Concurrent Application Bias Scheduling for Energy Efficiency of Heterogeneous Multi-Core platforms

Elham Shamsa, *Student Member, IEEE*, Anil Kanduri, *Member, IEEE*,
Pasi Liljeberg, *Member, IEEE*, Amir M. Rahmani, *Senior Member, IEEE*

Abstract—Minimizing energy consumption of concurrent applications on heterogeneous multi-core platforms is challenging given the diversity in energy-performance profiles of both the applications and hardware. Adaptive learning techniques made the exhaustive Pareto-optimal space exploration practically feasible to identify an energy efficient configuration. Existing approaches consider a single application's characteristic for optimizing energy consumption. However, an optimal configuration for a given application in isolation may not be optimal when other applications are run concurrently. Approaches that consider concurrent application scenarios overlook the weight of total energy consumption per application, restricting them from prioritizing among applications. We address this limitation by considering the mutual effect of concurrent applications on system wide energy consumption to adapt resource configuration at run-time. We characterize each application's power-performance profile as a weighted bias through off-line profiling. We infer this model combined with an on-line predictive strategy to make resource allocation decisions for minimizing energy consumption while honoring performance requirements. The proposed strategy is implemented as a user-space process and evaluated on a heterogeneous hardware platform of Odroid XU3 over the Rodinia benchmark suite. Experimental results show up to 61% of energy saving compared to the standard baseline of Linux governors and up to 27% of energy gain compared to state-of-the-art adaptive learning-based resource management techniques.

Index Terms—On-chip Resource allocation, Heterogeneous Multi-core Systems, Energy, Concurrent applications

1 Introduction

Embedded heterogeneous multi-core platforms (HMP) require intelligent resource allocation strategies to achieve energy efficiency while sustaining performance requirements. At an *application level*, different applications running concurrently lead to significant diversity in workload characteristics at run-time. At the *hardware level*, different actuation knobs such as degree of parallelism (DoP), dynamic voltage-frequency scaling (DVFS), and the type of active cores among a heterogeneous set expose a wide range of performance-energy trade-offs [1], [11], [15], [24]. Both these put together exacerbates the challenge of understanding application requirements, followed by allocating and scaling system resources to co-optimize performance and energy efficiency. Further, the energy consumption profile of an application under different resource configurations varies based on the application's characteristics. Operating trivially at lower frequency levels and fewer active cores for lower energy consumption degrades the performance significantly. Despite the abundance of resource configuration choices to minimize energy consumption, the variation and diversity among different applications make it an exhaustive exploration [10], [18]. Further, on-line selection of resource configuration becomes even harder, considering an unknown sequence of concurrent applications, workload variation, inter-application interference and resource contention [1], [22]. Further, on-line selection of resource configuration becomes even harder, considering an unknown sequence of concurrent applications, workload variation, inter-application interference, and resource contention [1], [22].

Encapsulation of applications' characteristics as a metric called *bias* (which is defined in Section3) enables making a relevant choice of resource configuration for minimizing energy consumption while respecting performance target [10], [12], [18]. Existing resource management approaches use off-line profiling to characterize the applications at design time and use this information at run-time to avoid exhaustive search [10], [16], [21]. Other approaches use on-line prediction strategies, their efficiency and accuracy depend on the amount of power-performance statistics collected over a significant period of execution time [1], [18]. Both off-line and on-line approaches are confined to extract an optimal resource configuration of a single application running in isolation [3], [10], [12], [18], and are thus not readily adaptable to multiple concurrent application scenarios. While some approaches [21], [22] consider multiple applications, those i) overlook the combination of all the existing knobs (DoP, DVFS, and core selection) in heterogeneous platforms [4], [21], [22], [26], ii) ignore dynamic workload scenarios where applications arrive and leave the system in an unknown manner, limiting their efficiency and adaptability in optimizing resource allocation, and iii) do not consider the weight of total energy consumption per application, restricting those from prioritizing among applications [1], [22], [26]. This subsequently limits the significance of energy gains that can be achieved, since an optimal resource management configuration for one application (obtained from off-line/on-line profiling) which results in lower energy could be non-optimal when another application arrives for

• *Elham Shamsa, Anil Kanduri and Pasi Liljeberg are with University of Turku, 20500 Turku, Finland. Amir M. Rahmani are with University of California, Irvine, CA 92617 USA.*
E-mail: elsham@utu.fi, spakan@utu.fi, pakrli@utu.fi, a.rahmani@uci.edu

concurrent execution. This necessitates a run-time resource management strategy for understanding the applications' characteristics under concurrent workload scenarios.

