Energy-Performance Co-Management of Mixed-Sensitivity Workloads on Heterogeneous Multi-core Systems

Elham Shamsa
Department of Future Technologies
Turku, Finland
elsham@utu.fi

Anil Kanduri
Department of Future Technologies
Turku, Finland
spakan@utu.fi

Amir M. Rahmani
Department of Computer Science and School of Nursing
Irvine, USA
a.rahmani@uci.edu

Pasi Liljeberg
Department of Future Technologies
Turku, Finland
pakrli@utu.fi

ABSTRACT

Satisfying performance of complex workload scenarios with respect to energy consumption on Heterogeneous Multi-core Platforms (HMPs) is challenging when considering i) the increasing variety of applications, and ii) the large space of resource management configurations. Existing run-time resource management approaches use online and offline learning to handle such complexity. However, they focus on one type of application, neglecting concurrent execution of mixed sensitivity workloads. In this work, we propose an energy-performance co-management method which prioritizes mixed type of applications at run-time, and searches in the configuration space to find the optimal configuration for each application which satisfies the performance requirements while saving energy. We evaluate our approach on a real Odroid XU3 platform over mixed-sensitivity embedded workloads. Experimental results show our approach provides 54% lower performance violation with 50% higher energy saving compared to the existing approaches.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *System on a chip.*

KEYWORDS

On-chip Resource allocation, Heterogeneous Multi-core Systems, Performance, latency, throughput, Concurrent applications

ACM Reference Format:
Elham Shamsa, Anil Kanduri, Amir M. Rahmani, and Pasi Liljeberg. 2021. Energy-Performance Co-Management of Mixed-Sensitivity Workloads on Heterogeneous Multi-core Systems. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3394885.3431516

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431516

1 INTRODUCTION

Run-time resource management in Heterogeneous Multi-core Platforms (HMP)s is critical, considering diversity in power-performance characteristics of cores and workload variation among concurrent applications [20, 23]. Applications with streaming inputs require a certain fixed level of performance within a strict deadline - these are *latency-sensitive applications*. Other applications processing batch inputs do not have any strict latency constraints, but require a higher overall performance - these are *throughput-driven applications*. Real workload scenarios in embedded processors often feature a combination of both latency-sensitive and throughput-driven applications running concurrently. For example, streaming video in a mobile processor, while uploading files to the cloud in the background represents both latency and throughput applications running concurrently. In this scenario, the deadline for latency applications have to be met (eg., target video frame rate), while ensuring higher performance for throughput applications (eg., file uploads as fast as possible). Prioritizing among these concurrent latency and throughput applications can lead to conflicting resource allocation decisions [19, 20]. On the other hand, embedded processors are largely constrained by power and energy budgets. In addition to managing performance, optimizing energy efficiency is another key criterion for resource management strategies. Core level heterogeneity makes HMPs efficient in handling performance requirements of concurrent mixed sensitivity workloads (latency and throughput), through appropriate application-core mapping [22]. Similarly, other resource actuation knobs such as dynamic voltage and frequency scaling (DVFS), Degree of Parallelism (DoP), and task migration (TM) support energy efficiency. However, managing mixed sensitivity workloads' requirements with the combination of the aforementioned knobs (heterogeneity, DVFS, DoP, TM) exposes a wide range of resource allocation choices. Pruning such a large Pareto-space at run-time for efficient resource allocation is an exhaustive process.

In this context, application characteristics can be an informative guide for resource allocation to satisfy applications' performance while optimizing energy consumption under mixed sensitivity workloads. Some of the existing approaches for energy and performance management ignore application characteristics, limiting their efficiency in adapting to various workloads[5, 8, 13]. The other approaches leverage offline profiling [11] or on-line learning

[14, 15, 18] for understanding applications characteristics. However, those ignore considering dynamic workload scenario with combination of multiple concurrent *latency* and *throughput* applications. The approaches in [10, 11] focuses on satisfying throughput-driven applications by maximizing performance per power, while the other approaches [3, 15] focus on satisfying latency applications. A real workload scenario contains different types of applications that enter and leave system dynamically, which may cause varying priorities for applications at run-time. Therefore, there is a critical need for an intelligent resource management approach which considers applications' requirements based on their performance requirement category and prioritizes them for efficient energy performance co-management. To handle this problem, we propose a holistic resource allocation approach which leverages offline characterization for online management. Offline characterization can efficiently reduce the computation overhead at run-time. Our framework i) monitors applications' requirements periodically at run-time, ii) prioritizes the applications based on their performance requirements and sensitivities to latency and throughput, iii) selects a suitable resource configuration which satisfies applications' requirements while saving energy. Our contributions in this paper are:

- Offline characterization and analysis of mixed-sensitivity applications from the Rodinia benchmark suite.
- On-line prioritizing of throughput and latency driven applications based on their requirements.
- Run-time resource management for satisfying mixed sensitivity workload's requirement while minimizing energy consumption by using an on-line progress predictor.
- Implementation and evaluation of the resource management method on a real heterogeneous Odroid XU3 platform on real embedded benchmark workload scenarios.

