The Journal of Supercomputing (2022) 78:7729–7748
https://doi.org/10.1007/s11227-021-04159-8

# Online energy-efficient fair scheduling for heterogeneous multi-cores considering shared resource contention

**Bagher Salami**[1] · **Hamid Noori**[1] · **Mahmoud Naghibzadeh**[1]

Accepted: 19 October 2021 / Published online: 3 January 2022


**Abstract**
Heterogeneous multi-core processors (HMP) are dual-objective hardware platforms which integrate both high-performance and low power consumption processors. Investigation of simultaneous fairness and energy efficiency could widen their applicability and reveal their strengths. This paper proposes a scheduling framework considering energy efficiency, shared resource contention, and fairness for heterogeneous multi-core processors. The presented framework is implemented and evaluated on a real HMP platform. The obtained experimental results via SPEC CPU2006 benchmark indicate that the proposed framework surpasses Linux and four other schedulers in terms of fairness (58% on average) and energy efficiency (37% on average). The source code of the proposed framework and the opponent algorithms are available online at https://github.com/baghers/CEEF.

**Keywords** Shared resource contention · Energy efficiency · Fairness · Scheduling · Heterogeneous multi-core · big.LITTLE

## 1 Introduction

Heterogeneous multi-core processors (HMP) have different computational capabilities. HMP as a promising paradigm show potential performance improvement and power consumption reduction. This fact has yielded the development of recent HMPs like ARM's big.LITTLE processors, where powerful cores (big cluster) are integrated with low-performance cores (little cluster). Both clusters share the same

✉ Hamid Noori
hnoori@um.ac.ir

Bagher Salami
bagher.salami@mail.um.ac.ir

Mahmoud Naghibzadeh
naghibzadeh@um.ac.ir

[1] Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

instruction set architecture (ISA), while each cluster has a particular cache size, pipeline stages, and voltage and frequency levels.

Multi-core processors share resources such as caches and memory controllers with adjacent cores, which results in performance degradation due to contention for shared resources. The shared resources consist of last level cache (L2 or L3), the memory bus or interconnects, DRAM controllers, and pre-fetchers [1]. An application may slow down by hundreds of percent if it shares resources with applications running on adjacent cores compared to running alone [2]. There are several contention minimization techniques, such as last-level-cache partitioning [3], DRAM controller [4], and thread-level scheduling [2] (mapping threads to the cores). Different mappings lead to different mixes of threads competing for shared resources. Thread scheduling techniques try to find the mappings that result in less shared resource contention, achieving the best possible performance.

Energy efficiency is also achieved through optimizing both the performance (execution time) and energy consumption. The execution time and energy consumption of any program in HMPs are affected by cluster type, cluster's frequency, and the program behavior. Various scheduling algorithms have been proposed to improve energy efficiency of HMPs [5–11].

In contention-aware algorithms, performance is improved alongside other criteria such as reliability [12] and fairness [2]. A scheduler is considered fair if all programs suffer from the same performance degradation normalized to the isolated run on a base configuration [13, 14]. Likewise, in energy-efficient schedulers, energy and performance are enhanced alongside other factors such as reliability [9], aging rate [7], and temperature [11]. Dynamic voltage–frequency scaling (DVFS) remains one of the prevalent options for chip power management despite the overhead and associated static power (due to leakage current) [15]. This paper promotes the state of the art in fairness-aware energy-efficient scheduling, by ((1) managing energy efficiency online and (2) considering shared resource contention effects.

The disadvantage of offline energy efficiency management is that the applications should be known by the system and the tedious tasks profiling of the running applications are required, which limits the application of offline scheme. The phase changes of applications are also not considered in the offline energy efficiency management. This paper addresses the problem of considering contention, energy efficiency, and fairness scheduling (CEEF) of tasks online on HMPs.

The ratio of the IPS (instruction per second) per watt (IPW) of the big cluster to the little cluster for each program is an indicator of a program's suitability for each cluster type. According to our experiments, programs with higher energy efficiency ratio (EER) values are more energy efficient to run on the big cluster which enhances energy efficiency in terms of energy–delay product (EDP). Also, the problem of starvation and wait times of workloads are crucial issues, which have to be considered in the policy of co-execution selection. The theoretical analysis is performed to confront the problem of starvation by considering the wait time in program selection for co-executing on each cluster. Experimental results also demonstrate that the programs with the higher energy efficiency are more energy efficient to run on the big cluster and those with low ratio values are suitable for the little cluster. 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 and co-executing pair of programs on each cluster with less shared resource contention. Through proper task assignment to clusters, co-executing less shared resource contention pair on each cluster, and management of each cluster's voltage and frequency, a certain level of fairness known as the fairness threshold, is guaranteed, while energy efficiency (EDP) is improved.

The proposed framework is evaluated on a real ARM big.LITTLE multi-core processor and compared with standard Linux scheduler and four state-of-the-art schedulers. After running various experiments, the results demonstrate that the proposed scheduler guarantees the fairness threshold while improving overall energy efficiency. In summary, the present paper makes the following contributions:

- It addresses the problem of contention, energy efficiency, and fairness scheduling of tasks on HMPs with DVFS capability simultaneously.
- Unlike [16], the proposed online framework does not need any offline profiling of running applications and also considers cache contention. The time complexity of the scheduling process is low.
- Our proposed framework exploits a small set of performance metrics that can be easily extracted using performance monitoring unit available in commercial HMPs.

The remainder of the paper is organized as follows: The related work is discussed in Sect. 2, and the preliminaries are presented in Sect. 3. Section 4 describes our proposed framework in detail. In Sect. 5, the implementation and analysis results are discussed, and conclusions are made in Sect. 6.

## 2 Related work

The advent of the HMP era and the emphasis of criteria such as performance, fairness, and energy efficiency have forced us to reconsider the scheduling techniques. The prior works examination is started with covering the performance, energy efficiency, and fairness improvement proposals. Finally, the shared resource contention management schemes are investigated.

### 2.1 Performance optimization

Performance improvement in HMPs has been achieved through proper mapping of applications to different big and little cores. Workload memory intensity is one of the first schemes [17] to guide applications mapping by assigning CPU-intensive workloads to big cores, while memory-intensive workloads are run on the little cores. Memory intensity alone is not a correct indicator for workload to core mapping, which causes suboptimal scheduling [8]. Van Craeynest et al. [8] exploit ILP (instruction-level parallelism) and MLP (memory-level parallelism). They show that small cores provide better performance for computer-intensive workloads whose

subsequent instructions in the dynamic instruction stream are usually independent (i.e., high ILP) and big cores supply high performance for workloads that present a large amount of MLP.