An efficient way of adapting to varying workloads is to profile applications' power-performance metrics, combined with the amount of energy representing a weighted characterization. In this work, we present such weighted characterization, which can be leveraged at run-time to prioritize among a wide range of workload, containing memory and compute intensive applications and re-evaluate the optimal configuration. We consider DoP, DVFS, and core selection combinatorially, which creates a large space of knob setting. Searching among such space can make run-time management inefficient [4]. Thus, we shift profiling and model generation to off-line to reduce the overhead of run-time computation. We generate a bias model combining with a performance model and use that at run-time for on-line management. Moreover, we have an on-line performance controller to manage resource contention and application interference at run-time. Our contributions are:

- Off-line characterization and modeling of weighted bias and performance for single and multi-threaded applications.
- On-line management of weighted biases for prioritizing among concurrent applications, and selecting optimal resource configuration to minimize the total energy consumption of multi-programmed workloads.
- Combine off-line characterization and on-line controller to design a resource management framework for HMPs, which can handle complexity of concurrent workload scenarios.
- Implementation and evaluation of the resource management strategy on a heterogeneous hardware platform of Odroid XU3, over embedded benchmark workloads.

The rest of this paper is organized as follows. Section 2 presents the background and motivation of the problem. Section 3 describes the proposed method, including off-line characterization and on-line management. Section 4 presents the experimental setup, discusses, and evaluation of our framework. Finally, Section 5 concludes the paper.

2 Background and Motivation

In this section, we present the significance of concurrent applications' resource allocation and summarize relevant existing approaches, along with their limitations.

2.1 Motivation

We demonstrate the challenges of resource management for multiple concurrent applications through an example, considering a workload scenario with `ParticleFilter` running and `StreamCluster` arriving during *ParticleFilter*'s execution. In this work, we use Odroid XU3 [14], which consists of 4 ARM A15 big cores and 4 ARM A7 LITTLE cores operating in various ranges of frequencies. We consider 6 different configurations for each application, which are comprehensive enough for showing the complexity of resource management. The configurations present as nT, where n shows the number of cores assigned to the application, and T is the type of cores that can be big (b) or LITTLE (L). Figure 1 (a) and (b) show the normalized energy consumption of *Particlefilter* and *StreamCluster* when executing individually at fixed frequency 1.8 GHz for *big* and 1.4 GHz for *LITTLE* cluster, using 6 different configurations. The red and orange points show the first and second optimum configurations in terms of energy consumption. For both applications, mapping to 4L is the best configuration when run individually - therefore, both the applications can be mapped to 4L. However, under concurrent workload scenarios, the second application arrives when the first application is already running on the best configuration i.e., 4L. One solution is to map both applications to 4L, sharing the resources - since it is the known best configuration when run individually [10]. Despite the approach in [10] uses this solution, the configuration that provides the best energy consumption for each individual application may not be suitable for running those concurrently. The other solution is to map the second application to the second best configuration, which is 4b. Figure 1 (c) shows normalized energy consumption of executing *Particlefilter* and *Streamcluster* concurrently by considering different combinations of mapping. The x-axis shows different combinations of configurations as (nT_1, mT_2), where nT_1 refers to mapping configuration of *Particlefilter* and mT_2 refers to *Streamcluster*'s configuration. The purple and green points in Figure 1

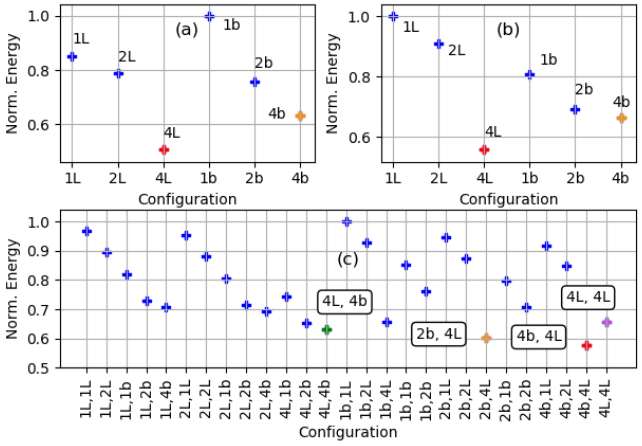

Fig. 1: Energy consumption of (a) Particlefilter, (b) Streamcluster, (c) ParticleFilter and Streamcluster concurrently. Each application run in 6 configurations - 1, 2, 4 big (b) and LITTLE (L) cores.