2 BACKGROUND AND MOTIVATION

In this section we discuss the challenges of resource management for mixed-sensitivity workloads and present a motivation examples, then we refer to the related works in this area.

2.1 Motivation

In this section, we highlight two challenges in resource management by considering mixed type workload scenarios:

(1) Prioritizing the latency and throughput applications in dynamic workload scenarios.
(2) Finding the best configuration which satisfies performance requirements of all the applications among a large space of configurations.

The first challenge is demonstrated through an example in Figure 1. We use `ParticleFilter` (*Pf*) and `Streamcluster` (*St*) application form Rodinia benchmark suite [7] which are throughput and latency applications, respectively. These applications arrive and leave the system over three different baselines viz., P1) higher priority for throughput application, P2) higher priority for latency application, P3) Intelligent prioritizing. The red dotted line shows a strict target for latency application and the blue dotted lines show the acceptable range of performance for throughput application. The higher performance than the upper bound leads to overusing and waste of energy and the lower leads to user dissatisfaction. P1 assigns

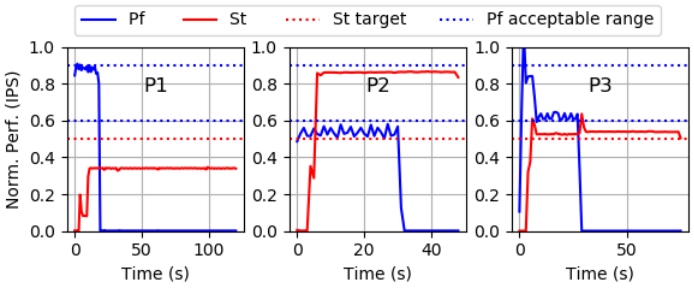

Figure 1: Normalized performance of *Particle Filter (Pf)* and *Stream Cluster (St)* running concurrently using 3 different policies.

higher priority to *Pf* and maps it to 4 big cores to provide higher performance, however, by mapping the *St* to 4 LITTLE cores misses the target latency of *St*. P2 assigns higher priority to *St* and maps it to 4 big cores while mapping *Pf* to 4 LITTLE cores which leads to overusing of *St* and missing the performance limit for *Pf*. However, an intelligent prioritizing (P3) which we use in our method, dynamically updates the priority at run-time. P3 prioritizes *Pf* and maps it to 4 big cores at first when *Pf* is running singular. When *St* arrives the priority of *Pf* decreases and *St* becomes prior. Then the resource management search for a reasonable mapping for both applications within the available configurations to find the best mapping in which the performance of both applications is satisfied while avoiding energy wasting. In this example, P3 maps *Pf* to 2 big cores and *St* to the 2 other big cores to reach the goal. Finding such configuration is the second challenge which is demonstrated through another example in Figure 2. In this example, we show the various execution times of *pf* and `Ferret` (Latency application) applications when running concurrently on different configurations of Odroid XU3 [12] platform. Such a platform provides 44 possible mapping configurations (1-4 big and LITTLE cores for each application) for running two applications concurrently. The red dashed line in the Figure 2 (b) shows strict deadline of *Ferret* which must be respected in resource management. Finding the best configuration which satisfies both applications requires an exhaustive search among all 44 configurations. The configurations which lead to violating *Ferret* deadline are not suitable (red circles). Within the remained configurations, the configuration which results in higher throughput for Pf (within the acceptable range) and at the same time meets the deadline for *Ferret* with the lowest energy consumption should be selected. In this example, the green point provides the highest throughput (lowest execution time) for Pf by mapping it to 4 big cores and meet the *Ferret* deadline by using 3 Little cores. Finding this optimal configuration by considering the DVFS knob is even more challenging and requires an intelligent resource management framework. In this paper, we propose a method which tackles the above-mentioned challenges.

2.2 Related Work

By increasing the complexity of workload scenarios in embedded systems, the existing resource management approaches characterize the applications using i) offline data collection [11, 16, 21], ii)

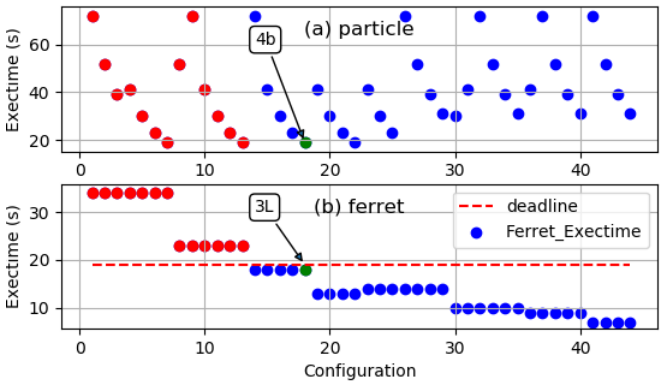

Figure 2: Execution time of Particle Filter and Ferret application through different configurations. (a) Particle Filter, (b) Ferret.