Recent studies [16] suggest to exploit the big to little speedup factor for mapping applications to different core types. Performance improvement is achieved through mapping of applications with higher big to little speedup factor (SF) to big cores and applications with lower speedup factor to the little cores. Application's SF measurement is a crucial matter in the contemporary HMP performance management era. Sampling [2] is a premier suggestion to calculate application's SF via running each program on both big and little cores to extract IPC. Further schemes of application's SF calculation consist of estimating application's SF using performance models and extracted runtime information from performance monitoring unit (PMU) [2].

### 2.2 Energy efficiency management

Improving energy and performance requires assigning tasks to the appropriate core types and adjusting their frequencies to an appropriate value through DVFS (dynamic voltage–frequency scaling). The first effort of closed-loop (feedback-based controller) load balancing is SmartBalance [6]. Dissimilar to open-loop Linux load balancer, which assigns the tasks evenly, SmartBalance through three phases of sense, predict, and balance tries to assign the threads to the matched core type to achieve the best energy efficiency. Myungsun et al. [10] studied energy efficiency management through DVFS. They propose a utilization-aware load balancer via a utilization estimator to determine the most appropriate frequency and core type for a given set of programs. DTPM [11] is one of the latest studies of dynamic power and frequency management using power budget and power consumption predictors at different frequency level. DTPM specifies the maximum feasible frequency under the available power budget.

### 2.3 Fairness improvement

Several fairness definitions have been presented in the prior works. Frequently, a system is considered fair when all the running programs suffer the same slowdown corresponding to their isolated execution [2]. Performance asymmetry in HMPs is the main reason of different slowdown and low fairness. Asymmetry-aware Round-Robin (ARR) scheduler is considered as first approach of fair scheduling on HMPs through fair-sharing big core cycles [18]. RR improves fairness compared to fairness agnostic schedulers. However, as shown in [18], it makes a suboptimal fairness solution and degrades performance substantially.

Application's SF is considered when distributing big core cycles among the applications to improve both fairness and performance. In [19], RR and application's SF are combined to efficiently distribute big core cycles among different applications so that target fairness is guaranteed. The remain big core cycles are allocated to applications according to their SF. Progress-based proposals are novel schemes to optimize both fairness and performance on HMPs. Balanced progress of applications

causes even out slowdowns, improving both fairness and performance. Feliu et al. [13] present a progress-aware scheduler that estimates the progress experienced by the processes and gives priority to the processes with lower accumulated progress. HFEE [16] is the latest study of energy efficiency and fairness, including dynamic voltage–frequency scaling (DVFS). HFEE tries to improve energy efficiency through appropriate mapping of applications to different core types and make equal progress of applications via both progress management (software approach) and frequency scaling (hardware approach) to uniform application slowdowns, hence improving fairness.

### 2.4 Shared resource contention management

HMPs are not independent processors, where last level cache (L2 or L3), the memory bus or interconnects, DRAM controllers are shared with neighboring cores that results in performance degradation due to contention for shared resources. Contention-aware schedulers are very promising solutions at mitigating the performance loss of shared resource contention. Contention-aware schedulers identify which tasks should run near together and which tasks should run far away to mitigate the harmful effects of contention. The initial efforts of contention management are performed by Tian et al. [20] that assume the performance degradation of all possible pair of tasks are known. Then, they constitute the complete graph, while nodes of the graph are the tasks to be scheduled and the edges are the performance degradation of co-scheduled tasks to the same cluster. The optimal solution is a *minimum weight perfect matching* of the graph. However, the problem is NP-complete for HMPs with more than two cores per cluster [20]. Zhuravlev et al. [21] proposed a heuristic method known as distributed intensity (DI), where all tasks are sorted based on their miss rate and tasks are co-executed from the opposite ends of the list. Co-scheduling threads with complementary usage of shared resources is the most prevalent scheme of contention management. One of the novel researches of contention and fairness management has been presented in [2]. They exploit progress information of tasks to even out slowdowns to improve fairness, and complementary usage of bus transfer rate is used to mitigate the contention problem.

An issue in task scheduling that has not been addressed by previous works is the simultaneous consideration of energy efficiency, fairness, and contention to enhance task scheduling on the asymmetric multi-core processors. This overlooked problem is the motivation behind the current study.

## 3 Preliminaries

In this section, workload and platform models as well as energy efficiency and fairness metrics are investigated. Also, the challenges associated with energy efficiency and shared resource contention are studied.

### 3.1 System model and metrics

*Workload model.* There is a set of $m$ single thread programs as $P=\{p_1,p_2,\ldots,p_m\}$ with the possibility of the program count being greater than that of core count ($m \geq$ Core#). In this paper, the term *task* is used for both programs and tasks. Task scheduling epoch is $\tau$ seconds.

*Platform model*. The HMP platform consists of a single-ISA heterogeneous multicore processors with $n$ cores as $C=\{c_1,c_2,\ldots,c_n\}$. Cores are classified into $s$ set of clusters as $Z=\{z_1,z_2,\ldots,z_s\}$, and voltage/frequency of each cluster is adjusted dynamically. It is supposed that HMP is equipped with *hardware performance counters* and *power sensor* per cluster.

*Energy efficiency metrics*. Energy efficiency is defined as the combination of reduced energy consumption and performance improvement (execution time) [22–28]. The energy–delay product (EDP) is calculated by the product of the total amount of energy consumed and the runtime duration, where the higher the energy efficiency, the less the EDP. In contrast to EDP, the IPS per watt (IPW) is the total amount of committed instruction for every watt of power consumed. The more the IPW, the higher the energy efficiency is achieved.

*Fairness metric*. Previous studies of fairness management [13, 14, 16] define a scheduler as fair if equal-priority applications in a workload suffer the same slowdown due to sharing the system. In other words, a scheduler is considered fair if the variation of performance degradation normalized to isolated run (one second on the big and one second on the little core with the highest voltage and frequency) is minimal.

