Fairness-Aware Energy Efficient Scheduling on Heterogeneous Multi-Core Processors

Bagher Salami, Hamid Noori, and Mahmoud Naghibzadeh

Abstract—Heterogeneous multi-core processors (HMP) with the same instruction set architecture (ISA) integrate complex high performance big cores with power efficient small cores on the same chip. In comparison with homogeneous architectures, HMPs have been shown to significantly increase energy efficiency. However, current techniques to exploit the energy efficiency of HMPs do not consider fair usage of resources that leads to reduced performance predictability, a longer makespan, starvation, and QoS degradation. The effect of different cluster voltage and frequency levels on fairness is another issue neglected by previous task scheduling algorithms. The present study investigates both the fairness problem and energy efficiency in HMPs. This article proposes a heterogeneous fairness-aware energy efficient framework (HFEE) that employs DVFS to meet fairness constraints and provide energy efficient scheduling. The proposed framework is implemented and evaluated on a real heterogeneous multi-core processor. The experimental results indicate that the introduced technique can significantly improve energy efficiency and fairness when compared to Linux standard scheduler and two energy efficient and fairness-aware schedulers.

Index Terms—Energy efficient scheduling, fair scheduling, heterogeneous multi-core, big.LITTLE architecture

1 Introduction

The dark silicon phenomenon, process variation, and the failure of Dennard scaling pushed computer designers to develop heterogeneous (asymmetric) multi-core processors (HMP). HMPs can be divided into two categories: I) cores with the same instruction set architecture, such as ARM's big.LITTLE and Nvidia's Kal-El, and II) cores with different instruction set architectures, such as IBM Cell, Nvidia's Tegra, and AMD's Fusion.

ARM's big.LITTLE processors contain two distinct types of cores: high performance Cortex-A15 (big cluster) and low power Cortex-A7 (little cluster). Each cluster has a specific microarchitecture, voltage and frequency levels, cache size and pipeline stages. The execution time and energy consumption of any program is affected by: a) cluster type and b) the voltage and frequency level of each cluster. Therefore, exploiting these characteristics at the OS (Operating System) scheduler level is crucial. With the aim of optimizing both the overall makespan (the duration time of the start of programs to the end of the last program) and energy consumption, a variety of scheduling algorithms have been proposed for asymmetric multi-core processors [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]. To achieve this, the algorithms learn about application behaviors and map CPU intensive workloads to big cores, while assign memory intensive workloads to little cores. For workload distribution among different clusters, some techniques [16], [27] exploit ILP (instruction level parallelism) and MLP (memory level parallelism) instead of the CPU and memory intensity of tasks.

As a critical objective seriously affecting the performance and power consumption of running programs, fairness has been ignored by the previous energy efficient task scheduling algorithms. One scheduler is considered fair if all programs suffer from the same performance degradation normalized to the isolated run on a base configuration [30], [32]. Ignoring fairness in scheduling algorithms may cause undesirable behaviors in the system [29], such as reduction in performance predictability, a longer makespan, starvation, and hence QoS degradation. Although some proposed algorithms take into account fairness for heterogeneous multi-cores in the OS scheduler [29], [30], [31], [32], they do not consider power consumption and energy efficiency. To the best of the present work's knowledge, both fairness and energy efficiency of a task scheduler on HMPs have not yet been studied. The current paper introduces a scheduler that simultaneously addresses both fairness and energy efficiency.

The effect of different cluster voltage and frequency levels on energy efficiency and fairness is another overlooked matter in task scheduling algorithms. According to the current study's experimental results, the voltage and frequency ratio of big to little clusters significantly affects the fairness and energy efficiency of the scheduler.

Generally, the proposed algorithm aims to improve the scheduler's energy efficiency by assigning big's appropriate programs to the big cluster and little's appropriate programs to the little cluster. For each program, the ratio of the instruction per watt (IPW) of the big cluster to that of the little cluster serves as an indicator of a program's suitability for each cluster type. Experimental results indicate that programs with a

• *The authors are with the Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad 9177948974, Iran. E-mail: bagher.salami@mail.um.ac.ir, {hnoori, naghibzadeh}@um.ac.ir.*

Manuscript received 20 May 2019; revised 21 Nov. 2019; accepted 26 Mar. 2020. Date of publication 2 Apr. 2020; date of current version 11 Dec. 2020. (Corresponding author: H. Noori.) Recommended for acceptance by B. Parhami. Digital Object Identifier no. 10.1109/TC.2020.2984607

higher energy efficiency ratio (IPW_{big}/IPW_{little}) are more energy efficient to run on the big cluster and those with a lower ratio value are suitable for the little cluster. The present research does an exhaustive exploration on how voltage and frequency values affect fairness. Based on this study, a reactive algorithm is proposed to select the voltage and frequency of each cluster, so that the target fairness is achieved. For managing fairness among different programs, the voltage and frequency ratio of the big to little cluster is considered. Through proper task assignment to clusters and management of each cluster's voltage and frequency, a certain level of fairness, known as the fairness threshold, is guaranteed, while energy efficiency (energy delay product) is improved.

The current paper presents a scheduler that works effectively for heterogeneous big.LITTLE processors with DVFS support. This scheduler is designed to replace task mapping in Linux-like runtime systems and the ondemand DVFS governor. The present research could not find a scheduler with the same objectives as its own. Therefore, the proposed scheduler is evaluated on a real asymmetric multi-core processor with ARM big.LITTLE architecture and also compared with Standard Linux scheduler and two state-of-the-art competitors: 1) an energy efficient scheduler which does not consider fairness [28] and 2) a fairness-aware scheduler [31] which does not consider energy efficiency. The source code of the proposed framework and the implemented opponent algorithms are available online at https://github.com/baghers/HFEE. The results show that the proposed scheduler guarantees the fairness threshold while improving overall energy efficiency. In summary, the present paper makes the following contributions:

- Investigation of the effect of different cluster voltage and frequency levels on the fairness of running programs.
- Extending fairness definition for heterogeneous multi-core processors with DVFS capability.
- Introduction of a scheduler that simultaneously governs both fairness and energy efficiency for heterogeneous multi-core processors.
- Improving both fairness and energy efficiency on a real asymmetric multicore processor through applying the proposed algorithm compared to Linux scheduler and two contemporary schedulers (i.e., an energy efficient scheduler which does not consider fairness and a fairness-aware scheduler which does not consider energy efficiency).

2 Motivation and Related Work

Various methods for task scheduling on HMPs have been proposed that can be categorized into single program (programs per core $< = 1$) and multi-programs (programs per core > 1) from the program count perspective. Also, these algorithms can be classified into serial and parallel, based on application types, forming four categories that are depicted in Fig. 1. Single serial program schedulers [13], [15], [16] are used usually in program phase detection [13], studying and managing temperature, performance, and power behaviors of different clusters [15], and analyzing programs' attitude on asymmetric multi-cores [16]. On the other hand, single

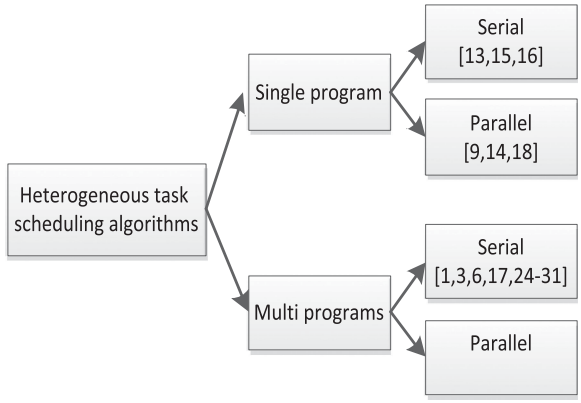

Fig. 1. A hierarchical classification of related work.

program schedulers for parallel applications [9], [14], [18] are utilized in load balancing in order to prevent bottleneck [9], asymmetric data partitioning [14], and critical section management in asymmetric environments [18].