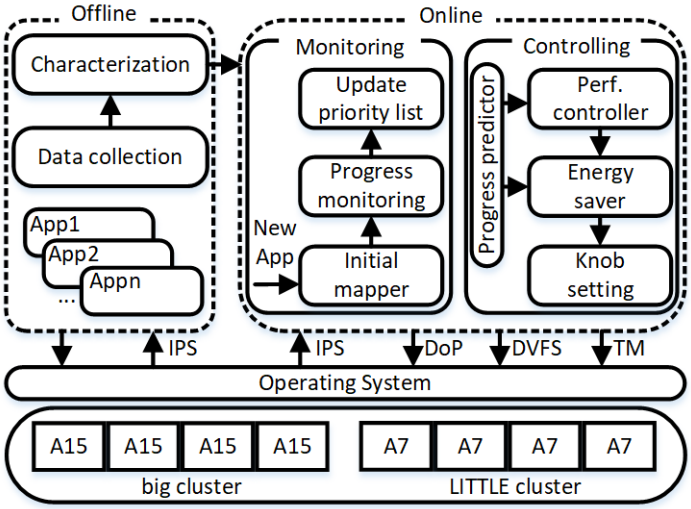

Figure 3: High level architecture of the proposed method.

online learning method [3, 15], or iii) combination of both [2, 14]. However, some resource management approaches ignore application characterization, limiting their efficiency for adapting various workload scenarios [5, 8, 13]. HMPs are well-suited platforms for handling complex workload scenarios by various controlling knobs such as DVFS, DoP for multi-thread applications, and TM to provide a suitable trade-off between energy and performance. Several existing approaches do not use the benefits of all controlling knobs and remain space for improving the resource management efficiency [8, 13]. Other approaches which use a combination of different controlling knobs focus on a single application workload and ignore the complexity of multiple concurrent applications [10, 11, 15]. Some of these techniques focus on satisfying throughput applications [9, 11], failing to meet deadlines for latency applications, while the others ignore throughput applications' requirements [15]. Although these approaches overlook the complexity of multiple applications, some other approaches do consider multiple applications scenario [1, 6, 18], but ignore the different types of applications in workload scenarios. These techniques consider applications with a specific requirement which is throughput or latency and try to satisfy that requirement for all the applications. In this work, we provide a run-time resource management framework which satisfies performance requirements of complex mixed type of workload scenarios by using combinations of all the controlling knobs in an HMP platform.

3 PROPOSED METHOD

In this section, we explain the components of our framework which is shown in Figure 3 and consists of i) offline characterization and ii) online monitoring and controlling. In the following, we explain each part in Figure 3 in details.

3.1 Offline characterization

When a multi-thread application runs on a heterogeneous platform, each resource allocation configuration e.g core type, DoP, and frequency can provide a different *progress* rate for that application. We characterize applications based on the variation of their progress rates in different configurations on the Odroid XU3 platform.

Data collection: We run various applications from Rodinia [7] and Parsec [4] benchmark suits on different configurations and collect Instruction-per-second (IPS) for each application during their executions. We consider various combinations of $< num\_cores >< core\_type >$ including 1b-4b, and 1L-4L, where b and L represent big and LITTLE cores and set the frequency to 1.4 GHz for the Little cluster and 1.8 GHz for the big cluster. The Odroid XU3 platform has 19 levels of frequency for the big cluster and 13 levels for the LITTLE cluster. Considering all combinations of frequencies, num_cores, and core_type makes a huge space of configurations even for offline analysis. Thus, we collect data for a fixed frequency and predict IPS for the other frequencies. Then, we use such data for encapsulating each application's characteristics and use that at run-time management.

Characterization: We define progress (*Prog*) of each application in a given time as the ratio of completed instructions at that time to the total instructions that the application should complete.

$$Prog = \frac{Acc\_IPS_t}{Total\_IPS}, \tag{1}$$

where Acc_IPS_t refers to accumulated IPS at time t, and $Total\_IPS$ shows total IPS that an application should complete in its execution. Based on the collected data, the progress rate is different for various applications in a fixed configuration, thus it can be a characteristic for an application. Similarly, the ratio of progress rate in two various configurations is different for a unique application. Therefore we define *Core_Speedup* and *Parallelism_speedup* for each application to leverage these two characteristics at run-time management.