The slowdown of program $i$ under a scheduler is expressed as $\text{slowdown}_i = \frac{T_{sch.,i}}{T_{\text{isolated run},i}}$, where $T_{sch.,i}$ is the execution time of program $i$ and $T_{\text{isolated run},i}$ is the execution time of isolated run of program $i$. The evaluation of fairness is performed in terms of uniformity:

$$\text{Uniformity} = 1 - \frac{\sigma_S}{\mu_S} \tag{1}$$

where $\sigma_S$ and $\mu_S$ are the standard deviation and the average of *slowdown* values of all programs.

*Problem statement*. This study explores the problem of assigning $m$ single-thread tasks to one $n$ cores big.LITTLE processor. The shared resource contention among the tasks is considered, and voltage and frequency of clusters are identified dynamically such that energy efficiency is improved and system fairness is more than or equal to the user-defined threshold.

### 3.2 Shared resource contention impact on performance and energy efficiency

The impact of shared resource contention on performance of HMPs is fully investigated in previous researches [2–4], while the resource contention impacts on energy efficiency of schedulers are overlooked matters in task scheduling algorithms. An experiment is conducted, and all dual pairs of programs from SPECCPU2006 benchmark suite are executed, and the values of instruction per second (IPS) and IPW are extracted for each dual pair that is shown in Fig. 1. As shown in Fig. 1, the diagram of IPS is much correlated with the diagram of IPW. Pearson product moment correlation coefficient (PPMCC) [29] is a prevalent correlation computation method that allows calculating the exact correlation of IPS to IPW. PPMCC is used to measure the correlation of two variables X and Y. The r coefficient is calculated using Eq. 2:

$$r = \frac{\sum_{i=1}^{N}\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right)}{\sqrt{\sum_{i=1}^{N}\left(X_i - \overline{X}\right)^2}\sqrt{\sum_{i=1}^{N}\left(Y_i - \overline{Y}\right)^2}}, \tag{2}$$

where $N$ is the number of sampled data and $\overline{X}$, $\overline{Y}$ are the averages of $X$ and $Y$ variables, respectively. The relationship between $X$ and $Y$ is perfect, when r is 1 or $-1$. The negative value indicates that if $X$ variable increases, $Y$ will decrease. The $r$ coefficient of IPS and IPW is about 0.801, which shows a high relationship between IPS and IPW. In other words, improving IPS of workload execution boosts the IPW consequently. Recent processors feature a performance monitoring unit consisting of a set of counters that can be programmed to extract different events during the execution of tasks.

In order to find out the contention impact of application's co-execution on performance (IPS), one experiment is conducted to find out the performance counter (events) which has the greatest correlation with performance (IPS) of each application's pair. Most correlated performance counter with performance is used as performance increment/decrement identifier. After identifying the most

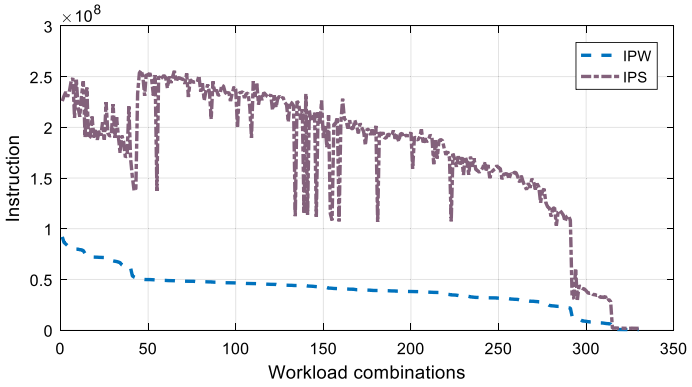


**Fig. 1** IPW and IPS of various dual pairs of workloads

correlated performance counter ($PC_m$) with IPS of application's pairs, the application's pair with the highest $PC_m$ is selected to co-execute at the same time. To do so, all 756 ($27 \times 28$) dual applications pairs of SPECCPU2006 benchmark suite are run and the average IPS of each pair is extracted. Then, each single program of SPECCPU2006 benchmark suite is run alone and for each program the amount of 30 events (performance counters) are logged. For each performance counter, the average ($corr_{\_avg}$) and difference ($corr_{\_dif}$) of performance counter values of two programs running in isolation are correlated with the IPS value of running two programs in shared. Figure 2 shows correlations of average and difference of various performance counter values of dual programs running in isolation with the IPS value of running two programs in shared resources. As shown in Fig. 2, $corr_{avg}$ of IPC has the strongest correlation value among the other events, so it is concluded that if two tasks with the highest IPC values in isolation are co-executed, the greatest performance (IPS) and energy efficiency (IPW) are achieved. Our proposed framework uses this event as a metric to mitigate contention through proper selection of paired tasks in each cluster. Therefore, the larger the IPC value for a workload, the larger the performance and energy efficiency of workload and vice versa.

## 4 CEEF framework

The proposed hierarchical CEEF framework is shown in Fig. 3 and consists of three components: (1) *Energy Efficiency Management (EEM)* exploits CPU power sensors and performance counters for energy-efficient assignment of tasks to big and little clusters; (2) *Contention Administration (CA)* specifies which pair of tasks has less shared resource contention and more energy efficient for running together, meanwhile preventing starvation of tasks; (3) *Fairness Improvement (FI)* boosts scheduling fairness through intra-cluster fairness enhancement. The following subsections fully describe the different parts of the CEEF framework.