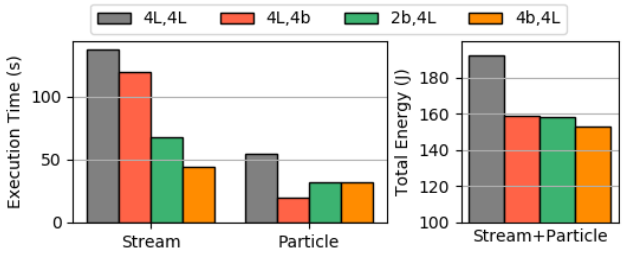

Fig. 2: Energy and execution time of Particlefilter and StreamCluster run concurrently in different configurations. L,b -Little and big cores

TABLE 1: Summary of the most recent existing works against the proposed method

Tech.	Controlling knob			Workloads			Methodology		Decision by		
	DVFS	DoP	TM	Mul-th	Mul-app	Dyn.	Off-line	On-line	Perf req.	Bias	WBias
PARMA [3]	✓	✓	✗	✓	✗	✗	✓	✓	✗	✗	✗
Imitation [16]	✓	✓	✗	✓	✗	✗	✓	✓	✓	✗	✗
Online Conc. [22]	✓	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗
Inter-cluster [21]	✓	✓	✗	✓	✓	✗	✓	✓	✓	✗	✗
Power-aware [2]	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗
Projection [26]	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓	✗
DyPO [10]	✓	✓	✗	✓	✗	✗	✓	✗	✗	✓	✗
AdaMD [4]	✓	✓	✓	✓	✓	✓	✗	✓	✓	✗	✗
Proposed approach	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

(c) show energy consumption of the suggested solutions, which are not optimal. On the other hand, it should be noted that the best configuration (red point) in fact is 4b for the first application and 4L for the second application, which requires migrations of the first application from LITTLE cores to big cores. Even the second best configuration (orange point) requires migration of the first application from 4L to 2b. The comparison of the energy consumption and execution time of these two applications (*Particlefilter* and *Streamcluster*) running concurrently, for four different configurations viz., "4L, 4L", "4L, 4b", "2b, 4L", and "4b, 4L" is shown in Figure 2. Mapping both of the applications to "4L" leads to the highest execution time and energy consumption, while "4b, 4L" results in the lowest execution time and energy consumption. Thus, combination of optimal configurations for each individual application i.e., 4L is not optimal when two applications are executing concurrently. Finding this optimal configuration can be challenging for different types of applications due to a large number of possible combinations for various configurations. Thus, in this paper, we aim to optimize energy consumption over concurrent workload scenarios without exhaustive search at run-time.

2.2 Related Work

The resource management approaches leverage machine learning methods either offline or online for optimizing energy consumption on embedded systems [9], [10], [12], [17], [18]. On-line training [18] leads to exhaustive search at run-time to find the optimal resource management for various range of applications. On the other hand, off-line profiling without on-line prediction and run-time monitoring is not comprehensive enough for unknown workloads. The techniques in [10], [12] use the off-line characterization of applications as performance-per-power - called *bias* in this content- for guiding resource management for a single application, however, such approaches ignore concurrent and dynamic workload scenarios and amount of consumed energy for each application as a *weight*. We propose a method that uses off-line profiling as *weighted bias* (which is defined in Section 3) combining with on-line prediction and run-time monitoring while considering dynamic multi-programmed workloads. Table 1 comparatively summarizes the most recent and relevant state-of-the-art approaches as per controlling knobs, ability to handle dynamic multi-programmed workloads, methodology, and decision strategies that they used. The frameworks presented in [3], [10], [21], [22], [26] cannot handle multi-thread applications and/or combinational optimization of different knobs, restricting their scope. The approach in [16] does consider multi-thread applications and combinational knobs setting, however, this approach neglects multi-application and dynamic workload scenarios. Power-aware [2], Projection [26], and DyPO [10] consider the application bias for guiding resource allocation, which we also consider in this work. Thus, we select DyPO [10] for comparison against our work to show the effect of using weighted bias for multi-application workload scenarios. DyPO uses exhaustive off-line profiling to find an optimal resource allocation for one application. AdaMD [4] is the most similar work to our approach, which considers multi-applications dynamic workload with combinational knobs setting, however, it uses a different decision strategy that overlooks the effect of the total energy consumption per application as a weight. Our proposed approach (shown in the last line of Table 1) presents a more comprehensive method compared to the others, addressing energy optimization of multi-application and dynamic workloads scenarios, considering weighted application bias. The experimental results in Section 4 show the efficiency of our work over DyPO [10] and AdaMD [4] which are the most relevant approaches.