$$Core\_Speedup = \frac{Prog_b}{Prog_L}, \tag{2}$$

where Core_Speedup is the gained speedup by changing cluster from LITTLE to big, $Prog_b$ and $Prog_L$ are the progress rate when the application runs on one big or one LITTLE core, respectively. Parallelism_Speedup which is the gained speed_up by increasing

the number of cores is formulated as follows.

$$Parallelism\_speedup = \frac{Prog_{n+1\_core}}{Prog_{n\_core}}, \quad (3)$$

where $Prog_{n+1}$ is the progress rate when the application runs on n+1 cores of the big or LITTLE cluster, and $Prog_n$ is the progress rate when the application runs on n cores of the same cluster. The analysis based on extensive experiments shows Parallelism_speedup for big cluster and LITTLE cluster are almost the same, thus, in our framework, we consider the average Parallelism_speedup of the big and LITTLE cluster as the characteristic for each application. *Core_Speedup* and *Parallelism_speedup* are two characteristics which we leverage at run-time for progress prediction and select the best configuration consequently.

Frequency_speedup is also required for progress prediction in various range of frequencies. The Frequency_speedup is not application-specific and for all the applications can calculate as follows:

$$Frequency\_speedup = \frac{freq_t + 100}{freq_t}, \quad (4)$$

where $freq_t$ is the frequency in epoch t.

3.2 Online monitoring and controlling

In the online phase, we monitor the progress rate of running applications periodically and prioritize the applications based on their types and requirements. Then, we control applications' performance, and energy consumption of the system by setting resource allocation.

3.2.1 Monitoring. Periodically monitors arriving and leaving of applications at run-time and updates running applications list. When new applications arrive, *Initial mapper* maps the applications to the free cores based on their types (latency or throughput). Then the framework monitors the progress rate of each running application and updates the priority list based on the current progress rate and the requirements.

Initial mapper: Figure 4 shows the workflow of handling latency and throughput applications. The first step is the initial mapping of new applications. When a new application arrives the Initial mapper checks the core status, then selects the initial core assignment for the new application. If the new application is a throughput application, it will be mapped to one free big core; otherwise, the application will be mapped to one LITTLE core. If all the cores are full, the core assignment will be changed to release at least one core. Then, the *monitoring* and *controlling* components adjust the new configuration to satisfy all the applications requirement.

Progress monitoring: The *progress monitoring* is called in every parametrizable epoch, and check the progress rate of each application in the running application list. *Progress monitoring* read Instruction-per-epoch (IPE) for each running application and consider it as the progress rate of that application.

Update priority list: To prevent conflicting actions between performance controller and energy saver, we define one specific priority list for each one: i) performance priority list, and ii) energy priority list. The progress rate of running applications is used for updating the priority lists. Each application is assumed to present a target IPE as its nominal requirement. For updating priority list, we define *Relative Progress rate (RP)* for latency and throughput application in Equation 5 and 6. As latency applications have strict deadline, the priority of such applications increase when progress rate is lower than the target rate. Although throughput applications do not have a strict deadline, executing slower than a specific limit is not acceptable for a user and faster than another limit is waste of energy. Thus, for defining the priority for throughput applications we consider lim_l as a lower limit and lim_u as an upper limit.

$$RP_L = \frac{Prog_t}{Prog_{target}}, \quad (5)$$

$$RP_T = \begin{cases} \frac{Prog_t}{lim_l} & \text{if } Prog_t < lim_l \\ 1 & \text{if } lim_l < Prog_t < lim_u \\ \frac{Prog_t}{lim_u} & \text{if } Prog_t > lim_u \end{cases} \quad (6)$$

Where RP_L and RP_T are RP for latency and throughput applications, respectively, $Prog_t$ is progress rate which is monitored at epoch t, and $Prog_{target}$ is the target progress rate. When $RP < 1$, the progress rate is low and the application fails to meet the deadline in latency applications or satisfy user requirements in throughput applications, thus the lower *RP* shows higher priority in the performance list. $RP = 1$ is the ideal RP which leads to applications satisfaction without any energy wasting. Therefore, when $RP = 1$, the priority is the lowest for both performance and energy lists. When $RP > 1$, the applications can meet their requirements, however, may waste energy consumption. Thus, we similarly define a priority list for energy saving. The higher *RP* for an application, when $RP > 1$ shows that application waste more energy and has higher priority in the energy priority list. As shown in Figure 4, performance priority list (L1) and energy priority list (L2) contains different applications at run-time. L1 and L2 are sent to the performance controller and energy saver for energy performance co-management. For preventing switching between performance controller and energy saver we consider a threshold for changing the mode to energy saver. For example, for creating energy priority list, when $1 < RP < 1.2$ the priority for applications is zero and when $RP > 1.2$ the application place in energy priority list.

3.2.2 Controlling. Receives updated priority lists and decide to control performance or save energy based on the value of RP. The *performance controller* and *energy saver* in *controlling* leverage progress predictor to find the best configuration, then send it to knob setting for applying on the platform.