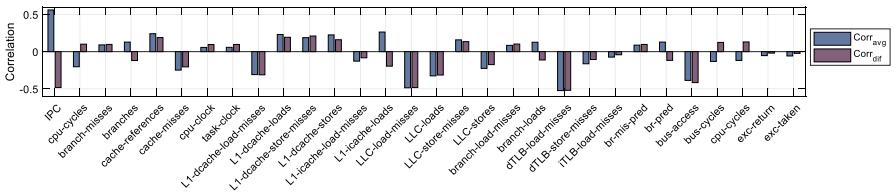


**Fig. 2** The correlation of average ($corr_{\_avg}$) and difference ($corr_{\_dif}$) of different performance counter values of programs running in isolation with the IPS value of running two programs simultaneously

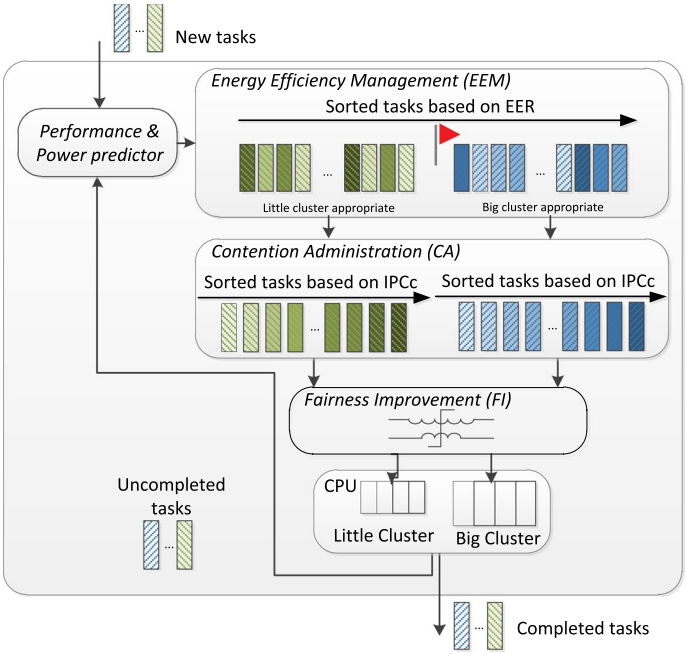


**Fig. 3** Contention, energy-efficient, and fairness-aware framework

### 4.1 Energy efficiency management (EEM)

Appropriate tasks to clusters (core types) mapping on HMPs improve energy efficiency significantly [5]. An energy efficiency speedup factor known as energy efficiency ratio (EER) has been proposed in [16]. EER is described as the ratio of instruction per watt on big to instruction per watt on the little cluster and is expressed as:

$$EER(i) = \frac{\frac{IPS(i)_{\text{big}}}{P(i)_{\text{big}}}}{\frac{IPS(i)_{\text{little}}}{P(i)_{\text{little}}}} = \frac{IPS(i)_{\text{big}} * P(i)_{\text{little}}}{IPS(i)_{\text{little}} * P(i)_{\text{big}}} \tag{3}$$

where $IPS(i)_{\text{big}}$ and $IPS(i)_{\text{little}}$ are the average instruction per second (IPS) of program $i$ on big and little cores, respectively, extracted using the CPU performance counters. The average power consumption of program $i$ running on a big core is denoted by $P(i)_{\text{big}}$, and $P(i)_{\text{little}}$ is also the average power consumption of program $i$ running on a little core. The measurement of these four values is an important problem in EER calculation. Sampling is the simplest scheme of IPS and power measurement, where each task is executed on both core types and extracted values are used to calculate EER. The major issue of this technique is the high-performance overhead of sampling. The IPS and power prediction is a preferable approach to calculate EER with low performance overhead. The linear regression models in [30] are exploited to

predict IPS and power consumption of tasks on a core type based on the IPS, LLCM (last level cache misses), and power consumption of tasks on the other core type. The task performance and power consumption prediction are extracted by collecting performance and power consumption data from a set of nominating workloads in the system and constructing the following linear regression models:

$$IPS_{\text{little}} = w_1 * IPS_{\text{big}} + w_2 * LLCM_{\text{big}} + w_3 \tag{4}$$

$$IPS_{\text{big}} = w_4 * IPS_{\text{little}} + w_5 * LLCM_{\text{little}} + w_6 \tag{5}$$

$$LLCM_{\text{little}} = w_7 * IPS_{\text{big}} + w_8 * LLCM_{\text{big}} + w_9 \tag{6}$$

$$LLCM_{\text{big}} = w_{10} * IPS_{\text{little}} + w_{11} * LLCM_{\text{little}} + w_{12} \tag{7}$$

$$P_{\text{little}} = w_{13} * IPS_{\text{big}} + w_{14} * LLCM_{\text{big}} + w_{15} \tag{8}$$

$$P_{\text{big}} = w_{16} * IPS_{\text{little}} + w_{17} * LLCM_{\text{little}} + w_{18} \tag{9}$$

These performance and power prediction models are extracted from performance and power data achieved by running different programs of the SPEC CPU2006 benchmark suite individually on each core type, and the coefficients $w_1$ to $w_{18}$ are derived through curve fitting. After the EER value of each task is specified, energy efficiency improves through assigning tasks with higher EER to big cores and mapping lower EER tasks to little cores; thus, the tasks set is divided into high EER and low EER task sets and are assigned to big and little clusters, respectively.

### 4.2 Contention administration (CA)

After determining energy-efficient task sets for each cluster, task selection for co-execution on each cluster is performed. In order to mitigate the effect of shared resource contention, the selected co-running tasks on each cluster must have less contention among each other. According to previous contention investigations, the selected tasks with higher IPC in isolation cause less contention and higher energy efficiency.

The IPC of tasks in isolated are unknown, and accurate estimation of isolated IPC is out of scope of this paper. An adequate approximation of stand-alone IPC of a task is to consider IPC at the state of low-contention co-execution. Co-running tasks with the highest IPC causes less contention. The IPC of a task is measured during the execution of the low contention (co-running with high IPC tasks) and used as an estimation of its stand-alone IPC for the following epochs. A history table is used to store the estimation of stand-alone IPCs of all tasks.

However, mitigating contention impact through selecting high IPC tasks causes starvation and increase in wait time of some tasks, which can spoil fairness of the scheme. To prevent starvation, the wait time of each task is considered along with IPC.

For this purpose, IPC of tasks are normalized (i.e., they are bounded between 0 and 1):

$$IPC_N(i) = \frac{IPC(i) - \min_{j \in P}(IPC(j))}{\max_{j \in P}(IPC(j)) - \min_{j \in P}(IPC(j))} \tag{10}$$

Then, IPC with considering wait time are represented by $IPC_C(i)$ as:

$$IPC_C(i) = IPC_N(i) + \left(1 - IPC_N\right).\frac{K(i)}{W_{max}} \tag{11}$$

where *K(i)* denotes the number of epochs waiting for a CPU and $W_{max}$ is the maximum number of epochs that a task can wait for CPU. $W_{max}$ is a user-defined value which should be specified before the start of scheduling. Equation 11 guarantees that all tasks receive CPU time before at most $W_{max}$ epochs. There is a history table for each cluster that contains the $IPC_c$ of all tasks. All tasks are sorted based on their $IPC_c$, and tasks with higher $IPC_c$ values are selected to co-execute alongside each other.