Multi-program schedulers utilize two kinds of tasks; serial, and parallel, while to the best of authors' knowledge, no prior attempt has been made to implement an algorithm for scheduling multi parallel programs simultaneously. Multi-program serial schedulers [1], [3], [6], [17], [24], [25], [26], [27], [28], [29], [30], [31] for asymmetric multi-core platforms are employed to govern the trade-offs between two criteria: e.g., performance versus power [1], [3], [17], [25], [26], [27], performance versus fairness [29], [30], [31], or performance versus other criterion such as temperature [28], reliability [24] and aging rate [6].

Efficiently utilizing performance-power trade-offs need to assign tasks to the appropriate core types and adjust their frequencies to a suitable value through DVFS. SmartBalance [3] is one of the first attempts of closed-loop load balancing, consists of three phases of sensing, estimation, and prediction. Unlike the open-loop standard Linux load balancer, which distributes the threads uniformly, SmartBalance as a Feedback based controller tries to assign the threads to the matched core type to achieve the best energy efficiency with the cost of performance overhead. HPM [27] is a control-based framework to achieve the optimal performance-power trade-off, with the aid of multiple PID controllers (one for each application and one for each cluster), considering the TDP (Thermal Design Power) budget. Cluster controller allocates the power budget to each cluster and the other controllers try to meet the TDP budget. However, for higher number of clusters, its performance degrades dramatically. ApxSched [25] presents a static scheduler, considering various approximate versions of tasks to maximize performance with respect to power constraints. Different versions of each task are produced based on loop perforation [25] technique and scheduling decisions are made according to an off-line heuristic. Myungsun *et al.* [26] propose a utilization-aware load balancer for big.LITTLE processors. A processor utilization estimator is presented to determine the most appropriate frequency for a given set of tasks, considering performance constraints. But, utilization-based criterion is not adequate for power management of asymmetric multi-cores. Paragon [24] is a resource allocation approach for unknown incoming workloads. Paragon uses classification

TABLE 1
Specification of Related Work Schedulers

Scheduler	Real platform	DVFS	Performance	Energy	Fairness
HPM [27]	✓	✓	✓	✓	
Utilization-Aware [26]	✓		✓		
SmartBalance [3]		✓	✓	✓	
ApxSched [25]	✓	✓	✓	✓	
Paragon [24]	✓		✓		
Algorithmic Opt. [28]	✓	✓	✓	✓	
Perf&Fair [30]	✓		✓		✓
Contention-Aware [29]	✓		✓		✓
Min-Fair [31]			✓		✓
HFEE	✓	✓	✓	✓	✓

techniques to estimate the impact of heterogeneity and interference on performance uses this information for workload assignment to different server types. The target server for each workload provides the best performance and has less interfere with other collocated workloads. Workload classification is based on sampling, that has a huge overhead and it may not be applicable to asymmetric multi-cores. DTPM [28] is one of the latest studies of dynamic power and frequency management. Bhat *et al.* [28] propose a power budget predictor to estimate the power budget based on current temperature and temperature threshold. The other presented predictor, predicts the power consumption based on the next frequency setting using power sensors. Then, DTPM specifies the maximum feasible frequency under the available power budget. A leakage power model of the ARM big.LITTLE architecture is used in their proposed technique.

On the other hand, performance-fairness trade-off management is the other problem of task scheduling. Several definitions of fairness have been proposed in the literature. Frequently, a system is considered fair when all the running programs suffer the same slowdown corresponding to their isolated execution [29]. On asymmetric multi-cores, the slowdown depends on two main factors [29]: (1) performance asymmetry and (2) shared-resource contention. Feliu *et al.* [30] present a process scheduler for SMT multicores that estimates the progress experienced by the processes, and gives priority to the processes with lower accumulated progress. This algorithm requires extension for asymmetric multi-cores. One of the first researches of considering shared-resource contention in task scheduler, which is the second source of slowdown, has been presented in [29]. In [31] some different fair schedulers are presented to efficiently distribute big-core cycles among different applications. They ask for the target fairness from the user as an input and try to meet the target fairness, while maximizing performance. Table 1 summarizes the related schedulers in terms of their specification.

An issue in task scheduling that has not yet been addressed by previous works is the simultaneous consideration of both fairness and energy efficiency for task scheduling on the asymmetric multi-core processors. This overlooked problem is the motivation behind the current work. The present study also explores the cluster's frequency and fairness relationship. The results of this study are considered in developing the proposed scheduler. Additionally, the proposed HFEE scheduler supports DVFS and is the first to include DVFS in a fairness-aware scheduler. When compared to Linux Standard Scheduler and two state-of-the-art works (an energy efficient [28] and a fair scheduler [31]), HFEE improves both energy efficiency and fairness.

3 System Model, Metrics and Problem Statement

This section discusses the workload and platform models as well as energy efficiency and fairness metrics are presented.

Workload Model. We consider a set of m single thread programs as $P = \{p_1, p_2, \ldots, p_m\}$ that can be more than total core count $(m \geq \text{Core}\#)$. For uniformity, in this paper the term *task* is used interchangeably for both programs and tasks. We assume that total instructions and average power consumption of every task on each core type is known. Task scheduling is done at fixed periods called epochs, which is denoted by τ.

Platform Model. The system considered in this paper is an HMP platform consists of multiple cache-coherent cores as $C = \{c_1, c_2, \ldots, c_n\}$, that share the same ISA and memory address space. Cores are organized into the set of clusters as $Z = \{z_1, z_2, \ldots, z_s\}$, while all cores in the same cluster support the same voltage/frequency pairs; therefore DVFS is being applied at the cluster level. It is assumed that each core provides *hardware performance counters* and each cluster has a *power sensor*, which allows to characterize programs power consumption.

Energy Efficiency Metrics. Energy efficiency is defined as the combination of reduced energy consumption and performance improvement (runtime) [33]. The energy-delay product (EDP) is considered as a long-term metric and calculated by the product of the total amount of energy consumed and the runtime duration. The higher the energy efficiency, the less EDP value. As a short-term criterion, the instruction per watt is another energy efficiency metric, which is the total amount of committed instruction for every watt of power consumed. Clearly, the more IPW, the higher energy efficiency.

Fairness Metric. According to our studies, there is not a single and unique definition of fairness. One of the most prevalent definition of fairness is expressed as: *An scheduler is considered fair if the variation of performance degradation normalized to isolated run is minimal* [30], [32], Where, Van Craeynest [32] considers fast cores, while Feliu [30] considers equal usage of both big and little cores as isolated run. Dynamic frequency scaling has not been studied in the previous works and we need to consider frequency in the fairness definition.

The fairness definition in [30], [32] has been extended to support DVFS as: *A scheduler is considered fair if the variation of performance degradation normalized to isolated run on big core with highest voltage and frequency is minimal.*

The slowdown of each program under a scheduler is expressed as: $S\_max_i = \frac{T_{sch.,i}}{T_{\max freq.,i}}$, where $T_{sch.,i}$ is the execution time of program i under the scheduler and $T_{\max freq.,i}$ is the execution time of program i on the big core with maximum voltage and frequency which enables the evaluation of fairness in terms of $uniformity_{max}$:

$$Uniformity_{max} = 1 - \frac{\sigma_{S\_max}}{\mu_{S\_max}}, \tag{1}$$

Where σ_{S_max} and μ_{S_max} are the standard deviation and the average of S_max values of all programs respectively.