3 Proposed Method

In this section, we present our resource management framework and the approach for minimizing the energy consumption of multi-application workload scenarios. Figure 3 shows the high level architecture of our methodology consisting of two phases viz., off-line characterization - to generate applications' model, and on-line management - to infer the off-line model for energy minimization and performance guarantee. We profile different applications on the target heterogeneous multi-core platform to extract applications' characteristics and generate bias and performance models. In the on-line phase, we use learned models as predictors to extract applications' characteristics i.e., applications' bias. The *On-line Controller* chooses an appropriate resource configuration that minimizes the energy consumption, and then fine-tunes the allocation decisions to fit the application's performance requirements as Instruction Per Second (IPS). When multiple applications are running concurrently, the Controller combines the resource configuration space of each running application and prunes it to re-evaluate the suitable resource configuration in a co-optimization manner. While this may result in altering the resource configuration

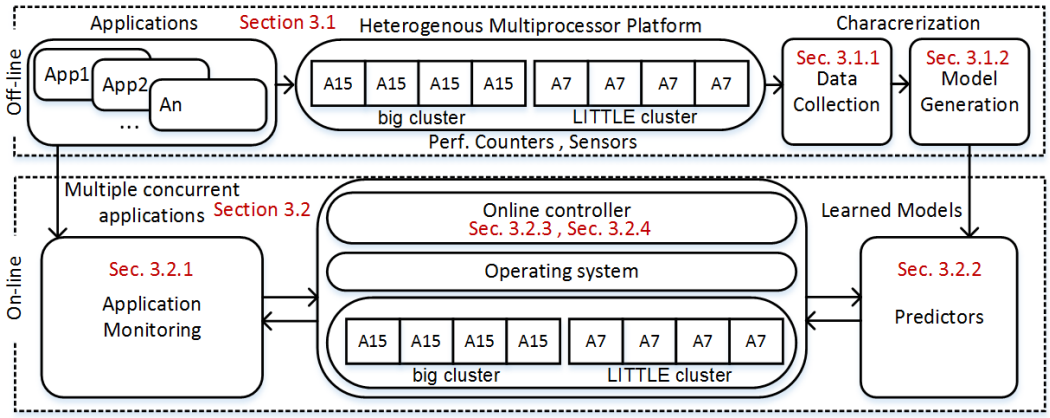

Fig. 3: High level architecture of the proposed method.

of already running application(s) from an optimal point, the total energy consumption of all the applications running will be minimized. However, most of the existing approaches neglect the possibility of task migration by arriving a new application, restricting their efficiency in adapting to multi-application dynamic workload scenarios.