### 4.3 Fairness improvement (FI)

Fair scheduling is defined as scheduling tasks such that all tasks suffer the same slowdown corresponding to their isolated run. The performance asymmetry affects task's slowdown substantially. *Performance asymmetry* originates from the different high-performance (big cores) and low-power (little cores) architecture of HMPs and produces diverse speedups (slowdowns) that endanger the fairness. Different CPU frequency levels can be another source of performance asymmetry. However, it also can be an effective instrument to reduce performance asymmetry and boosting fairness through making two core types of speedups (computing power) almost the same. If two cluster's computing power becomes rather equal, performance asymmetry decreases and fairness improves. DVFS is a tool which can be used to equalize two cluster's computing power, which causes the tasks taking the equal processing resources and even out tasks slowdowns, hence improving fairness.

In our previous study [16], a finite-state machine (FSM) fairness manager is presented via adjusting the cluster's frequency. As shown in Fig. 4, FSM fairness manager consists of five states; each state is represented by the notation of (uniformity, frequency) and enumerated as: {(L, L), (L, H), (H, L), (H, H)}. Two uniformity states and two frequency states have been considered known as low and high, which are expressed as {L, H} for short. When the current uniformity is lower than $Uniformity_{threshold}$ (a user-defined value), the system is in the low fairness state and high fairness state happens when the current uniformity is higher than $Uniformity_{threshold}$. The little cluster always operates at its maximum frequency (similar to [11, 19]), and FSM only adjusts the big cluster frequency. When the big cluster's frequency is under $f_{\text{threshold}}$, the frequency state is low, and the frequency state is high, when the big cluster's frequency is more than $f_{\text{threshold}}$. $f_{\text{threshold}}$ described in [16] is a frequency level where for higher big cluster's frequency level than $f_{\text{threshold}}$, uniformity does not improve anymore. The

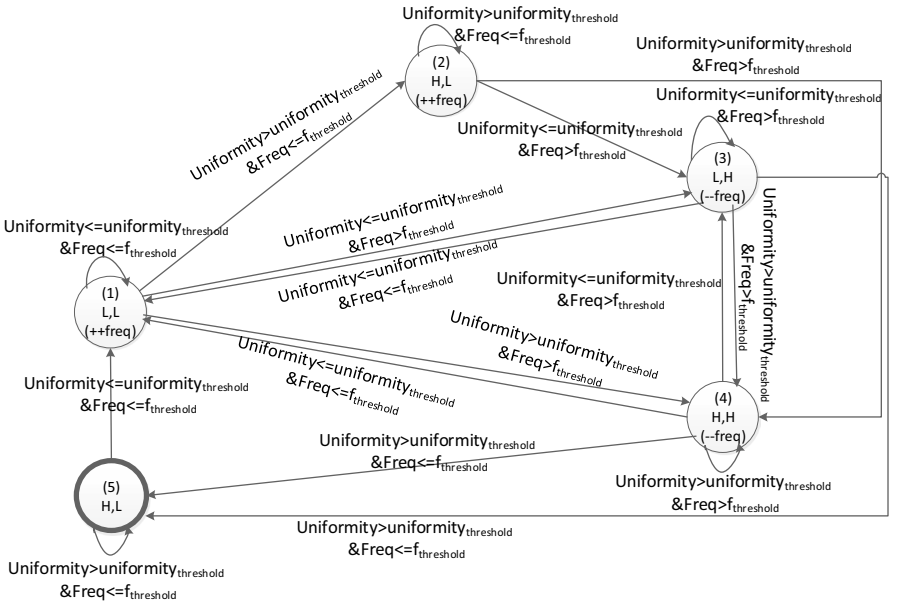


**Fig. 4** FSM fairness manager through DVFS

processor at each scheduling epoch can be in one of the five states. When the system reaches the optimal state five, uniformity is high and frequency in this state remains fixed.

### 4.4 Complexity analysis

Given the numbers of cores |C| and programs |P|, the proposed scheduler at each scheduling epoch has the complexity of $|P| \times \log(|P|)$ for sort of tasks based on EER and $IPC_c$ in *EEM* and *CA* phases. Also, the fairness management phase has the complexity of |1|. If we assume $|P| >= |C|$, then the runtime is bounded by $O(|P| \times \log(|P|))$.

## 5 Experimental evaluation

The experimental results of different schemes for various workloads on a real platform and analysis of the obtained results are presented in the following.

### 5.1 Experimental setup

The proposed contention-, energy efficiency-, and fairness-aware scheduling framework is evaluated on a real ARM big.LITTLE processor. The embedded platform is an ODROID-XU3 board integrating the Exynos Octa 5422 chip with four Cortex-A15 3-way out-of-order and four Cortex-A7 2-way in-order cores. The operating frequencies of the big cluster range are from 200 MHz to 2 GHz and from 200 MHz to 1.4 GHz for the little cluster. Big cluster shares 2 MB L2 cache, and little cluster shares 512 KB L2 cache.

The size of DRAM is 2 GB which is inadequate to run eight programs on all eight cores simultaneously [19]; therefore, at most six programs are executed simultaneously. Six (three big and three little) cores are enabled, and the remaining are turned off. Also, Ubuntu MATE 16.04.3 (4.14 generic Kernel) and *Perf* library are employed. The frequencies of clusters are scaled through *cpufreq,* and *taskset* is used to assign a thread to a dedicate core and embedded power sensors of each cluster are exploited for power measurement. The SPEC CPU2006 benchmark suite is used, and applications are categorized in terms of instructions per second (IPS). IPS and IPC are calculated with the aid of cycle count. So, if we know IPC and cycle count, IPS is calculable. Also, if we know IPS and cycle count, IPC is calculable. The IPS values of different applications on the big core are extracted, while IPS values are within a range from $0.22\times10^9$ to $2\times10^9$. Each application is classified into one of three classes: 1) **L**ow (IPS $<1\times10^9$), 2) **M**edium ($1\times10^9<$ IPS $<1.5\times10^9$), and 3) **H**igh (IPS $>1.5\times10^9$), respectively. The experimental workload consists of various mixes of applications from different classes that are demonstrated in Table 1.