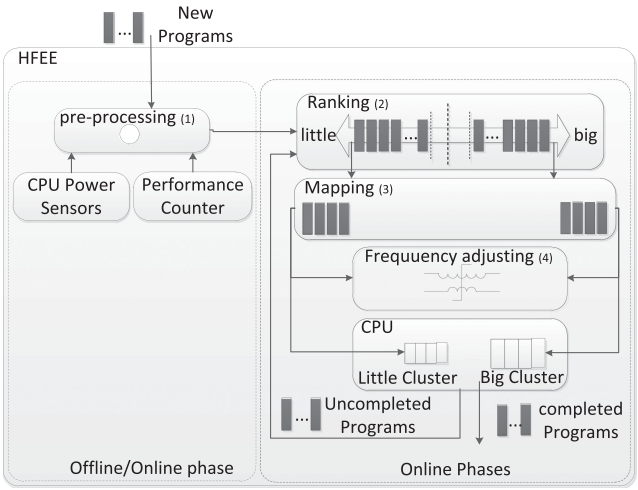

Fig. 2. Heterogeneous fairness-aware energy efficient framework.

Problem statement. We study the problems of assigning m single thread programs to one n cores big.LITTLE processor and determining the voltage and frequency of each cluster (voltage and frequency of each cluster can be adjusted locally) dynamically such that the system fairness is less than a user defined threshold and energy efficiency is maximum.

4 HFEE Framework

As shown in Fig. 2, the HFEE framework for HMPs is composed of four parts: 1) *Pre-processing* exploits CPU power sensors and performance counters to identify the energy efficiency ratio of each program; 2) *Ranking* specifies a program's suitability score for the two big and little clusters; 3) *Mapping* maximizes energy efficiency through choosing appropriate programs for each cluster after the programs have been ranked; 4) *Frequency scaling* guaranties fairness threshold through proper frequency selection. The following subsections fully describe the different parts of the HFEE framework.

4.1 Pre-Processing

As mentioned, energy efficiency improvement is achieved by assigning tasks to the appropriate core types [1]. The ratio of instruction per watt on the big cluster to instruction per watt on the little cluster of a program can be an indicator of its suitability for each cluster type. The energy efficiency ratio (EER) is the name given to this ratio by the current study:

$$EER\ (i) = \frac{\frac{inst.(i)_{big}}{watt(i)_{big}}}{\frac{inst.(i)_{little}}{watt(i)_{little}}} = \frac{inst.(i)_{big} * watt(i)_{little}}{inst.(i)_{little} * watt(i)_{big}}, \tag{3}$$

where $inst.(i)_{big}$, $inst.(i)_{little}$ are the instructions per second (IPS) of program i on big and little cores, respectively, and can be extracted using the CPU performance counters. The average power consumption of program i on big and little cores are denoted by $watt(i)_{big}$ and $watt(i)_{little}$, which are obtained from the CPU power sensors. When there are 2N programs, N little cores and N big cores, to reach higher energy efficiency, N programs with lower EER values are more suitable to run on little cores and N programs with higher EER values are more suitable to run on big cores. The fully investigation of EER values of SPEC CPU2006 benchmark are presented in the Section 5.3.

As it is shown in Fig. 2, this phase can be performed online or offline based on hardware capabilities. Since our evaluated board provides power sensor per cluster, we are not able to do pre-processing phase online and it is carried out offline. For offline preprocessing, all applications are run on both types of cores and the average of power consumption and retired instructions of each application from start to end is used to calculate EER. However, to have online pre-processing phase, the evaluation board has to be equipped with the power sensor per core. To have online preprocessing, one solution can be as following. For the first two epochs, each program is run on big and little cores, then IPW of big and little cores are known. These IPWs are updated on next epochs.

4.2 Ranking

After determining EER values during the pre-processing phase, program ranking phase then decides the assignment of programs to different cores. For this purpose, first $EER(i)$ is normalized (i.e., it is limited between 0 and 1):

$$EER_N(i) = \frac{EER(i) - \min_{j\in P}(EER(j))}{\max_{j\in P}(EER(j)) - \min_{j\in P}(EER(j))}. \tag{4}$$

Programs with $EER_N(i)$ values closer to one are more appropriate for big cores, while programs with $EER_N(i)$ values closer to 0 are more suitable to run on little cores. If programs are sorted solely by $EER_N(i)$, then some programs with $EER_N(i)$ values close to 0.5 confront starvation. To prevent starvation, the wait time of each program is considered along with $EER_N(i)$, so that different programs are sorted according to $Score(i)$ as:

$$Score(i) = EER_N\ (i) + \left(\frac{EER_N(i)}{0.5} - 1\right).K(i). \tag{5}$$

where $K(i)$ denotes the number of epochs waiting for a CPU and $score(i)$ is the score of program i. If $EER_N(i)$ is greater than 0.5, then $(\frac{EER_N(i)}{0.5} - 1)$ will be from 0 to 1. If $EER_N(i)$ is less than 0.5, then it will be from -1 to 0. With the use of Eq. (5), programs with $EER_N(i) = 0.5$ still confront starvation and programs with $EER_N(i)$ near to 0.5 must wait a long time for a CPU. Therefore, it is necessary to ensure that all programs receive CPU time after at most K epochs. For this purpose, the $EER_N(i)$ value of programs waiting for more than K epochs is corrected. Fig. 3 shows the program ranking for big and little cores. α_1 and α_2 are the suggested parameters where α_1 is the minimum $EER_N(i)$ value for a program to receive a CPU (big cores) after K epochs (via Eq. (5)):

$$\begin{aligned} 1 = \propto_1\ + \left(\frac{\propto_1}{0.5} - 1\right).K \Rightarrow 1 = \propto_1\ + (2\propto_1 - 1).K \\ \Rightarrow \propto_1 + 2K\propto_1 - K = 1 \Rightarrow\ \propto_1 = \frac{K+1}{2K+1}. \end{aligned} \tag{6}$$

Similarly, α_2 is the maximum $EER_N(i)$ value for a program to obtain a CPU (little cores) after K epochs (via Eq. (5)):

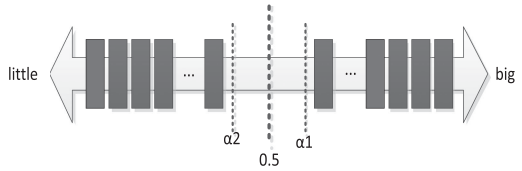

Fig. 3. Program ranking for big and little cores.

$$0 = \propto_2 + \left(\frac{\propto_2}{0.5} - 1\right).K \Rightarrow 0 = \propto_2 + (2\propto_2 - 1).K$$
$$\Rightarrow \propto_2 + 2K\propto_2 - K = 0 \Rightarrow \propto_2 = \frac{K}{2K+1}. \quad (7)$$

In the proposed framework, if $EER_N(i) \in (0.5.\alpha_1)$. then the $EER_C(i)$ value is considered as α_1; if $EER_N(i) \in (\alpha_2.0.5)$. then the $EER_C(i)$ value is presumed to be α_2:

$$EER_C(i)\begin{cases} \text{Max}(EER_N(i), 0.5+\propto_1) & EER_N(i) > 0.5 \\ \text{Min}(EER_N(i), 0.5-\propto_2) & EER_N(i) < 0.5 \\ Rand(0.5+\propto_1, 0.5-\propto_2) & EER_N(i) = 0.5 \end{cases}. \quad (8)$$

Finally, $score(i)$ is obtained as follows:

$$Score(i) = EER_C(i) + \left(\frac{EER_C(i)}{0.5} - 1\right).K(i). \quad (9)$$

With Eq. 9, the wait time of each program will be less than K epochs. The highest wait time of different programs is:

$$Max\_wait = K + \frac{M}{C}.$$

Where M and C are program and core counts, respectively. At the end of this phase, all programs are sorted based on their scores via Eq. (9).

4.3 Mapping

This phase maps appropriate programs to different clusters. High score programs are more suitable for big cores, while low score programs are more appropriate for little cores. As demonstrated in Fig. 3, selected programs from the rightmost side of the sorted list are assigned to the big cores and programs from the leftmost side are mapped to the little cores.