3.1 Off-line Characterization

Given a multi-threaded application running on a heterogeneous platform, each core-type, DoP, and frequency level can provide a different performance-per-power. We define bias for each pair of application-configuration as the ratio of performance-per-power when an application runs on that configuration to the best performance-per-power. Bias for each application running on each configuration may be exhibited by applications based on their characteristics. Such characteristics can be IPS, number of threads, CPU-intensity or memory-intensity of the application, and execution time of the application. Understanding these biases for various applications can enable effective energy optimization decisions at run-time, as the relative value of the bias can guide prioritization among multiple applications. In addition to these, the actual amount of energy consumed by an application shows the significance of energy gains that can be achieved by choosing a specific configuration. Specifically, in the context of multi-programmed workloads, considering the amount of energy consumption of an application lends an insightful weightage to the application's bias. We use *weighted bias* as the combination of *application bias* and *amount of energy* to guide resource configuration decisions. *Weighted bias* provides a better characteristic of applications in terms of relative energy consumption, which is essential for handling multiple concurrent applications. Thus, by considering the weight of energy consumption for each application, we have a better estimation of total energy consumption over different configurations and workload scenarios. However, pruning a large resource configuration space to learn applications' biases at run-time leads to an exhaustive search making it impractical for on-line decision making [10]. Hence, we use an off-line profiling method to model and characterize applications' bias, which will be inferred on-line for energy minimization decisions.

Application Bias: We define the application bias (AB) as a list of configuration-biases for possible configurations in a platform. We assume the heterogeneous platform to have a maximum of N cores, composed of core types b and L, representing typical big and LITTLE cores, and M and K various frequency levels for each core type. We use $(Perf)_c$ and $(Pow)_c$ to represent performance and power consumption when running an application on configuration c. We represent the configuration-bias as:

$$CB_c = \frac{(Perf)_c/(Pow)_c}{(Perf)_{Bc}/(Pow)_{Bc}} \quad (1)$$

where CB_c shows the configuration bias of an application when runs on configuration c, $(Perf)_{Bc}$ and $(Pow)_{Bc}$ are performance and power consumption on the best configuration with minimum energy consumption for that application. The computation of an application may consist of different phases, such as CPU-intensive and memory-intensive. Such phase variation leads to various performance and power consumption for the application during its execution time. In our test case (Rodinia benchmark suite), the phase changes of applications are not significant, thus, we calculate CB_c for the phase which is dominant in total energy consumption (i.e., CPU-intensive phase). A higher CB_c shows configuration c is more proper for the application in terms of energy consumption. When two applications are competing for resources the application with a higher CB_c has higher priority to assign to the configuration c.

Weighted Bias: The bias of each application-configuration multiplied by the weight of energy consumption of that application represents the weighted-bias (WB) of the application in the given resource configuration as:

$$WB = W_A \times CB_c \quad (2)$$

The weight of energy consumption W_A for each application is calculated as:

$$W_A = \frac{Average(E_i)}{E_t} \quad (3)$$

where E_i represents energy consumption of the application in 6 major configurations (Explained in Section 3.1.1) of the platform, and E_t is the total energy of the system (i.e., battery capacity). Upon estimating different weighted

biases, the application with higher weighted bias is chosen as the priority.

3.1.1 Data Collection

We outline six major configurations that cover different core types and degrees of parallelism including 1, 2, and 4 threads running on big (b) and LITTLE (L) clusters. These configurations are represented as $< num\_cores >< core\_type >$, including 1L, 2L, 4L, 1b, 2b and 4b. These 6 configurations are comprehensive enough to show the performance-per-power characteristic of an application and provide optimal bias and performance prediction in our platform. For each of these configurations, we fix the frequency levels at 1.8 GHz for the big cluster and 1.4 GHz for the LITTLE cluster. Although we collect data in a platform with 2 types of core, for the other platforms with a higher number of core types, we can consider more than 6 configurations and collect the information. The collected information is then used to generate the required models. We select and run 9 different applications from Rodinia benchmark suite [5], which provides workloads with a wide range of behavior and is suitable for embedded processor [7], [8], [17], [18], [25]. We execute each application in the aforementioned six configurations on the Odroid XU3 heterogeneous platform [14] to collect power consumption, instructions-per-second, and execution time. Considering various levels of frequency, the number of total configurations is large even for off-line analysis. Thus, we estimate the power consumption, instructions-per-second, and execution time for the other frequency levels by using the power and performance models [15], [20], [23]. We use the collected data and estimated metrics to model the configuration bias, energy consumption, and performance satisfaction of each application in each configuration. The bias model and performance model are used at on-line management to minimize energy consumption. Instead of storing the data for all the configurations, we just store energy consumption of 6 various configurations - 1L, 2L, 4L, 1b, 2b, 4b into *App_Info_Array*, as application characteristic. The populated *App_Info_Array* encapsulates the required information for calculating the weighted bias, using the bias model. The details of the bias model and performance model are explained in the following.