### 5.2 Performance and power consumption prediction

The performance and power consumption predictor models (Eqs. 4–9) are exploited to estimate the values for the next scheduling epoch. In this section, the accuracy of performance and power consumption predictors are investigated. Figure 5 illustrates the predicted versus actual values of IPS and power for the random execution of SPEC benchmark when running on a cluster and intended to be run on the other cluster. As given in Table 2, the performance prediction error (given by the normalized root-mean-square error) for little core when running on big core ($IPS_{\_\text{little}}$) is 3.3%, while the performance prediction error for big core when running on little core is ($IPS_{\_\text{big}}$) 1.7%. The power consumption prediction error of $P_{\_\text{little}}$ and $P_{big}$ is 2.9% and 3.8%, respectively. The acceptable prediction errors of predictors indicate that our estimation model can accurately predict the performance and power consumption of a task running on different core types.

### 5.3 Results and discussion

The proposed CEEF framework considers energy efficiency, contention, and fairness simultaneously, when executing different tasks. *EEM*, *CA*, and intra-cluster

**Table 1** Multi-application workload combinations

| Name | Applications | Name | Applications |
|---|---|---|---|
| W1 | omnetpp + lbm + mcf + perlbench + leslie3d + astar | W13 | gobmk + bwaves + astar + mcf + perlbench + omnetpp |
| W2 | gobmk + sjeng + GemsFDTD + povray + milc + cactusADM | W14 | leslie3d + namd + lbm + perlbench + omnetpp + mcf |
| W3 | bzip2 + gamess + calculix + xalancbmk + tonto + zeusmp | W15 | gromacs + bwaves + omnetpp + astar + mcf + perlbench |
| W4 | libquantum + h264ref + gcc + soplex + mcf + perlbench | W16 | povray + GemsFDTD + gromacs + bwaves + calculix + tonto |
| W5 | leslie3d + zeusmp + lbm + hmmer + povray + perlbench | W17 | omnetpp + povray + bwaves + gromacs + libquantum + calculix |
| W6 | gromacs + bwaves + leslie3d + zeusmp + lbm + perlbench | W18 | gobmk + omnetpp + povray + GemsFDTD + gromacs + bwaves |
| W7 | hmmer + libquantum + h264ref + povray + milc + cactusADM | W19 | bzip2 + gobmk + omnetpp + povray + GemsFDTD + gromacs |
| W8 | xalancbmk + soplex + GemsFDTD + povray + milc + perlbench | W20 | milc + bzip2 + gobmk + omnetpp + povray + GemsFDTD |
| W9 | GemsFDTD + povray + milc + omnetpp + mcf + perlbench | W21 | h264ref + bwaves + perlbench + zeusmp + mcf + milc |
| W10 | cactusADM + GemsFDTD + povray + milc + perlbench + omnetpp | W22 | leslie3d + h264ref + bwaves + perlbench + zeusmp + mcf |
| W11 | soplex + GemsFDTD + povray + milc + cactusADM + omnetpp | W23 | lbm + leslie3d + h264ref + bwaves + perlbench + zeusmp |
| W12 | gcc + soplex + omnetpp + astar + mcf + perlbench | W24 | soplex + lbm + leslie3d + h264ref + bwaves + perlbench |

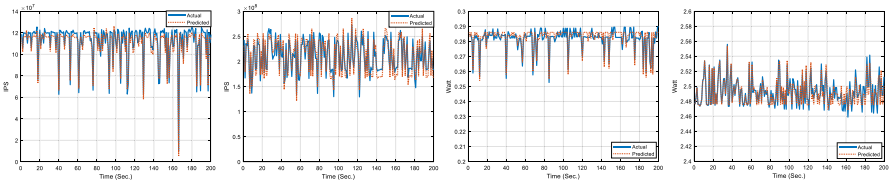


**Fig. 5** The predicted versus actual values of **a** $IPS_{\_little}$, **b** $IPS_{\_big}$, **c** $P_{\_little}$, and **d** $P_{\_big}$ for the random execution of SPEC benchmark

**Table 2** Performance and power prediction error

| | $IPS_{\_little}$ | $IPS_{\_big}$ | $P_{\_little}$ | $P_{\_big}$ |
|---|---|---|---|---|
| Performance prediction error (%) | 3.3 | 1.7 | 2.9 | 3.8 |

**Table 3** Specification of different schedulers

| Scheduler | Energy eff. Aware | Contention aware | Fairness aware |
|---|---|---|---|
| Linux | | | |
| DTPM [11] | ✓ | | |
| Minfair [19] | | | ✓ |
| HFEE [16] | ✓ | | ✓ |
| CAMPS [2] | | ✓ | ✓ |
| CEEF | ✓ | ✓ | ✓ |

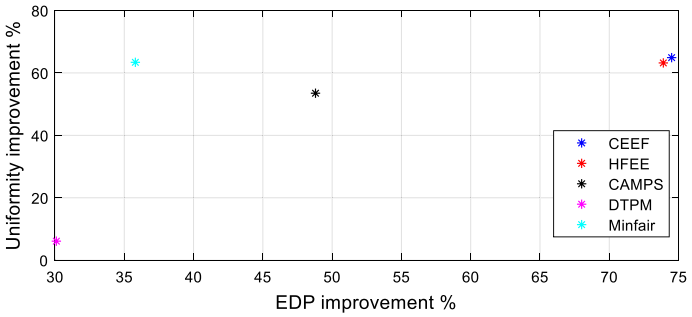


**Fig. 6** EDP and uniformity improvement in different schedulers compared to standard Linux schedulers

fairness management sections of the CEEF are repeated every one second (epoch duration). When a task execution completes, it is not relaunched and the number of tasks decreases until all of them terminate. The $W_{max}$ and $Uniformity_{\text{threshold}}$ are user-defined values which are specified before the start of scheduling. CEEF framework is an extension of our previous framework. Two significant changes have been

applied as the extensions: (1) online energy efficiency management; (2) performing contention administration.

