strategy is illustrated in the right-top corner of Figure 2. The off-line strategy has four main stages: **r-EDF**, **RELAX**, **IMPROVE**, **ADAPT**. We will use a simple example to illustrate the algorithm throughout. The input for the example is illustrated in Table II. We assume two clusters c_1 and c_2 where the first cluster has two simple cores and the second cluster has one complex core. We have three tasks t_1, t_2, t_3 with different periods and different execution time on the two clusters. Each task in this example has two different versions. During the hyper-period, we need to execute two jobs for t_1, t_2 and one job for t_3. Each cluster has two different frequency levels and the speedups for different frequency levels are given in Table II. The TDP constraint in this example does not allow both clusters to run at the highest frequency levels; any other frequency assignment keeps the system below the TDP. For simplicity, we do not show the exact QoS values.

TABLE II
ASSUMPTION FOR THE SIMPLE EXAMPLE

T		$w_{t,c}$		sv_t^v		p_t
		c_1	c_2	v_t^1	v_t^2	
t_1	$j_{t_1}^1$	100	80	1	1.2	100
	$j_{t_1}^2$					
t_2	$j_{t_2}^1$	100	60	1	1.3	100
	$j_{t_2}^2$					
t_3	$j_{t_3}^1$	200	160	1	1.5	200

C		sf_c^m	
		f_c^l	f_c^h
c_1	e_1	1	1.5
	e_2		
c_2	e_3	1	2

r-EDF Given a set of tasks, we first schedule the tasks using the r-EDF [13] policy. For each task, we choose the version with the shortest WCET and set all the clusters to run at the highest frequency f_c^h. This guarantees that if the performance demands cannot be satisfied with this schedule, then the performance demands cannot be satisfied with any other choice of task versions and/or frequency assignment. The original r-EDF algorithm is designed for symmetric multi-core architecture where a global scheduler assigns each newly arrived job to any core with enough available capacity. Then the corresponding local scheduler uses uni-processor EDF policy to meet the deadline constraints. Our strategy uses the same mechanism. In symmetric multi-core architecture, the capacity required by a task on each core is identical. However, for asymmetric multi-core architecture, we need to adjust the capacity required by a task on different clusters. The utilization of a task t on cluster c can be defined as $\frac{w_{t,c}}{sv_t^{V_t} \times sf_c^{F_c} \times p_t}$, i.e., we adjust the execution time for the highest frequency level on cluster c for the task version with the least WCET. For example, the execution time of $j_{t_2}^1$ running on cluster c_1 is calculated as $\frac{w_{t_2,c_1}}{(sv_{t_2}^2 \times sf_{c_1}^h)} = \frac{100}{(1.3 \times 1.5)}$. Figure 3 shows the schedule after applying the modified r-EDF policy on our example.

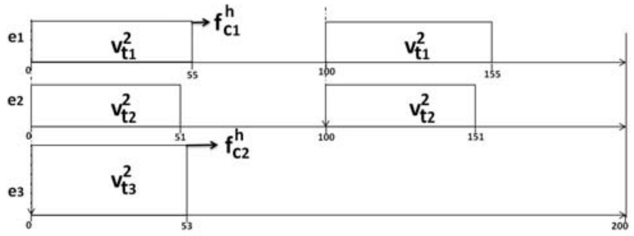

Fig. 3. Schedule after r-EDF: Every job selects the highest speedup (most approximate) version and all the cores run with the highest frequency. The power consumption from 0 to 51 exceeds the TDP constraint.

RELAX Once the schedule is generated by modified r-EDF policy, we proceed to ensure that the TDP constraint is satisfied in each time interval along the hyper-period. If the TDP constraint is violated in a time interval x, we reduce the frequency of the cluster with the least workload in that time interval, say cluster c, till the TDP constraint is satisfied. However, this leads to increased execution time of the tasks on cluster c. In our example in Figure 3, the TDP constraint is not satisfied during time 0 to 51; so we proceed to reduce the frequency of cluster c_2 containing core e_3. This leads to increased execution time of job $j_{t_3}^1$.