Progress predictor: Uses Core_speedup and Parallelism_speedup as offline characteristics of applications as well as Frequency_speedup for prediction. The above-mentioned characteristics are profiled offline and stored for each application. The progress needs to be predicted in three cases, virtual actuation of i) DVFS, ii) DoP, and iii) TM. The performance controller and energy saver virtually actuates each of the aforementioned knobs by one step and use progress predictor for prediction. The performance controller virtually increases frequency or DoP by one step or move the application from the LITTLE cluster to the big. The predicted progress for performance controller ($Prog_{pred-perf}$) for each of the above actuation is estimated as:

$$Prog_{pred-perf} = Prog_t \times speedup, \quad (7)$$

where $Prog_t$ is the current progress rate, and speedup can be frequency_speedup, Parallelism_speedup, or Core_speedup based on

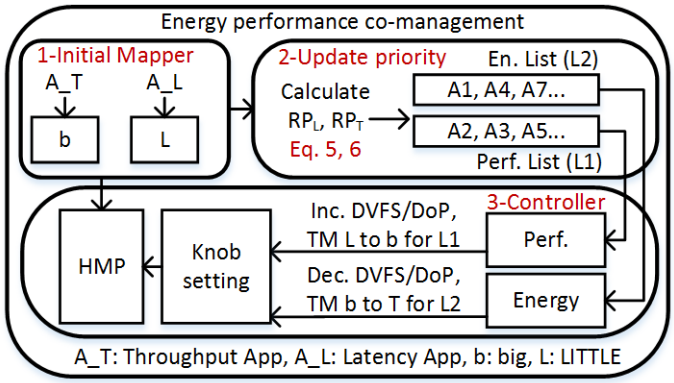

Figure 4: Workflow of handling latency and throughput applications

the action. In the other hand, the energy saver virtually decreases the level of frequency or DoP by one step or move application from big cluster to LITTLE, and estimates $Prog_{pred-en}$ as:

$$Prog_{pred-en} = Prog_t \times 1/speedup, \tag{8}$$

where $Prog_{pred-en}$ is progress rate for energy saver actuation.

Performance controller: Receives the priority list of applications which require performance enhancement and their RP is lower than 1. Performance controller searches for a suitable configuration in which applications' performance meet the requirement. Within the available actuation knobs, DVFS has the lowest run-time overhead compare to DoP and TM. Therefore, the performance controller starts with trying virtual DVFS and predicting progress in the higher levels of DVFS. The progresses are predicted using Equation 7 by virtually increasing the frequency level step by step e.g 100 Mhz, then, we calculate RP for the predicted progress (RP_{pred}) based on Equation 5 and 6. Instead of $Prog_t$ we replace predicted progress $Prog_{pred}$, then if the $RP >= 1$ for both latency and throughput applications the performance is satisfied, thus performance controller sends the desired frequency level to knob setting for applying. In the second step, if increasing the frequency level does not satisfy the performance requirement, we check the free cores and increase the level of parallelism. We predict progress for higher levels of parallelism step by step and stop when $RP >= 1$, then send that to the knob setting. Finally, if the predicted progress by increasing frequency and DoP does not provide target progress rate, the application will be migrated from LITTLE core to big core, or if the application is in the big core, we check the number of free LITTLE core if they are two times higher than free big cores, the application will be migrated to LITTLE cluster and DoP level will be increased. In the performance controller, we find the best knob setting based on prediction and then send it to *knob setting* module in Figure 3 for applying on the platform. Using prediction prevents run-time overhead for real applying of DVFS, Dop, and TM. Such a process applies to all the applications based on their priority.

Energy saver: After adjusting performance for all the applications among their priorities, the monitoring section updates the energy priority list and *energy saver* searches for suitable configuration to save more energy. Energy saver predicts progress rates by virtual actuation of DVFS, DoP, and TM. The energy saver virtually decreases the frequency and Dop step by step and estimates the progress rate for each step. When the relative progress rate for the predicted progress is near the target e.g. $1 =< RP_{pred} < 1.2$ for latency applications and $RP = 1$ for throughput applications, the configuration is desired. When energy saver virtually decreases the frequency of a cluster, the progress of all the applications on that cluster will change, thus we predict the progress for all the applications to prevent performance violation. After finding suitable frequency, *knob setting* component actuate DVFS on the platform, then frequency decreases, and *monitoring* updates the priority list. If still there is any application in the energy priority list, the energy saver continues to combine knobs for saving energy, otherwise saving more energy may lead to performance violation.

Knob setting: Apply the knob settings which are sent by the performance controller and the energy saver. The knob settings can be combinations of DoP, DVFS, and TM.

4 EVALUATION

In this section, we present the experimental setup and the evaluation of the proposed approach.