### 5.3.1 CEEF versus other schedulers

Various schedulers, shown in Table 3, are implemented, and their results are compared to CEEF in terms of EDP and uniformity. Figure 6 exhibits the joint EDP and uniformity improvement ([uniformity of CEEF-uniformity of other scheduler]/uniformity of CEEF) of CEEF framework and various schedulers compared to standard Linux scheduler for all 24 selected workloads. As shown in Fig. 6, CEEF incorporates most improvement for EDP and uniformity simultaneously, among all other schedulers. CEEF framework improves EDP and uniformity by 74.5% and 64.9% compared to Linux standard scheduler. The second-best scheduler is HFEE, while EDP and uniformity improvements are 73.9% and 63.2%, respectively. The lack of contention management and static energy management in HFEE compared to CEEF causes less EDP and uniformity improvement. CAMPS and Minfair are unaware of energy efficiency that leads to less EDP improvement which are 48.8% and 35.8%, respectively. Uniformity improvement in CAMPS and Minfair compared to standard Linux schedulers is 53.5% and 63.4% correspondingly. DTPM is just energy-aware algorithm and does not consider uniformity and makespan. It has least EDP and uniformity improvement, 30.1% and 6.1% accordingly. The fully structural analysis of different schedulers is presented in the following subsections.

### 5.3.2 Clusters utilization

Big (little) cluster's utilization is defined as the ratio of execution time of a workload on big (little) cluster to the total execution time of both clusters. Big clusters' utilization and little clusters' utilization are crucial factors that affect the scheduler performance and EDP significantly. Higher little cluster's utilization increases makespan compared to higher big cluster's utilization. Figure 7 shows the big/little cluster' utilization of different algorithms under various workloads. CEEF exploits little cluster when the number of tasks is more than big core count and little cores are idle,

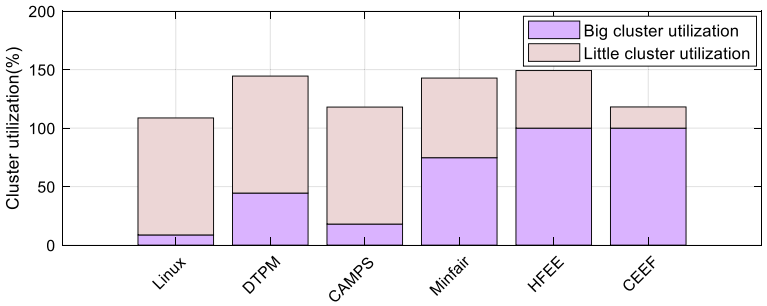


**Fig. 7** The average big and little cluster usage of different schedulers under various workloads

in case the task count is less than big core count. This reduces the execution time significantly.

HFEE has 31.2% more utilization of little cluster compared to CEEF, which is due to lack of contention management. Minfair maximizes fairness by equal utilization of big and little clusters with the cost of higher execution time. Both big and little clusters are always active in this policy even when the task count is lower than big core count, which degrades performance (execution time) dramatically. DTPM target is energy consumption management and does not consider fairness. It uses big cluster less than little cluster in order to reduce energy consumption, which results in longer execution time. Heterogeneity agnostic Linux standard scheduler uses little cluster more than big cluster and increases execution time consequently.

### 5.3.3 Clusters frequency

Another important characteristic of schedulers is the level of clusters frequency utilization which is defined as the period of time a specific frequency level of cluster is used that affects substantially performance, energy consumption, and uniformity. Using higher frequency level improves performance at the cost of higher energy consumption. Also, the frequency level of clusters affects the uniformity considerably.

According to previous studies [16], DVFS is performed on the big clusters and the frequency of little cluster remains unchanged. Figure 8 shows the frequency level of all schedulers under running *w3* workload. CEEF enhances uniformity via DVFS and even out cluster's computing power, which results in higher fairness. Also, frequency scaling is applied only at the state of task count more than big core count, otherwise all tasks are mapped to big cores, which improves performance (execution time decrement) and fairness significantly. As shown in Fig. 8, CEEF exploits low frequency levels compared to other schedulers (excluding DTPM that is just energy-aware and does not consider fairness). Lower execution time and energy consumption result in lower EDP compared to other schedulers. Close frequency-level utilization of CEEF and HFEE makes energy consumption of two schedulers rather equal. Little cluster utilization of HFEE is more than CEEF, so it has lower performance (more execution time) compared to CEEF; therefore, the EDP of HFEE

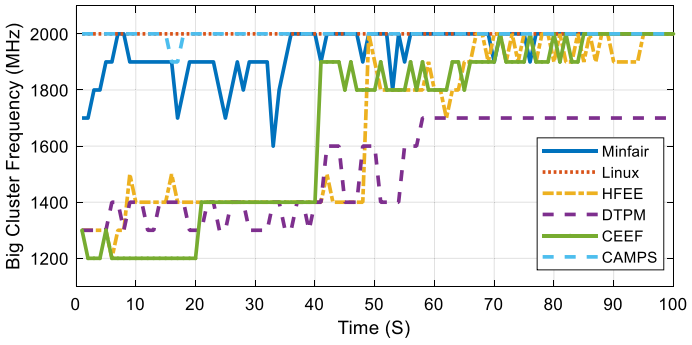


**Fig. 8** The frequency level of all schedulers under running w3 workload

is slightly more than that of CEEF. Dynamic *EEM* of CEEF versus static *EEM* of HFEE boosts CEEF uniformity about 1.7% compared to HFEE uniformity. DTPM as a fairness agnostic scheduler exploits low frequency levels and little cluster more than big cluster, which results in lower energy consumption and higher execution time and, hence, degrades EDP dramatically.

Min-Fair target is fairness improvement without considering DVFS and energy efficiency which results in higher energy consumption. High execution time due to both clusters' activation increases EDP substantially. CAMPS and Linux standard scheduler always operate at high frequency levels, which results in higher energy consumption. Little cluster utilization of these two schedulers is more than big cluster utilization that increases execution time. Their longer delay and higher energy consumption result in higher EDP. In order to investigate the CEEF overhead, the framework is operated at two different states of isolated and shared (alongside the workload set) and their performance is compared. The results indicate 1.95% overhead.

## 6 Conclusion and future work

In this paper, a scheduling framework consisting of energy efficiency, shared resource contention, and fairness management for heterogeneous multi-core processors is presented via frequency scaling support. The performance per watt ratio of big to little cluster which is predicted online with a highly accurate regression model plays a critical role in energy efficiency management. The presented shared resource contention administration boosts performance and fairness through scaling the frequency of big cores. The experimental results indicate that the proposed framework surpasses Linux and four other schedulers in terms of fairness and energy efficiency. Future work will be the presentation of performance, energy efficiency, and fairness models in order to estimate shared metric values via isolated values. The extension of FSM in order to support local DVFS is considered as the other further study.