Algorithm 1 is used to insert a new time $slot$ after x to accommodate the extra execution time of the tasks on cluster c. The additional time for job j_t^k is denoted as $left(j_t^k)$. If the time interval after x is free to accommodate the extended execution time of the tasks on each core of cluster c, then we can simply insert the slot and move on to the next time interval that violates the TDP constraint (lines 8 - 16). The extra time in the new slot is adjusted based on the frequency deployed (line 12 and line 15). However, if the time interval after x is occupied on an affected core e by another task t', then t' may or may not be preempted by the extended execution time of task t depending on their relative deadline according to the EDF policy (line 18). Thus it is possible that either t or t' may end up missing the deadline in the worst case. In some cases $left(j_t^k)$ might be a negative value as in the **ADAPT** stage where the frequency may be increased to improve QoS/energy. Figure 4 shows the schedule after this stage for the simple example. Cluster c_2 decreases the frequency from $f_{c_2}^h$ to $f_{c_2}^l$ and the corresponding expanded execution time ranging from 53 to 104 is inserted by the $insertSlot$ function.

Algorithm 1: insertSlot($j_t^k, x, slot, f_c^m$)

```
1  $e \leftarrow$ core $e$ that $j_t^k$ is running on;
2  $left(j_t^k) \leftarrow left(j_t^k) + slot$;
3  for $left(j_t^k) > 0$ do
4      if $x > k \times p_t$ then
5          return false;
6      $x' \leftarrow x + 1$;
7      $t' \leftarrow task(e, x')$;
8      if $t' = \perp$ then
9          $task(e, x') \leftarrow t$;
10         if $freq(c, x') == 0$ then
11             $freq(c, x') \leftarrow f_c^m$;
12             $left(j_t^k) \leftarrow left(j_t^k) - length(x') \times sf_c^m$;
13         else
14             $f_c^{m'} \leftarrow freq(c, x')$;
15             $left(j_t^k) \leftarrow left(j_t^k) - length(x') \times sf_c^{m'}$;
16     else
17         $k' \leftarrow x \mod p_{t'}$;
18         if $j_{t'}^{k'} \neq j_t^k$ && $k' \times p_{t'} > k \times p_t$ then
19             $x' \leftarrow x + 1$;
20             $task(e, x') \leftarrow t$;
21             $f_c^{m'} \leftarrow freq(c, x')$;
22             $left(j_t^k) \leftarrow left(j_t^k) - length(x') \times sf_c^{m'}$;
23             if !insertSlot($j_{t'}^{k'}, x', sf_c^{m'}, f_c^{m'}$) then
24                 return false;
25     $x \leftarrow x + 1$;
26 if $left(j_t^k) < 0$ then
27     for $left(j_t^k) + length(x) \times sf_c^m \leq 0$ do
28         $freq(c, x) \leftarrow 0$;
29         $task(e, x) \leftarrow \perp$;
30         $left(j_t^k) \leftarrow left(j_t^k) + length(x) \times sf_c^m$;
31         $x \leftarrow x - 1$;
32 return true;
```

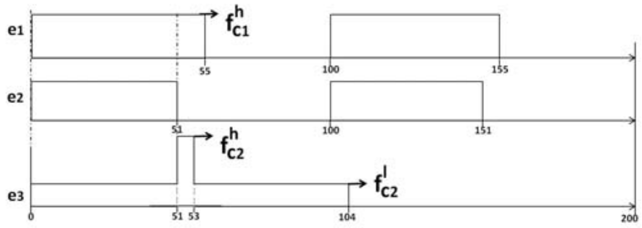

Fig. 4. Schedule after RELAX: Frequency on core $e3$ of cluster $c2$ from time 0 to 51 is decreased and the extra time slot is added for the affected job $j_{t_3}^1$.

IMPROVE So far in the schedule, we have used the task versions with the least WCET that corresponds to the worst QoS. In this stage, we attempt to choose better versions for each job as long as performance and TDP constraints are not sacrificed. Given the job assignment to the cores, we create a versions metric $vm(j, v)$ to guide in selecting the appropriate version for each job j. Let $j = j_t^k$, $v = v_t^n$, $f = f_c^m$ and let c and e be the cluster and the core where job j has been mapped to.