4.1 Experimental Setup

We evaluate our framework by running applications on the Hardkernel Odroid XU3 board with an HMP containing 4 big and 4 LITTLE cores [12]. The LITTLE cores operate within 0.2 to 1.4 GHz and provide energy efficiency, while the big cores operate in 0.2 to 2 GHz for high performance. The frequency of each cluster is adjustable by using DVFS which is provided in the platform. We measure the performance in terms of IPS by reading operating system counters. We monitor power by reading on-board power sensors periodically and estimate the energy consumption of the framework by integrating the measured power over time. Our framework is implemented as a Linux user-space process on top of the MARS framework [17] and is invoked periodically at run-time. We designed the monitoring and controlling epochs to be parameterizable. For experimentation, we set it to 1s in our setup.

4.2 Experimental Results

For evaluation purposes, we use 6 embedded systems applications from Rodinia benchmark suite [7]. We create a workload with mixed sensitivity of latency and throughput applications. The created workload contains at least 2 concurrent latency and throughput applications. Since there is no work which considers mixed type applications, we compare our method with DyPO [11] and AdaMD [3] which are recent state-of-the-art resource management approaches for optimizing performance and energy on HMP platform. We compare the instantaneous performance of each application in the created workload over Dypo, AdaMD, and our framework in Figure 5. The dotted lines show the target deadline for latency applications and acceptable performance limits for throughput applications. App2, App3, App6, and App9 are throughput applications which are concurrently running with the other latency applications. At first, when App1 and App2 are executing concurrently, AdaMD assigns higher priority to App2 and leads to missing the deadline for

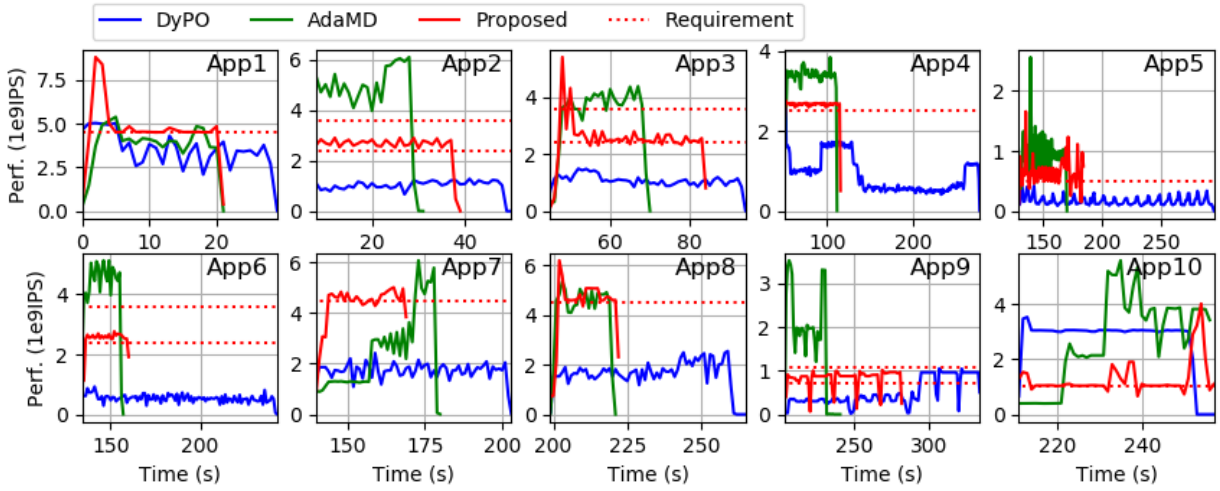

Figure 5: Performance comparison of latency and throughput applications in a dynamic workload scenario at run-time. App1,7,8 : Heartwall, App2,3,6 : Particle filter, App4 : Streamcluster, App5 : Srad, App9 : bfs, App10 : Leukocyte tracking.

Table 1: Comparison of our method with existing approaches.

Objective	DyPO	AdaMD	Proposed
Energy saving	45%	11%	22%
Perf. violation	91%	30%	14%

APP1 while APP2 is overusing. DyPO focuses on maximizing performance per energy for each singular application, thus maps both applications to their best configuration. Therefore, App1 meets the deadline until 6s, but when APP2 arrives, both application miss their requirement because of their competition between the resources. However, our method can satisfy both applications' requirement while saving energy by keeping App2 usage within the lower limit of its requirements. Similarly, AdaMD provides overusing for App3-10 except for App8, and Dypo could not handle multiple concurrent applications which causes performance violation. Our framework provides performance satisfaction for all the applications while preventing energy wasting. Although the behavior of applications is different, for example, App5 and App9 have a periodic behavior, our method can handle the performance to be in the acceptable range most of the time. Table 1 shows a comparison of energy saving and performance violation between DyPO, AdamD, and our framework. The presented energy saving is in comparison to Ondemand Linux governor which is using in several smartphones as a resource manager[18]. Our method provides the lowest performance violation while saving 22% energy which is 50% higher than AdaMD. Although Dypo has higher energy saving, it misses the performance requirements most of the time. We demonstrate the functionality of our framework in Figure 6 and 7 for better evaluation. Figure 6 shows each application from APP1-10 (y-axis) is mapped to which cores during run-time. The zero in the y-axis shows the core is empty and the other numbers show the App with that number is assigned to one of the cores which are shown in