4.4 Frequency Scaling

As mentioned, the selected voltage and frequency of each cluster impacts the fairness in executing programs on HMP at runtime. However, previous task scheduling algorithms have neglected the effect of different cluster voltage and frequency levels on the fairness of running programs. If the applications take the equal processing resources, the fairness would be high. But there are different processing resources in the heterogeneous processors (different core types and frequency levels), thus applications suffer more unfairness compared to homogeneous processors. The more difference of core's computing power, the less fairness amount. To investigate how different cluster frequencies may alter the fairness of the scheduler, the present study investigates how both big and little voltage and frequency values impact fairness. To this end, various set of workloads are selected and executed over combinations of big and little frequency levels and the fairness value of each combination is calculated in terms of uniformity, which is fully investigated in the Section 5.3. The experimental results indicate when the computing power of two clusters are the same (the big to little cluster speedup = 1), the higher fairness is achieved.

CPU frequency is a source of diversity among the two clusters. For example, when the little cluster's frequency is constant at 1400 MHz and big cluster's frequency is 2000 MHz, the big computing power is much more than little. When the big cluster's frequency decreases from 2000 MHz, the difference of two core's computing power decreases at first (rising uniformity) until two cluster's computing power become rather equal (the big to little cluster speedup = 1). This frequency is application dependent and called $freq_{eq}$. By scaling down big cluster's frequency lower than $freq_{eq}$, while little cluster's frequency is fixed 1400 MHz, the difference of the two cluster's computing power increases (little cluster computing power would be more than big), causes lower uniformity. So, at the state of decreasing big cluster's frequency from maximum frequency value, uniformity at first rises until $freq_{eq}$ is reached, but for more scaling down under $freq_{eq}$, then uniformity falls substantially. Thus, it is vital to stop scaling down big cluster's frequency when uniformity starts to decline.

According to our experiments (Section 5.3), another observation is that the highest values of uniformity for all workloads happen when the big cluster's frequency is lower than 1400 MHz. The scaling up of big cluster's frequency higher than 1400 MHz, always causes fairness corruption for all workloads. So, the improvement of uniformity is never achieved by scaling up big cluster's frequency value more than 1400 MHz in our workloads. This frequency value is defined as $f_{threshold}$ by the present study. Generally, the procedure of $f_{threshold}$ calculation consists of two steps: 1) For each workload, the value of $freq_{eq}$ is measured. 2) After $freq_{eq}$ are identified for all workload, $\max(freq_{eq})$ is considered as $f_{threshold}$.

Motivated by these observations and with the intent of controlling system fairness and guaranteeing a user-defined level of fairness known as $Uniformity_{threshold}$ (demonstrated in Fig. 4), the present research proposes a reactive frequency adjusting technique based on a state transition. In the proposed approach the little cluster always operates at its maximum frequency similar to [28], [31]. Little cluster power consumption is always low, so it is not necessary to exploit DVFS for power management. In the proposed state transition, there are only two fairness states, namely {low, high} or {L, H} for short. When the current system's uniformity is under $Uniformity_{threshold}$, then the system is in the low fairness state. In contrast, the high fairness state occurs when the current system's uniformity is higher or equal to $Uniformity_{threshold}$. Also, we assume there are two frequency states: {L, H} or low (when the big cluster's frequency is under $f_{threshold}$) and high (when the big cluster's frequency is higher or equal to $f_{threshold}$) respectively. Thus, the processor at each scheduling epoch can be in one of four states represented by the notation of (uniformity, frequency) and enumerated as: {(L, L), (L, H), (H, L), (H, H)}.

The present study's target (optimal) state is (H, L) when system uniformity is high and the frequency of the big cluster is lower than $f_{threshold}$. However, as mentioned earlier,

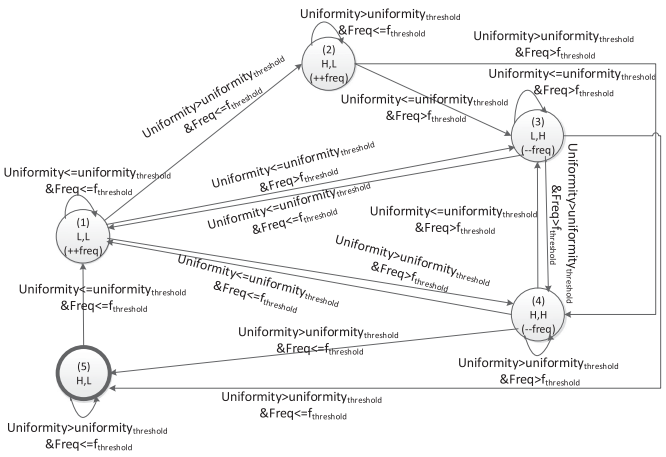

Fig. 4. State transition diagram of DVFS adjusting.

the (H, L) state can be reached through either incrementing or decrementing the big cluster's frequency. Therefore, as seen in Fig. 4, there are two (H, L) states: state numbers 2 and 5. The distinction of these two states is in their previous states. Target (optimal) state number 5 is reached after a frequency down scaling in state numbers 3 and 4, while state number 2 is achieved following a frequency up scaling in state number 1. When the system reaches state number 5, uniformity and frequency are in appropriate conditions. In other words, the current system's uniformity is higher or equal to $Uniformity_{threshold}$, the big cluster's frequency is under $f_{threshold}$, and frequency in this state remains fixed.

4.5 Complexity Analysis

Given the number of cores $|C|$, and programs $|P|$, proposed scheduler at each scheduling epoch has the complexity of $|P| \times \log(|P|)$ for Ranking, $|C|$ in Mapping phase, and $|1|$ for frequency scaling, while $|P|$ in Pre-processing phase. If we assume $|P| >= |C|$, then the runtime is bounded by $O(|P| \times \log(|P|))$ defined by the Ranking phase.

5 Experimental Evaluation

This section presents the experimental results for different applications on a real platform and provides analysis of the obtained results.

5.1 Experimental Setup

The proposed fairness-aware energy efficient scheduling framework is evaluated by a real HMP processor with the ARM big. LITTLE architecture. The evaluation platform is an Odroid-XU3 board featuring the Exynos5422 SoC with four Cortex-A15 3-way out-of-order (big) cores and four Cortex-A7 2-way in-order (little) cores on a chip. The range of the big core frequencies is from 200 MHz to 2 GHz and from 200 MHz to 1.4 GHz for the little cores. Four big cores share a 2MB L2 cache and four little cores share a 512 KB L2 cache.

The device has only 2 GB DRAM which is insufficient to run eight benchmarks on all of the eight cores, which has also been mentioned in [31]. Thus, similar to [31], only two big cores and two little cores are used, and the remaining are turned off. Ubuntu-mate 16.04.3 is installed with kernel version of 4.14 on it and *Perf* library is employed as one of the two most common performance counter profiling tools on Linux. *cpufreq* is used to adjust the processor frequency and power consumption is extracted from the embedded power sensors of each cluster.

TABLE 2
Speccpu2006 Benchmark Categorization Based on IPS

Benchmark	Class	Benchmark	Class
998.specrand	L	416.gamess	H
999.specrand	L	401.bzip2	H
429.mcf	L	454.calculix	H
400.perlbench	L	483.xalancbmk	H
471.omnetpp	L	465.tonto	H
473.astar	L	434.zeusmp	H
403.gcc	M	435.gromacs	H
445.gobmk	M	410.bwaves	H
450.soplex	M	437.leslie3d	H
459.GemsFDTD	M	444.namd	H
458.sjeng	M	470.lbm	H
453.povray	M	456.hmmer	H
436.cactusADM	M	462.libquantum	H
433.milc	M	464.h264ref	H

5.2 Workloads

In the present study's experimental evaluation, the workloads consist of SPEC CPU2006 mixes, which are characterized in application throughput terms as instructions per second (IPS). Fig. 5 presents the IPS values of different applications on the big cluster, where IPS values spread over a range from 0.22×10^9 to 2×10^9. Application workloads are categorized based on their IPS values as low (IPS $< 1 \times 10^9$), medium ($1 \times 10^9 <$ IPS $< 1.5 \times 10^9$), and high (IPS $> 1.5 \times 10^9$) and denoted by L, M, and H respectively as depicted in Table 2. As demonstrated in Table 3, fifteen different subsets of SPEC CPU2006 benchmarks are selected for evaluations of the schedulers.