$$\gamma(j, v, f) = \frac{w_{t,c} \times (AP_e^m - IP_e^m)}{sv_t^n \times sf_c^m} + IP_e^m \times p_t \quad (5)$$

$$\delta(j, v, f) = \left(\frac{q_t^n - WQ_t}{1 - WQ_t}\right) \div \left(\frac{\gamma(j, v, f)}{\gamma(j, v_t^1, f_c^{F_c})}\right) \quad (6)$$

$$vm(j, v) = \frac{\sum_{f \in \mathcal{F}_c} \delta(j, v, f)}{F_c} \quad (7)$$

$\gamma(j, v, f)$ and $\delta(j, v, f)$ reflect the energy and normalized QoS/energy respectively for each a version of the task at a particular frequency level. $vm(j, v)$ averages $\delta(j, v, f)$ across all the available frequency levels. If $vm(j, v) < vm(j, v')$, then running the job with version v' is likely to achieve higher objective function compared to the version v. Therefore, we can select version v' unless TDP or performance constraints are violated. To select versions for the tasks, we first order all the job and version pairs in decreasing order of $vm(j, v)$ value. Let $\{j, v\}$ be the first job, version pair. If the version for job j in the schedule can be replaced by version v without violating TDP or performance constraint, then we change the version and remove all instances of job j from the ordered list. Changing the version may increase the execution time and hence we need to use the $insertSlot$ function to insert the new time slot with the additional execution time from the new version. As before, $insertSlot$ function checks for violation of TDP and performance constraints. The schedule table after this stage is shown in Figure 5. We can see that the QoS/energy of all the jobs are improved by employing better versions with higher vm value except job j_2. The reason is that choosing a better version for j_2 will violate the TDP constraint from time 51 to 53.

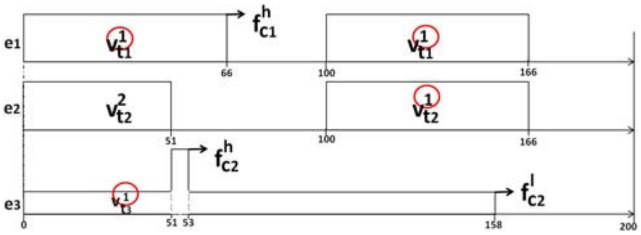

Fig. 5. Schedule after IMPROVE: QoS/energy is improved by changing the approximation versions for jobs.

ADAPT The final step in the offline schedule is to determine the appropriate frequency level for each cluster so as to improve the energy behavior without adversely affecting QoS, performance, and TDP constraints. For this purpose, we define the frequency metric $fm(f, c)$. The frequency metric for a cluster c computes the desirability of each frequency level f given the jobs assigned to the cluster c. Let JC_c be the set of jobs assigned to cluster c and let $j = j_t^k$, i.e., j belongs to task t. Then

$$fm(f, c) = \frac{\sum_{j \in JC_c} \delta(j, v, f)}{|JC_c|} \quad (8)$$

If $fm(f, c) > fm(f', c)$, then choosing frequency level f potentially achieves higher value of the objective function as long as it does not affect performance or TDP constraint. Similar to **IMPROVE**, we order the frequency, cluster pairs according to decreasing value of $fm(f, c)$. We process these pairs in order. Let $\{f, c\}$ be the first frequency, cluster pair. We check for every time interval x along the hyper-period if the frequency on cluster c can be changed to frequency level f without missing deadline or TDP constraint. This is checked using the $insertSlot$ function again. For the time intervals where this frequency change can be achieved, the frequency is changed and new time intervals are inserted to accommodation the additional execution time due to reduced frequency level. Figure 6 shows the schedule after ADAPT for our example.

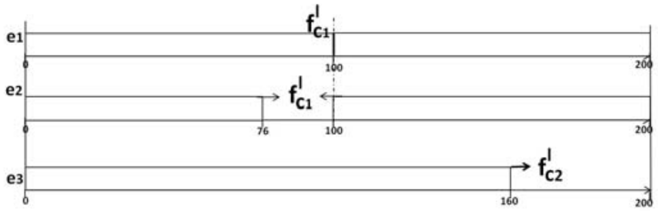

Fig. 6. Schedule after ADAPT: QoS/energy is improved by adjusting the frequency and the corresponding extra time is added for the affected jobs.