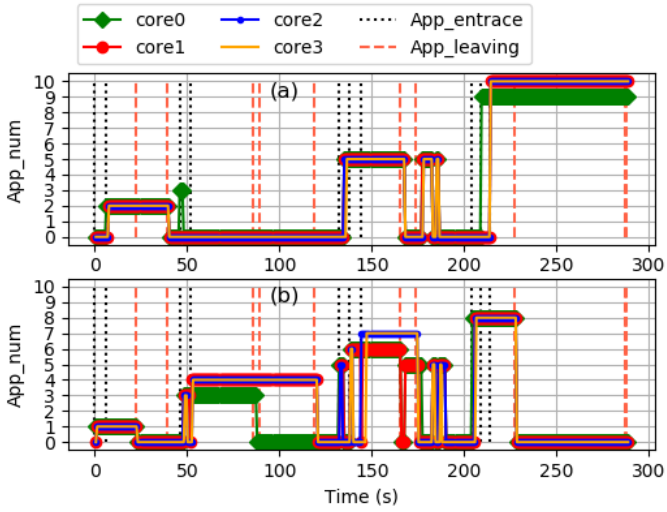

Figure 6: Mapping of Application 1-10 (Y-axis) to big and LITTLE cores at run-time. (a) Applications mapping to LITTLE cores, (b) Applications mapping to big cores.

different colors. Each solid line in Figure 6 (a) shows one of the LITTLE cores and in Figure 6 (b) one of the big cores. The dotted black lines show the applications' arrival times and the dashed orange lines show the completion of applications. In each arrival and completion of applications, the mapping configuration change to optimize performance and energy. When App1 arrives, it is mapped to big core0-3, as shown in Figure 6 (b). At t=6s, when App2 arrives, it is mapped to LITTLE core0-3. Before 50s, App2 finish and free the LITTLE cores, then App3 arrives, and it is mapped to one LITTLE core because it is a throughput application, but then it migrates to 1 big core since the performance was not satisfied by the LITTLE core. The App4 is assigned to 3 other big cores, and the LITTLE

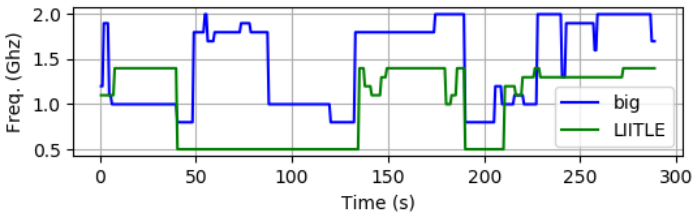

Figure 7: Frequency variation of big and LITTLE cluster during the experiment.

cores remain free until App5 arrival. App5 is a latency application and it is mapped to one big core at first by the *initial mapper*, then it migrates to 4 LITTLE cores. Similarly, APP6 is assigned to one big at first, then 2 big cores. By arriving App7, it is mapped to the remained big cores. The configuration of App5 is changed dynamically between big and LITTLE cores while concurrently running with App6 and App7. After completion of App5,6, and 7, App8 arrives and it is mapped to 4 big cores, then App9 uses one LITTLE core and App10 uses the other 3 LITTLE cores. In addition to the dynamic application to core mapping, the frequency of each cluster is also adjusted at run-time, which is presented in Figure 7. As we explained in Section 3.2.2, after initial mapping, the *performance controller* and the *energy saver* predict the suitable frequency and send that to the knob setting for applying DVFS. When the clusters are free, the frequency of the big cluster is set to 0.8 GHz and the LITTLE cluster is set to 0.5 GHz; otherwise, the frequency changes to the best match for satisfying the applications' requirements. To this end, our framework by determining mapping configuration at run-time combining with DVFS provides applications satisfaction and energy saving as shown in Figure 5 and Table 1.

5 CONCLUSIONS

We propose a resource management approach for satisfying the performance requirements of mixed-sensitivity workload scenarios while optimizing energy consumption. We calculate parallelism and core speed_up as applications characteristic by offline data collection. Then, we use such characteristics to predict the application's progress in various mapping configurations on an HMP platform. We monitor the applications at run-time and prioritize their requirements based on their types and characteristics, then find the optimal configuration by using the performance controller and the energy saver. We evaluated the proposed approach on the real HMP platform, and the results show lower performance violation with optimal energy saving compare to the existing approaches.