3.1.2 Model generation

After data collection, the data combination, i,e, (num_cores, $core\_type$, $freq(GHz)$, $E_1...E_6(J)$) is used to design a generic i) *bias model* and ii) *performance model* for all the applications. $E_1...E_6$ are the measured energy consumption for the 6 various configurations. The *bias model* is used at run-time to estimate applications' biases for each configuration. To prevent exhaustive calculation at run-time, we use a performance model to eliminate the configurations in which the application does not meet the performance requirements. For each application, the user can define a specific requirement in terms of the execution time of the application. We translate the user requirement to instruction per second and design a classifier that determines a specific configuration for an application satisfies the requirements or not. The input features of our models are (num_cores, $core\_type$, $freq$, $E_1...E_6$), and output is CB_c for *bias model*, and a binary label, which shows meeting (1) or not meeting (0) of the performance requirement, for *performance model*. The linear relationship between inputs and output motivates us to select Linear Regression for *bias model*, and Ridge classifier for the *performance model*. Linear Regression and Ridge classifier are easy to implement and have low prediction error for the set of inputs and outputs that we define. The bias for each configuration is estimated as follows, using a linear regression model.

$$CB_c = \sum \beta_i x_i \tag{4}$$

where x_i are the input features, and β_i is the regression coefficients learned using the collected data as follows:

$$\begin{aligned} CB_c = \; & -0.3freq - 0.05core\_type + 0.04num\_cores \\ & + 0.01E_3 + 0.0017E_4 \\ & - 0.0005E_5 + 0.001E_6 + 0.86 \end{aligned} \tag{5}$$

Similarly, we have a classifier for the performance model as follows:

$$P = \sum \gamma_i x_i \tag{6}$$

where P is the classification value, and γ_i is the ridge regression coefficients learned for classifier by using the collected data. The more specific model based on Equation 6 is:

$$\begin{aligned} P = \; & 0.7freq - 0.74core\_type + 0.14num\_cores \\ & + 0.01E_1 - 0.07E_2 + 0.051E_3 - 0.0018E_4 \\ & + 0.013E_5 - 0.012E_6 - 0.17 \end{aligned} \tag{7}$$

When P is positive the predicted label is 1 (meet the requirements), and when it is negative the label is 0 (do not meet).

3.2 On-line management

The on-line *Controller* (as in Figure 3) receives applications' models, generated from the off-line profiling phase to make resource configuration decisions. This Controller consists of four main components viz., *Application Monitoring*, *Predictor*, *Configuration Selection* and *Performance Controller*, as shown in Figure 4. Application Monitoring checks arriving and exiting of the applications and creates a list of running applications with their information that collected off-line, called *App_Info*. *App_Info* for each application is passed to the Predictor, and the predictor estimates the list of bias for all the possible configurations of that application. Then, Configuration Selection determines a suitable resource configuration (core type, DoP, and frequency), followed by Performance Controller, which decides on resource scaling to honor performance requirements.

3.2.1 Application Monitoring

For handling a dynamic and unknown workload scenario, we periodically monitor the arriving and exiting of the applications at run-time and update a list of running applications, which will be used in Configuration Selection. The monitoring period is generic and can be adjusted at design time. Application Monitoring extracts *App_Info_Array* for each application and pass the information to the predictor for bias generation. Our approach is generic but require

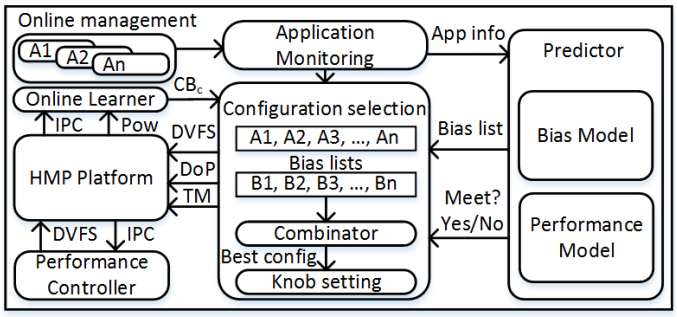

Fig. 4: On-line management overview.

one time off-line profiling for collecting *App_Info_Array*. For unknown applications, we use *Online Learner* module to collect data at run-time. *Application Monitoring* can detect application phase change and send a new *App_Info_Array* for each phase of the application. However, in our case study, the phase changes of applications do not significantly affect energy consumption, thus we overlook that. In future work, we will consider the various applications to show the ability of phase change detection in *Application Monitoring*.