B. On-line Scheduler

The actual execution time of a job can be shorter than the observed WCET. We propose an on-line scheduler that dynamically shuts down idle clusters to prevent extra energy consumption and changes the versions for the future jobs to improve the objective function further. Typically, when a job completes execution, if the corresponding cluster is completely idle, then we power down the cluster till the next job arrives on the cluster according to the offline schedule. In addition, a predictive policy is used here to improve the versions of the incoming jobs. To be specific, we average the execution time of the recently executed jobs for the incoming task and predict the execution time of the incoming job. Then based on this predicted execution time, we select an improved version of the task if that can be accommodated within the next deadline. Experimental evaluation shows that the simple prediction policy works well and there are hardly any deadline miss due to version changes based on the prediction.

V. Experimental Evaluation

We evaluate our framework using the Versatile Express Development Platform [5] that includes a prototype version of the ARM big.LITTLE chip containing 3 Cortex A7 cores and 2 Cortex A15 cores. The chip is equipped with power sensors to directly measure the power consumption at different frequency levels. The TDP constraint for this chip is observed as around 8W. We use single-threaded version of four applications with *simlarge* input from the PARSEC [3] benchmark suite for our experiments. The WCET of a task is computed as the execution time of one iteration of the compute-intensive kernel, for example, execution time per frame for a vide decoder. We compute the WCET for each task on both A7 and A15 core. We also compute the speedup at different frequency levels on a cluster averaged across all the benchmarks. We create multiple approximate versions of each application through loop perforation [7]. The QoS of a version of a task is defined using the accuracy metric introduced in [7]. Table III shows the QoS of each version normalized w.r.t. the original version and the corresponding speedup due to approximation.

TABLE III
QoS and Speedup for different application versions

	QoS	Speedup		QoS	Speedup
body-track	0.9963	2.79	**x264**	0.9990	1.84
	0.9869	3.94		0.9919	2.45
	0.9856	5.25		0.9898	4.1
	0.9847	5.42		0.9901	5.22
canneal	0.9629	1.98		0.9971	5.45
	0.9344	3.87	**stream-cluster**	0.9991	1.29
	0.9272	4.81		0.9990	1.35
	0.9067	10.8		0.9946	1.6
	0.8999	16.69		0.9945	1.65
	0.8907	55.61		0.9935	1.87

For testing the quality and scalability of our framework, we create eight test cases (see Table IV) based on the applications and different number of clusters. The task sets are created by combining different benchmarks with different period. Notice that the minimal QoS expected by the user is set as 0.96 for all the applications.

TABLE IV
Characteristics of Test Cases

	# Tasks	# Jobs	# Clusters	# Cores	TDP (watt)
SI1	1	1	1	2 LITTLE	3.2
SI2	2	7	1	2 LITTLE	3.2
SI3	3	20	2	3 LITTLE + 2 big	8
SI4	4	55	2	3 LITTLE + 2 big	8
SI5	12	112	2	3 LITTLE + 2 big	8
SI6	48	448	2	3 LITTLE + 2 big	8
SI7	108	1008	4	6 LITTLE + 4 big	16
SI8	216	2016	6	9 LITTLE + 6 big	24

We evaluate our framework in three aspects. First, we evaluate the energy savings and improved QoS obtained through our approximation-aware framework. Figure 7 demonstrates 28% to 84% energy savings and around 1% QoS loss across different task sets and platforms compared to the optimal schedule without approximation. The reduced execution time from approximation contributes significantly to the energy savings.

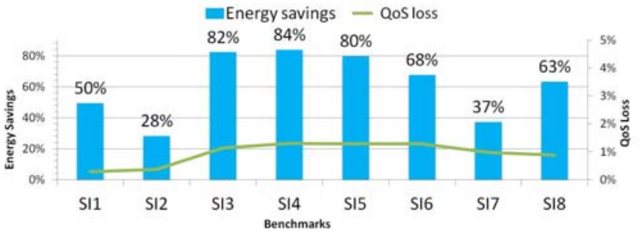

Fig. 7. Energy savings and QoS loss for test cases w.r.t. baseline.

Next we evaluate the quality of the solution returned by our strategy compared to the optimal solution obtained through an exhaustive design space exploration of all versions, task mapping, and frequency assignment. Figure 8 shows the objective function value of our solution as a percentage of the optimal solution. Clearly, our strategy returns close to the optimal solutions. In all cases, it can reach 93.9% of the optimal solution. Furthermore, our strategy is quite scalable with acceptable runtime even with large number of tasks and cores. Note that we regard the solution with the highest speedup version as the optimal one for the last four cases because the exhaustive design space exploration is too compute intensive to be feasible.