ACKNOWLEDGMENT

We acknowledge financial support by the University of Turku Graduate School and Academy of Finland project ACTER.

REFERENCES

[1] Aalsaud, A., Rafiev, A., Xia, F., Shafik, R., and Yakovlev, A. Model-free runtime management of concurrent workloads for energy-efficient many-core heterogeneous systems. In *2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)* (2018), IEEE, pp. 206–213.

[2] Al-hayanni, M. A., Rafiev, A., Xia, F., Shafik, R. A., Romanovsky, A., and Yakovlev, A. Parma: Parallelization-aware run-time management forenergy-efficient many-core systems. *IEEE Transactions on Computers* (2020).

[3] Basireddy, K. R., Singh, A. K., Al-Hashimi, B. M., and Merrett, G. V. Adamd: Adaptive mapping and dvfs for energy-efficient heterogeneous multi-cores. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019).

[4] Bienia, C., Kumar, S., Singh, J. P., and Li, K. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques* (2008), pp. 72–81.

[5] Brocanelli, M., and Wang, X. Surf: Supervisory control of user-perceived performance for mobile device energy savings. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (2018), IEEE, pp. 511–522.

[6] Brocanelli, M., and Wang, X. Supervisory performance control of concurrent mobile apps for energy efficiency. *IEEE Transactions on Mobile Computing* (2019).

[7] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on workload characterization (IISWC)* (2009), Ieee, pp. 44–54.

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

[9] Gupta, U., Babu, M., Ayoub, R., Kishinevsky, M., Paterna, F., and Ogras, U. Y. Staff: online learning with stabilized adaptive forgetting factor and feature selection algorithm. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)* (2018), IEEE, pp. 1–6.

[10] Gupta, U., Mandal, S. K., Mao, M., Chakrabarti, C., and Ogras, U. Y. A deep q-learning approach for dynamic management of heterogeneous processors. *IEEE Computer Architecture Letters 18*, 1 (2019), 14–17.

[11] Gupta, U., Patil, C. A., Bhat, G., Mishra, P., and Ogras, U. Y. Dypo: Dynamic pareto-optimal configuration selection for heterogeneous mpsocs. *ACM Transactions on Embedded Computing Systems (TECS) 16*, 5s (2017), 1–20.

[12] Hardkernel. ODROID-XU.

[13] Kanduri, A., Miele, A., Rahmani, A. M., Liljeberg, P., Bolchini, C., and Dutt, N. Approximation-aware coordinated power/performance management for heterogeneous multi-cores. In *Proceedings of the 55th Annual Design Automation Conference* (2018), pp. 1–6.

[14] Mandal, S. K., Bhat, G., Patil, C. A., Doppa, J. R., Pande, P. P., and Ogras, U. Y. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27*, 12 (2019), 2842–2854.

[15] Mishra, N., Imes, C., Lafferty, J. D., and Hoffmann, H. Caloree: Learning control for predictable latency and low energy. *ACM SIGPLAN Notices 53*, 2 (2018), 184–198.

[16] Mishra, N., Zhang, H., Lafferty, J. D., and Hoffmann, H. A probabilistic graphical model-based approach for minimizing energy under performance constraints. *ACM SIGARCH Computer Architecture News 43*, 1 (2015), 267–281.

[17] Muck, T., et al. Adaptive-reflective middleware for power and energy management in many-core heterogeneous systems. *Many Core Computing: Hardware and Software, IET* (2019).

[18] Reddy, B. K., Merrett, G. V., Al-Hashimi, B. M., and Singh, A. K. Online concurrent workload classification for multi-core energy management. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2018), IEEE, pp. 621–624.

[19] Shamsa, E., Kanduri, A., Rahmani, A. M., Liljeberg, P., Jantsch, A., and Dutt, N. Goal formulation: Abstracting dynamic objectives for efficient on-chip resource allocation. In *2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)* (2018), IEEE, pp. 1–4.

[20] Shamsa, E., Kanduri, A., Rahmani, A. M., Liljeberg, P., Jantsch, A., and Dutt, N. Goal-driven autonomy for efficient on-chip resource management: Transforming objectives to goals. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2019), IEEE, pp. 1397–1402.

[21] Singh, A., Basireddy, K. R., Prakash, A., Merrett, G., and Al-Hashimi, B. M. Collaborative adaptation for energy-efficient heterogeneous mobile socs. *IEEE Transactions on Computers* (2019).

[22] Xiang, Y., and Pasricha, S. Mixed-criticality scheduling on heterogeneous multicore systems powered by energy harvesting. *Integration 61* (2018), 114–124.

[23] Zhu, H., and Erez, M. Dirigent: Enforcing qos for latency-critical tasks on shared multicore systems. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (2016), pp. 33–47.