3.2.2 Predictor

Whenever a new application arrives, the Application Monitoring informs the *Predictor*, and the *Predictor* uses two models, which are designed off-line, to generate a bias list for that application. The *Predictor* receives the *App_Info*, which contains $E1 - E6$, then estimates the configuration biases for various combinations of core type i.e., b, L, DoP i.e., 1-4, and frequency i.e., 0.2-1.4 GHz for LITTLE and 0.6-2 GHz for big. To prevent excessive calculation, we first check if the configuration meets the performance requirement for the application, using the performance model. Then, we calculate the bias for that application-configuration pair and add the bias to the bias list. Each bias is linked to one possible configuration.

3.2.3 Configuration Selection

When a new application arrives, Application Monitoring informs the *Configuration Selection*, and *Configuration Selection* checks the *App_Info* for profiled information of the application to select the configuration with the least energy consumption (among E_1 - E_6). If this is the only application currently running, the selected configuration is finalized. If there are other application(s) that are concurrently running, then the *Controller* searches for the optimal and feasible configurations for all the running applications. A chosen configuration is considered to be feasible when no two applications are running on the same core. Mapping more than one application to one core cause relatively higher energy consumption and lower performance [6] compare to the other configurations (as shown in the motivation example in Section 2.1). As our selected platform has 8 cores, the framework can handle up to 8 concurrent applications which is relatively high for embedded systems such as smartphones and smartwatches. In the following, the details of handling multiple applications are explained.

Handling Multiple Applications: The resource configuration with the lowest energy consumption (maximum bias) for one application can be unfeasible intuitively due to other concurrently running application(s). In this case, the *Configuration Selection* formulates a configuration subspace combining the off-line characterized biases of each running application. The *Combinator* component in Figure 4 combines the running applications weighted bias lists and generates a combined bias list by summing up every combination of weighted biases with feasible configuration. Each element in the combined bias list is linked to a set of feasible configurations for the running applications. The element with the highest weighted bias value links to the optimal set of configurations for the currently running applications. Whenever an application arrives or leaves the system, the application list updates, then the Combinator updates the combined bias list and selects the configuration that is linked to the maximum value. This new configuration may alter an application's previous optimal configuration to a sub-optimal choice, yet results in lowering the combined energy consumption of all the applications. The chosen new configuration is enforced through (re-) mapping, core folding/un-folding, task (re-) migration, and DVFS. In Odroid XU3 there is per cluster DVFS thus, when several applications are running in one cluster, the maximum frequency which can satisfy the requirements of all the applications will be selected.

3.2.4 Online Learner

Handling unknown applications. To have a self-contained framework, we provide an *online learner* as a trivial method for handling unknown applications. When a new application is detected by *Application Monitoring*, if the application is known, the best configuration can be selected immediately by using the *Predictor* and *Combinator*, otherwise, the online learner will be activated. The *Online Learner* collects power measurements and performance of the unknown application in the current frequency of the system by assigning the application to 6 various configurations (1L, 2L, 4L, 1b, 2b, 4b), if the cores are free (the measurement time interval is generic). We estimate the power consumption and instructions-per-second for the other frequency levels by using the power and performance models [15], [20], [23]. If there is not enough free core for collecting data in all the 6 above-mentioned configurations, the controller changes the current core assignment for the running applications to release at least one big and one LITTLE core for collecting data, then, after data collection, the controller predicts the best configuration and updates the core assignment for all the applications. By collecting such data, we can calculate CB_c values based on Equation 1. As the weight is in relation to the total energy consumption of the application, it can be calculated only after completion of the application. Therefore, for unknown applications, we overlook the weight and decide on the best configuration based on CB_c values. After application completion, we can calculate the weight and weighted bias by using Equation 2 and store it for the future. By having fewer free cores, our collected data is less, and thus the predicted configuration for the unknown application may not be the optimal configuration. However, the experimental result shows it still improves the energy consumption compare to the other frameworks. The online learner can improve in future works.