5.3 Application Characterization

In this section, the EER value and DVFS impact on fairness are fully investigated.

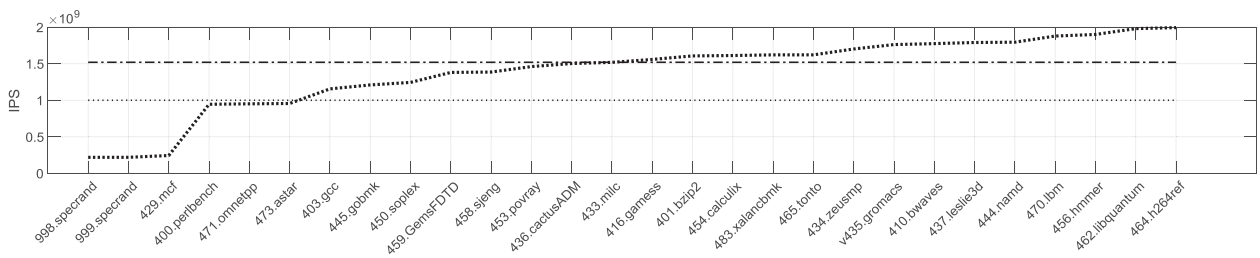

Fig. 5. Application characterization in terms of IPS.

TABLE 3
Multi-Application Workload Combinations

Name	Class	Benchmarks
W1	LLLLLL	471.omnetpp + 998.specrand + 429.mcf + 400.perlbench + 999.specrand + 473.astar
W2	MMMMMM	445.gobmk + 458.sjeng + 459.GemsFDTD + 453.povray + 433.milc + 436.cactusADM
W3	HHHHHH	401.bzip2 + 416.gamess + 454.calculix + 483.xalancbmk + 465.tonto + 434.zeusmp
W4	HHMMLL	462.libquantum + 464.h264ref + 403.gcc + 450.soplex + 429.mcf + 400.perlbench
W5	HHHHML	437.leslie3d + 434.zeusmp + 470.lbm + 456.hmmer + 453.povray + 400.perlbench
W6	HHHHHL	435.gromacs + 410.bwaves + 437.leslie3d + 434.zeusmp + 470.lbm + 400.perlbench
W7	HHHMMM	456.hmmer + 462.libquantum + 464.h264ref + 453.povray + 433.milc + 436.cactusADM
W8	HMMMML	483.xalancbmk + 450.soplex + 459.GemsFDTD + 453.povray + 433.milc + 400.perlbench
W9	MMMLLL	459.GemsFDTD + 453.povray + 433.milc + 471.omnetpp + 429.mcf + 400.perlbench
W10	MMMMLL	436.cactusADM + 459.GemsFDTD + 453.povray + 433.milc + 400.perlbench + 471.omnetpp
W11	MMMMML	450.soplex + 459.GemsFDTD + 453.povray + 433.milc + 436.cactusADM + 471.omnetpp
W12	MMLLLL	403.gcc + 450.soplex + 471.omnetpp + 473.astar + 429.mcf + 400.perlbench
W13	MHLLLL	445.gobmk + 410.bwaves + 473.astar + 429.mcf + 400.perlbench + 471.omnetpp
W14	HHHLLL	437.leslie3d + 444.namd + 470.lbm + 400.perlbench + 471.omnetpp + 429.mcf
W15	HHLLLL	435.gromacs + 410.bwaves + 471.omnetpp + 473.astar + 429.mcf + 400.perlbench

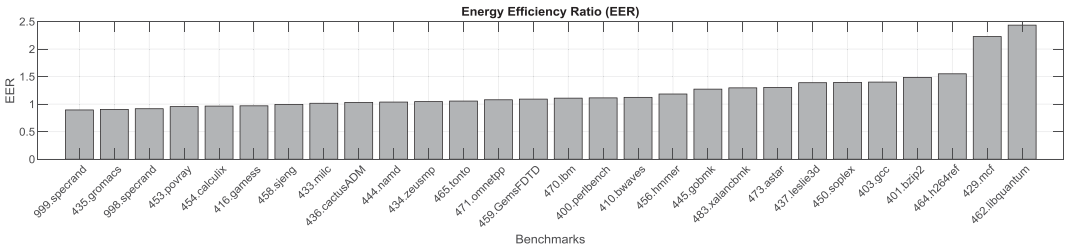

Fig. 6. Energy Efficiency Ratio (EER) of SPEC CPU2006 benchmarks.

5.3.1 EER Value

SPEC CPU2006 benchmark is employed in our experiments. Fig. 6 provides the EER values for all applications using Eq. (3). The EER values are spread over a range from 0.89 to 2.44. AS we mentioned before, programs with lower EER values are more energy efficient to run on little cores, while programs with higher EER values are more energy efficient for the big cores.

5.3.2 DVFS Impact on Fairness

Fairness is harder to achieve for heterogeneous multi-core processors compared to homogenous, due to more variation in computing resources, including different core types and frequency levels. The more difference of core's computing power, the less fairness amount. The computing power of big and little clusters are equal, when the big to little cluster speedup (ratio of execution time of running application on big to execution time of running those application on the little cluster) is one. Fig. 7 shows big to little cluster speedup of some programs at different big cluster frequencies, while little cluster's frequency is fixed at 1400 MHz. As it is seen in Fig. 7, two lessons can be learned: 1) due to different applications behavior, speedup of applications are different at fixed big and little frequencies and, 2) the speedup value of one (equal clusters computing power) is achieved at big cluster's frequency lower than 1400 MHz. These two insights are true for all applications of SPEC2006 that we examined, however only six applications have been reported for readability. We conducted an extensive experiment to assess the impact of CPU frequency on fairness. Fig. 8 shows the uniformity

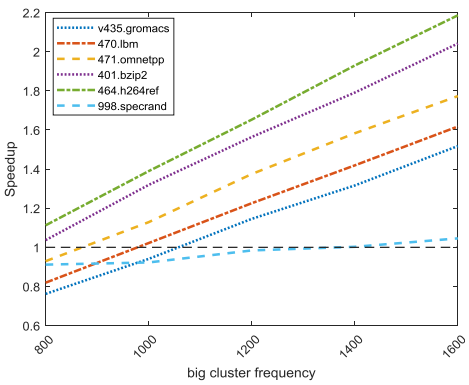

Fig. 7. The big to little cluster speedup of some benchmarks at different big cluster frequencies, while little cluster's frequency is fixed 1400 MHz.

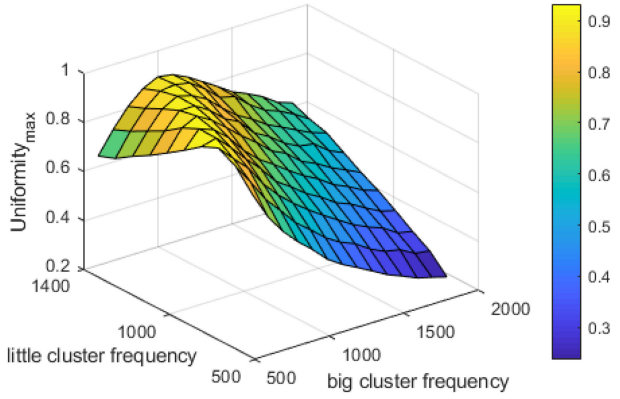

Fig. 8. Uniformity$_{max}$ of all big and little frequency level combinations for different sets of W3(HHHHHH) workload.