## References

1. Zhuravlev S, Saez JC, Blagodurov S, Fedorova A, Prieto M (2012) Survey of scheduling techniques for addressing shared resources in multicore processors. ACM Comput Surv 45(1):1–28
2. Garcia-Garcia A, Saez JC, Prieto-Matias M (2018) Contention-Aware Fair Scheduling for Asymmetric Single-ISA Multicore Systems. IEEE Trans Comput 67(12):1703–1719
3. Srikantaiah S, Das R, Mishra AK, Das CR, Kandemir M (2009) A case for integrated processor-cache partitioning in chip multiprocessors. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp 1–12
4. Delaluz V, Sivasubramaniam A, Kandemir M, Vijaykrishnan N, Irwin MJ (2002) Scheduler-based DRAM energy management. In: Design Automation Conference (DAC), pp 697–702
5. Lukefahr A, Padmanabha S, Das R, Dreslinski Jr R, Wenisch TF, Mahlke S (2014) Heterogeneous microarchitectures trump voltage scaling for low-power cores. In: Parallel architectures and compilation Techniques (PACT), pp 237–250

6. Sarma S, Muck T, Bathen LA, Dutt N, Nicolau A (2015) SmartBalance: a sensing-driven linux load balancer for energy efficiency of heterogeneous MPSoCs. In: Design Automation Conference (DAC), pp 1–6
7. Mück TR, Ghaderi Z, Dutt ND, Bozorgzadeh E (2017) Exploiting heterogeneity for aging-aware load balancing in mobile platforms. IEEE Trans Multi-Scale Comput Syst 3(1):25–35
8. Van Craeynest K, Jaleel A, Eeckhout L, Narvaez P, Emer J (2012) Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In: International Symposium on Computer Architecture (ISCA), pp 213–224
9. Delimitrou C, Kozyrakis C (2013) Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In: Architectural support for programming languages and operating systems (ASPLOS), pp 77–88
10. Kim M, Kim K, Geraci JR, Hong S (2014) Utilization-aware load balancing for the energy efficient operation of the big. LITTLE processor. In: Design, Automation & Test in Europe (DATE), pp 1–4
11. Bhat G, Singla G, Unver AK, Ogras UY (2018) Algorithmic optimization of thermal and power management for heterogeneous mobile platforms. IEEE Trans Very Large Scale Integr (VLSI) Syst, 26(3): 544–557
12. Roy SK, Devaraj R, Sarkar A, Maji K, Sinha S (2020) Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems. J Syst Archit 105(1):101706
13. Feliu J, Sahuquillo J, Petit S, Duato J (2017) Perf&Fair: a progress-aware scheduler to enhance performance and fairness in SMT multicores. IEEE Trans Comput 66(5):905–911
14. Van Craeynest K, Akram S, Heirman W, Jaleel A, Eeckhout L (2013) Fairness-aware scheduling on single-ISA heterogeneous multi-cores. In: Parallel Architectures and Compilation Techniques (PACT), pp 177–187
15. Ankit T, Chaudhari K, Shah M (2020) A comprehensive survey on energy-efficient power management techniques. Procedia Comput Sci 167(1):1189–1199
16. Salami B, Noori H, Naghibzadeh M (2020) Fairness-aware energy efficient scheduling on heterogeneous multi-core processors. IEEE Trans Computs, pp 1–1
17. Li T, Brett P, Knauerhase R, Koufaty D, Reddy D, Hahn S (2010) Operating system support for overlapping-ISA heterogeneous multi-core architectures. In: High Performance Computer Architecture (HPCA), pp 1–12
18. Becchi M, Crowley P (2006) Dynamic thread assignment on heterogeneous multiprocessor architectures. In: Computing Frontiers (CF), pp 29–40
19. Kim C, Huh J (2018) Exploring the design space of fair scheduling supports for asymmetric multi-core systems. IEEE Trans Comput 67(8):1136–1152
20. Tian K, Jiang Y, and Shen X (2009) A study on optimally co-scheduling jobs of different lengths on chip multiprocessors. In: Computing Frontiers (CF), pp 41–50
21. Zhuravlev S, Blagodurov S, Fedorova A (2010) Addressing shared resource contention in multicore processors via scheduling. ACM Sigplan Notices 45(3):129–142
22. Moreno IS Yang R, Xu J, Wo T (2013) Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement. In: International Symposium on Autonomous Decentralized Systems (ISADS), pp 1–8
23. Kim YG, Kim M, Kong J, Chung SW (2020) An Adaptive Thermal Management Framework for Heterogeneous Multi-Core Processors. IEEE Trans Comput 69(6):894–906
24. Jain PN, Surve SK (2020) A review on shared resource contention in multicores and its mitigating techniques. Int J High Perform Syst Archit 9(1):20–48
25. da Silva J, Leao L, Petrucci V, Gamatié A, Pereira F (2020) Mapping computations in heterogeneous multicore systems with statistical regression on inputs. In: Brazilian Symposium on Computing Systems Engineering (SBESC)
26. Singh AK, Dey S, McDonald-Maier K, Basireddy KR, Merrett GV, Al-Hashimi BM (2020) Dynamic energy and thermal management of multi-core mobile platforms: A survey. IEEE Des Test 37(5):25–33
27. Pasricha S, Ayoub R, Kishinevsky M, Mandal SK, Ogras UY (2020) A survey on energy management for mobile and IoT devices. IEEE Des Test
28. Ortega C, Alvarez L, Casas M, Bertran R, Buyuktosunoglu A, Eichenberger AE, Bose P, Moreto M (2020) Intelligent adaptation of hardware knobs for improving performance and power consumption. IEEE Trans Comput
29. Rodgers JL, Nicewander WA (1988) Thirteen ways to look at the correlation coefficient. Am Stat 42(1):59–66

30. Petrucci V, Loques O, Mossé D, Melhem R, Gazala NA, Gobriel S (2015) Energy-efficient thread assignment optimization for heterogeneous multicore systems. ACM Trans Embedded Comput Syst TECS 14(1):1–26