Figure 9 illustrates the additional improvement in the objective function and the deadline miss rate by employing on-line scheduler on top of the offline strategy. The online scheduler improves the QoS/energy ratio significantly for large task sets.

VI. Conclusion

In this paper, we present a scheduling framework for soft real-time systems on heterogeneous multi-cores. Our goal is

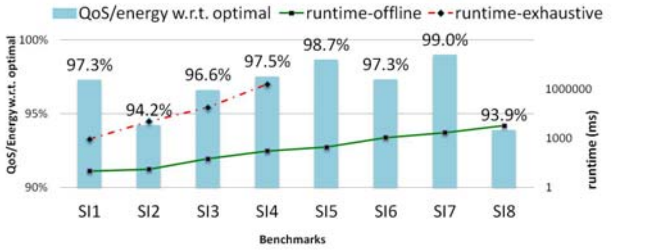

Fig. 8. QoS/energy and runtime for test cases w.r.t. optimal solution.

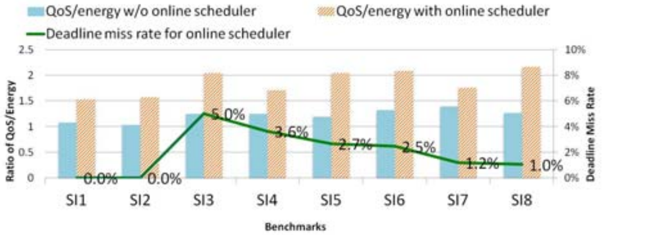

Fig. 9. QoS/energy and deadline miss rate improvement due to online scheduler.

to meet the performance and thermal design power (TDP) constraints while maximizing the energy efficiency. Towards that end, we employ approximate computing where we trade-off accuracy (QoS) with execution time/power in addition to traditional DVFS and core-level heterogeneity to create an enriched design space. We propose a scalable scheduling framework combining offline and online strategy that generates near-optimal solutions and achieves significant energy savings (up to 84%) with minimal impact on the QoS.

VII. ACKNOWLEDGEMENT

This work was partially supported by CSR and Singapore Ministry of Education Academic Research Fund Tier 2 MOE2012-T2-1-115.

References

[1] Muthukaruppan et al. Price Theory Based Power Management for Heterogeneous Multi-cores. In *ASPLOS'14.*

[2] Shafique et al. Dark Silicon as a Challenge for Hardware/Software Co-design: Invited Special Session Paper. In *CODES+ISSS'14.*

[3] Bienia et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *PACT'08.*

[4] Esmaeilzadeh et al. Dark Silicon and the End of Multicore Scaling. In *ISCA'11.*

[5] ARM Ltd. 2011. `http://www.arm.com/products/tools/development-boards/versatile-express/index.php.`

[6] Muthukaruppan et al. Hierarchical Power Management for Asymmetric Multi-core in Dark Silicon Era. In *DAC'13.*

[7] Sidiroglou-Douskos et al. Managing Performance vs. Accuracy Trade-offs with Loop Perforation. In *ECFSE'11.*

[8] Sondag et al. Phase-guided Thread-to-core Assignment for Improved Utilization of Performance-Asymmetric Multi-core Processors. In *ICSE Workshop on Multicore Software Engineering, 2009.*

[9] Cong et al. Energy-efficient Scheduling on Heterogeneous Multi-core Architectures. In *ISLPED'12.*

[10] Hanumaiah et al. Temperature-aware DVFS for Hard Real-time Applications on Multi-core Processors. In *TC'12.*

[11] Li et al. Temperature-aware Scheduling and Assignment for Hard Real-time Applications on MPSoCs. In *SC'07.*

[12] Chantem et al. Efficient Operating System Scheduling for Performance-Asymmetric Multi-core Architectures. In *VLSI'11.*

[13] Funk, Shelby Hyatt. EDF Scheduling on Heterogeneous Multiprocessors. In *University of North Carolina, 2004.*

[14] Davis et al. A Survey of Hard Real-time Scheduling for Multiprocessor Systems. In *ACM Computing Surveys (CSUR), 2011.*