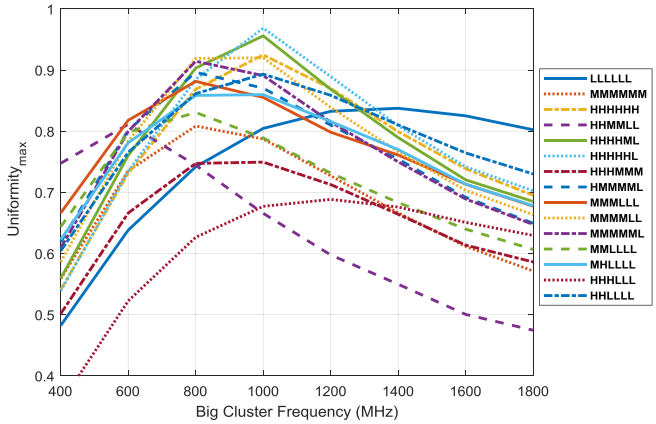

Fig. 9. Uniformity$_{max}$ of executing all workloads at different big core frequencies, while the little cluster's frequency is 1400 MHz.

values of all big and little frequency levels combinations for HHHHHH workload. The greater the uniformity value, the more fairness in the system. As illustrated in Fig. 8: 1) the highest values of uniformity are for points with big cluster frequencies lower than 1400 MHz, 2) the maximum values of uniformity are located on the zone where the big cluster's frequency is lower than the little cluster (when speedup is one). Little cluster consumes low power, so it is not necessary to exploit DVFS for its power management. Therefore, the frequency of little cluster is fixed at 1400 MHz similar to [28], [31]. So, for next experiments the uniformity of different workloads is calculated for various big core frequencies ranging from 400 MHz to 1800 MHz, while little cluster's frequency is fixed at 1400 MHz (Fig. 9).

As demonstrated in Fig. 9, by scaling up big cluster's frequency from 400 MHz to $freq_{eq}$ (depending on the application is between 600 MHz to 1400 MHz), uniformity rises in all scenarios. This is due to that the difference of two core's computing power decreases and big core computing power is getting closer to little core until two cluster's computing power become rather equal at $freq_{eq}$. But, when frequency scales up more than $freq_{eq}$ uniformity falls substantially, because big core computing power becomes larger than the little cores. For example, for the HHHHHL workload the maximum fairness happens at 1000 MHz for big cluster, however for the MMMMMM workload, the frequency for maximum fairness is at 800 MHz.

5.4 Results and Discussion

The proposed HFEE framework considers fairness and energy efficiency for asymmetric multi-cores, simultaneously, when executing different program types. Since our evaluation board provides power sensor per cluster, the pre-processing phase is not performed online and this phase is carried out offline, however, in case the target platform supports power sensor per core, it can be applied online. Ranking, mapping, and frequency scaling phases of the HFEE framework are repeated every one second (epoch duration). When a program completes its execution, it is not relaunched and the number of programs decreases until all of them finish. The K parameter, α_1, α_2, and $Uniformity_{\text{threshold}}$ are user-defined values specified before the start of scheduling.

5.4.1 HFEE Versus Other Schedulers

HFEE is compared against Linux standard scheduler for heterogeneous architectures. Also, fairness-aware (Min-Fair [31]) and energy-aware (DTPM [28]) schedulers are implemented for comparison. Each algorithm for every workload combination is repeated 100 times and the average is plotted in the Figs. 10 and 11.

Fig. 10 shows the uniformity of different schedulers for the representative workloads in terms of $Uniformity_{max}$. Two fairness agnostic techniques, Linux standard scheduler and DTPM demolish the uniformity in all workloads and has the least fairness compared to HFEE and Min-Fair. HFEE improves uniformity on average by about 57.6 and 51 percent compared to Linux standard scheduler and DTPM, respectively. Min-Fair scheduler focuses on the fairness and produces the best result of fairness and has a 3 percent higher uniformity than HFEE on average, without considering energy efficiency.

For a performance comparison, the makespan of the schedulers are measured. EDP represents the energy efficiency metric where the lower EDP, the more energy efficiency. Fig. 11 shows the makespan, energy delay product (EDP), and energy consumption of all schedulers for different workload combinations. As it is seen in Fig. 11, HFEE outperforms all other schedulers in makespan and EDP, while DTPM achieves the best result in energy consumption. After running various sets of programs, the proposed framework appears, on average, to improve EDP by about 68, 57, and 61 percent in comparison with Linux, Min-Fair, and DTPM, respectively. The experimental results also indicate that HFEE reduces makespan by about 57, 27, and 65 percent when compared to Linux, Min-Fair, and DTPM, correspondingly. Energy consumption of HFEE is about 9 percent more than DTPM, while 33 and 41 percent less than Linux and Min-Fair, respectively. The average improvement of uniformity, EDP, makespan, and energy consumption of HFEE compared to other schedulers for all 15 workloads are shown in Fig. 12.

5.4.2 Clusters' Usages Analysis

Additional experiments are performed for better behavior analysis of all schedulers. Big and little clusters' usages (the ratio of execution time on big or little cluster to the total execution time) are reported as the first parameter for schedulers'

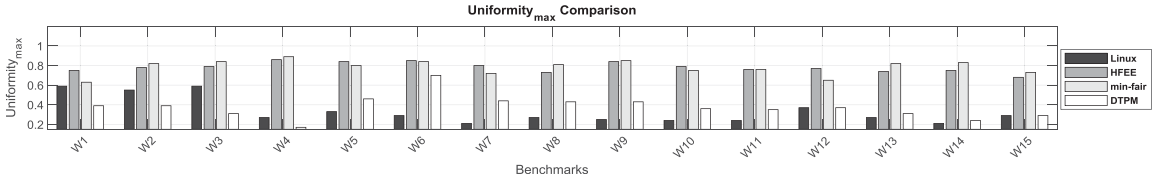

Fig. 10. Uniformity comparison of different schedulers for the representative workloads in terms of $Uniformity_{max}$.

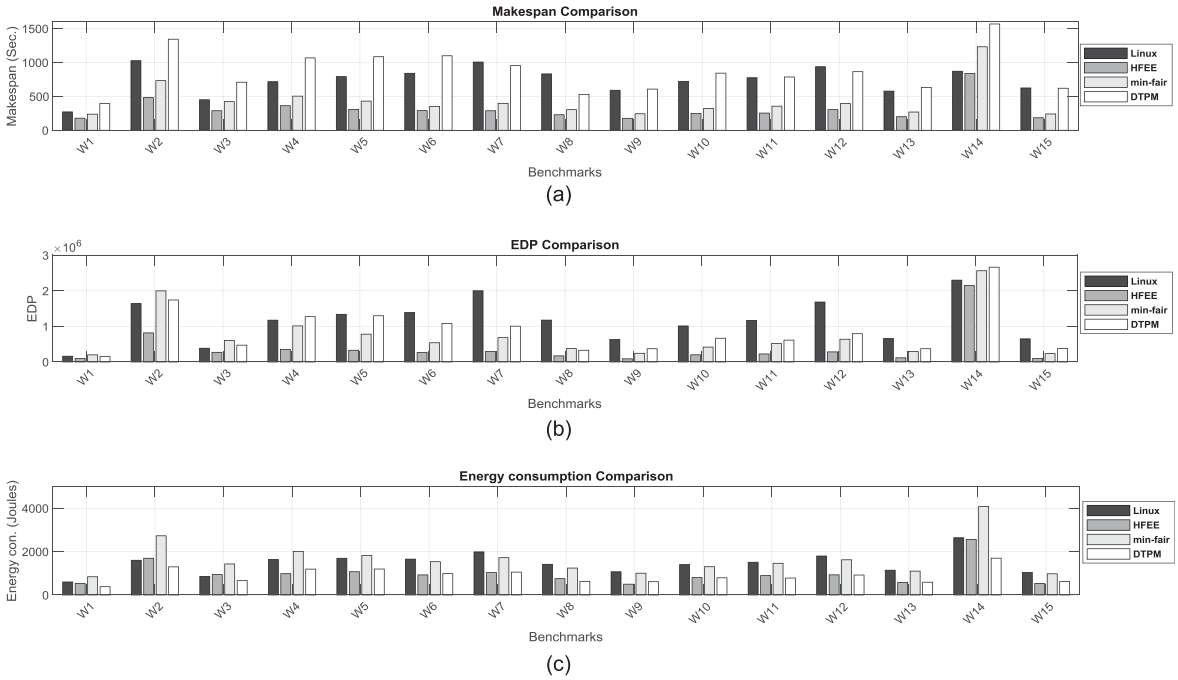

Fig. 11. (a) Makespan, (b) Energy delay product (EDP), and (c) Energy consumption comparison of different schedulers for the representative workloads.

attitude observation. This parameter is a key factor that affects makespan significantly. Makespan decreases if we use big cluster more than little cluster.

The Fig. 13 shows the big and little cluster' usage for different algorithms. The proposed framework (HFEE) uses little cluster when the number of programs is more than big cluster core count. It does not use little cluster, in case the program count is lower than big cluster core count. This improves makespan consequently.

Min-Fair scheduler tries to reach higher fairness by almost equal usage of big and little clusters. This policy improves fairness at the cost of higher makespan. Min-Fair uses both big and little clusters to improve fairness even when the program count is lower than big cluster core count which degrades makespan. However, HFEE uses big cluster in these cases which improves makespan.

DTPM just tries to save energy consumption and do not consider fairness. This scheduler uses little cluster more than the big cluster to improve energy consumption, therefore, it results in longer makespan. Linux standard scheduler is heterogeneity agnostic and uses little cluster more than big cluster, which results in higher makespan.

5.4.3 DVFS Analysis

The frequency level usage of clusters (the period of time a specific frequency in a cluster is used) is another important metric which should be studied to better understand the behavior of each scheduler. According to our observations, all schedulers use high frequency levels for little cluster and none of the schedulers changes the little cluster's frequency level. The big cluster's frequency level usage for different algorithms running HHHHHH workload are shown in Fig. 14.

HFEE controls only big cluster's frequency. The little cluster works at its highest frequency level. HFEE tries to improve uniformity through adjusting the big cluster's frequency, so that the two clusters operate almost with the same computing power, which results in higher fairness. In HFEE when the program count is lower than big cluster core count, all remaining applications are migrated to big cluster, which improves fairness significantly. On the other

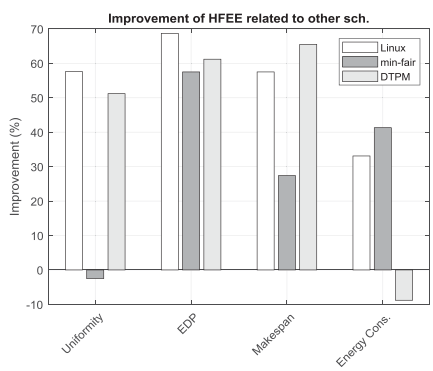

Fig. 12. The average improvement of HFEE related to other sch.

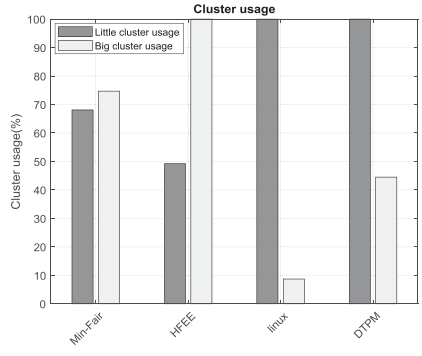

Fig. 13. The average big and little cluster usage of different workloads of different schedulers.

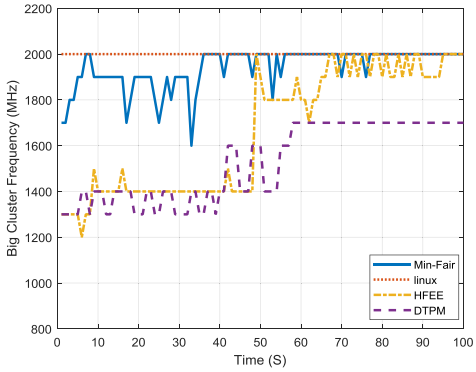

Fig. 14. The big cluster's frequency level usage for different algorithms running HHHHHH workload.

hand, HFEE usually uses low frequency levels of big cluster, which results in lower energy consumption compared to other schedulers (except DTPM, which is just energy-aware algorithm and does not consider fairness). Lower values of makespan and energy consumption of HFEE result in lower EDP compared to other approaches and makes HFEE a more effective algorithm. DTPM (which does not consider fairness) uses low frequency levels of big cluster and uses little cluster more than big cluster, which results in lower energy consumption, however, degrades the makespan and EDP consequently. Min-Fair scheduler focuses on the fairness and does not consider energy efficiency and tries to improve fairness as much as possible. Since it does not consider energy efficiency, it operates at high frequency levels, which results in more energy consumption. Also, because it migrates applications between big and little clusters, it lengthens makespan, and increases EDP. Linux standard scheduler always works under high frequency levels which results in higher energy consumption. According to Fig. 13, since it's usage of little cluster is very high, it has a longer makespan. Its longer makespan and higher energy consumption result in higher EDP.

5.4.4 HFEE Overhead

Another experiment is conducted to calculate scheduling overhead of HFEE framework. For this purpose, four cores are dedicated (two big and two little cores) for programs execution and one extra core (fifth core) is enabled for executing only HFEE framework. All 15 workload are run under this new condition (five enabled cores) and their makespan are compared to the general condition (four enabled cores where the HFEE framework is executed alongside other applications). The results show about 1.7 percent difference in makespan for these two cases (i.e., the scheduling overhead is about 1.7 percent).

6 Conclusion and Future Work

In this paper we explore the fairness and energy efficiency management via frequency scaling support for heterogeneous multi-core processors. The analysis concludes that frequency scaling plays a critical role in fair scheduling and energy efficiency can significantly improve by considering the performance per watt ratio of big to little cluster. To mitigate the problem, the current study proposes a heterogeneous fairness-aware energy efficient framework that utilizes DVFS to guarantee a minimum user-defined fairness, considering energy efficiency. The experimental results obtained by SPEC CPU2006 benchmark running on a real HMP platform indicate that the proposed framework outperforms Linux standard scheduler and two energy efficient and fairness-aware schedulers in terms of energy efficiency and fairness. Future work will extend the proposed framework to support resource contention management among the running programs and also explore the effect of contention on fairness and energy efficiency simultaneously. The other further study is the extension of FSM to support local DVFS.

References

[1] A. Lukefahr, S. Padmanabha, R. Das, R. Dreslinski Jr, T. F. Wenisch, and S. Mahlke, "Heterogeneous microarchitectures trump voltage scaling for low-power cores," in *Proc. Parallel Archit. Compilation Techn.*, 2014, pp. 237–250.

[2] A. Das, B. M. Al-Hashimi, and G. V. Merrett, "Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 2, pp. 1–24, 2016.

[3] S. Sarma, T. Muck, L. A. Bathen, N. Dutt, and A. Nicolau, "SmartBalance: A sensing-driven linux load balancer for energy efficiency of heterogeneous MPSoCs," in *Proc. Des. Autom. Conf.*, 2015, pp. 1–6.

[4] S. Jain, H. Navale, U. Ogras, and S. Garg, "Energy efficient scheduling for web search on heterogeneous microservers," in *Proc. Int. Symp. Low Power Electron. Des.*, 2015, pp. 177–182.

[5] J. Kong, S.W. Chung, and K. Skadron, "Recent thermal management techniques for microprocessors," *ACM Comput. Survey*, vol. 44, no. 3, 2012, Art. no. 13.

[6] T. R. Mück, Z. Ghaderi, N. D. Dutt, and E. Bozorgzadeh, "Exploiting Heterogeneity for Aging-aware Load Balancing in Mobile Platforms," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 3, no. 1, pp. 25–35, First Quarter 2017.

[7] A. Annamalai, R. Rodrigues, I. Koren and S. Kundu, "An opportunistic prediction-based thread scheduling to maximize throughput/watt in AMPs," in *Proc. Parallel Archit. Compilation Techn.*, 2013, pp. 63–72.

[8] S. Sharifi, R. Ayoub, and T. S. Rosing, "TempoMP: Integrated prediction and management of temperature in heterogeneous MPSoCs," in *Proc. Des. Autom. Test Eur.*, 2012, pp. 593–598.

[9] J. Yun, J. Park, and W. Baek, "Hars: A heterogeneity-aware runtime system for self-adaptive multithreaded applications," in *Proc. Des. Autom. Conf.*, 2015, pp. 1–6.

[10] G. Ayad, A. Acquaviva, E. Macii, B. Sahbi, and R. Lemaire, "HW-SW Integration for energy-efficient/variability-aware computing," in *Proc. Des. Autom. Test Eur.*, 2013, pp. 607–611.

[11] V. Petrucci *et al.*, "Octopus-man: QoS-driven task management for heterogeneous multicores in warehouse-scale computers," in *Proc. High Perform. Comput. Archit.*, 2015, pp. 246–258.

[12] M. K. Tavana, A. Kulkarni, A. Rahimi, T. Mohsenin, and H. Homayoun, "Energy-efficient mapping of biomedical applications on domain-specific accelerator under process variation," in *Proc. Int. Symp. Low Power Electron. Des.*, 2014, pp. 275–278.

[13] L. Sawalha and R. D. Barnes, "Phase-based scheduling and thread migration for heterogeneous multicore processors," in *Proc. Parallel Archit. Compilation Techn.*, 2012, pp. 493–494.

[14] K. Chandramohan and M. F. O'Boyle, "Partitioning data-parallel programs for heterogeneous MPSoCs: time and energy design space exploration," *ACM SIGPLAN Notices*, vol. 49, no. 5, pp. 73–82, 2014.

[15] Y. Zhang, L. Zhao, R. Illikkal, R. Iyer, A. Herdrich, and L. Peng, "QoS management on heterogeneous architecture for parallel applications," in *Proc. Int. Conf. Comput. Des.*. 2014, pp. 332–339.

[16] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in *Proc. Int. Symp. Comput. Archit.*, 2012, pp. 213–224.

[17] Q. Chen and M. Guo, "Adaptive workload-aware task scheduling for single-ISA asymmetric multicore architectures," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 1, pp. 1–8, 2014.

[18] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt, "Accelerating critical section execution with asymmetric multicore architectures," *IEEE Micro*, vol. 30, no. 1, pp. 60–70, Jan./Feb. 2010.
[19] K. Van Craeynest and L. Eeckhout, "Understanding fundamental design choices in single-ISA heterogeneous multicore architectures," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 1–23, 2013.
[20] J. Huang, A. Raabe, C. Buckl, and A. Knoll, "A workflow for runtime adaptive task allocation on heterogeneous MPSoCs," in *Proc. Des, Autom. Test Eur.*, 2011, pp. 1–6.
[21] I. Anagnostopoulos, A. Bartzas, G. Kathareios, and D. Soudris, "A divide and conquer based distributed run-time mapping methodology for many-core platforms," in *Proc. Des. Autom. Test Eur.*, 2012, pp. 111–116.
[22] P. Huang, H. Yang, and L. Thiele, "On the scheduling of fault-tolerant mixed-criticality systems," in *Proc. Des. Autom. Conf.*, 2014, pp. 1–6.
[23] M. M. Ozdal, A. Jaleel, P. Narvaez, S. Burns, and G. Srinivasa, "Trace alignment algorithms for offline workload analysis of heterogeneous architectures," in *Proc. Int. Conf. Comput.-Aided Des.*, 2013, pp. 654–661.
[24] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *Proc. Archit. Support Program. Lang. Operating Syst.*, 2013, pp. 77–88.
[25] C. Tan, T. S. Muthukaruppan, T. Mitra, and L. Ju, "Approximation-aware scheduling on heterogeneous multi-core architectures," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2015, pp. 618–623.
[26] M. Kim, K. Kim, J. R. Geraci, and S. Hong, "Utilization-aware load balancing for the energy efficient operation of the big. LITTLE processor," in *Proc. Des. Autom. Test Eur.*, 2014, pp. 1–4.
[27] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *Proc. Des. Autom. Conf.*, 2013, Art. no. 174.
[28] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, "Algorithmic optimization of thermal and power management for heterogeneous mobile platforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 544–557, Mar. 2018.
[29] A. Garcia-Garcia, J.C. Saez, and M. Prieto-Matias, "Contention-aware fair scheduling for asymmetric single-ISA multicore systems," *IEEE Trans. Comput.*, vol. 67, no. 12, pp. 1703–1719, 2018.
[30] J. Feliu, J. Sahuquillo, S. Petit, and J. Duato, "Perf&Fair: A Progress-Aware Scheduler to Enhance Performance and Fairness in SMT Multicores," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 905–911, 2017.
[31] C. Kim and J. Huh, "Exploring the design space of fair scheduling supports for asymmetric multicore systems," *IEEE Trans. Comput.*, vol. 67, no. 8, pp. 1136–1152, Dec. 2018.
[32] K. Van Craeynest, S. Akram, W. Heirman, A. Jaleel, and L. Eeckhout, "Fairness-aware scheduling on single-ISA heterogeneous multicores," in *Proc. Parallel Archit. Compilation Techn.*, 2013, pp. 177–187.
[33] I. S. Moreno, R. Yang, J. Xu, and T. Wo, "Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement," in *Proc. Int. Symp. Auton. Decentralized Syst.*, 2013, pp. 1–8.
[34] C. Kim and J. Huh, "Fairness-oriented OS scheduling support for multicore systems," in *Proc. Int. Conf. Supercomputing*, 2016, pp. 29:1–29:12.

Bagher Salami received the MSc degree in computer engineering from the Ferdowsi University of Mashhad (FUM), Iran, in 2013, he is currently working toward the PhD degree in FUM. His current research interests include OS scheduling, fairness, shared resource contention, performance, power, and thermal management of heterogeneous multicore processors.

Hamid Noori received the BSc degree from the Sharif University of Technology, in 1996, the MSc degree in computer systems architectures from the Amirkabir University of Technology, in 2000, and the PhD degree from the Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan, in 2007. He is currently an assistant professor with the Department of Computer Engineering, Ferdowsi University of Mashhad (FUM). Before joining FUM at January 2011, he was an assistant professor at the University of Tehran, College of Engineering, School of Electrical and Computer Engineering, from January 2009. He is the director of Advanced Computer Architecture Laboratory (ACAL).

Mahmoud Naghibzadeh received the undergraduate degree from USC, the graduate degree from the University of South Florida, USA, the MS degree in computer science, and the PhD degree in computer engineering, both from the University of Southern California (USC), USA. He is currently a full professor with the Ferdowsi University of Mashhad, Iran. He was a visiting professor at University of California-Irvine, UCI, USA, in 1991 and a visiting professor at Monash University, Australia, in 2003–2004. His research interests include scheduling aspects of real-time systems, grid, cloud, multiprocessors, multicores, and GPGPUs and also bioinformatics algorithms, especially genomics and proteomics. He has published numerous papers as well as eight books and has been the chairman of two international conferences and technical chair of many others. He is the reviewer of many journals and a member of many computer societies. He is the recipient of many awards including MS and PhD study scholarship.