TABLE 2: Summary of the used applications in the experiments.

Application	Computation	Domain	Norm. compute int.	Abbr.	Notation	Arrival time (s)
Particle filter	Structured grid	Filtering problem	0.625	Pf	A1,A3,A6	0, 61, 243
Heartwall	Structured grid	Medical imaging	0.47	Hw	A2,A7,A8	6, 349, 379
Streamcluster	Dense linear algebra	Data mining	1	Sc	A4	67
Srad	Structured grid	Image processing	0.44	Sr	A5	237
Breadth-first search	Graph traversal	Graph algorithms	0.53	Bf	A9	383
Kmeans	Dense linear algebra	Data mining	0.8	Km	A11	395
Leukocyte tracking	Structured grid	Medical imaging	0.97	Lt	A10	389

3.2.5 Performance Controller

After finding the suitable configuration at the *Configuration Selection* block, the *performance controller* fine-tunes the level of DVFS for satisfying the applications' performance requirements. The DVFS level that is selected in *Configuration Selection* may not satisfy concurrent applications' requirements due to applications' interference and resource contention. Thus on-line monitoring of performance and adjusting DVFS at run-time is an important part of optimized resource allocation. Each application is assumed to present a target IPS as its nominal requirement. The *Performance Controller* monitors the application's performance (IPS) at run-time, and when the current IPS of one application is less than target, the DVFS level is increased one step i.e., 100 Mhz to meet the requirements. The on-line controller operates at every parametrizable epoch e. Over every epoch, the application mapping, TM, and DVFS decisions are enforced to the platform. When the system is idle, and no application is monitored by *Application Monitoring*, the *Performance Controller* sets the frequency of each type of cores to their minimum level.

4 Evaluation

4.1 Experimental Setup

Platform: We validate our experiments on an embedded HMP platform of Odroid XU3 [14], which hosts a Samsung Exynos 5422. The processor has 8 cores, organized into *big* cluster - with 4 ARM A15 cores for high performance and *LITTLE* cluster - with 4 ARM A7 cores for low power operation. The platform provides on-board power sensors for measuring power consumption and operating system's performance counters for measuring performance in terms of instructions per second. In the platforms without on-board power sensors, we can use power models [27] for estimating power consumption. Energy consumption is calculated by multiplying of power consumption of cores and time duration -which includes energy consumption of OS, our framework, workload scenario, and other background applications. Power consumption of memory is negligible in Odroid XU3 compare to the big and LITTLE CPUs [2]. Per-cluster DVFS is supported by the platform by which the frequency of big cores is adjustable between 0.2 to 2 GHz with a 0.1 GHz step and the Little cores' frequency is adjustable from 0.2 to 1.4 GHz with the same steps size.

Implementation: The framework in this paper is implemented as a Linux user-space process and is built on top of the MARS framework [19]. The monitoring and resource allocation decision making operate at a parameterizable epoch, which is set to 1 s in this work.

Workload: To show the effectiveness of the proposed method, we use a combination of single and multi-thread applications from Rodinia benchmark suite [5] and create a dynamic workload scenario. A summary of the used applications, their notations, and arrival times in our experiments is shown in Table 2. The applications cover wide range of compute/memory intensity from high compute intensity like `Streamcluster` to low compute intensity like `Srad`. The compute intensity levels are calculated using the speedup definition in [4] and normalized to one using the highest speedup. The higher speedup shows higher compute intensity for an application [4]. The created workload represents the behavior frequently encountered in heterogeneous embedded systems [18]. The workload contains various combinations of applications (in Table 2) entering and leaving dynamically to make different sets of concurrent applications for evaluation. For further evaluation, we use some other applications from the machine learning area and the Mibench benchmark suite [13] as unknown applications. These applications are Sha, Patricia (Pa), and Dijkstra (Dij) which are in the network and security category from Mibench and Knn, linear regression (Lr), and list square (Ls) algorithms from machine learning.

Comparison: We compared our proposed method with recently proposed resource management approaches i) DyPO [10] - which relies on off-load characterization for exploration of optimal resource configuration and ii) AdaMD [4] - which relies on on-line data collection for estimating speed-up and resource allocation based on that. While DyPO is designed to handle single application scenarios, it still considers combinations of actuation knobs and bias for resource management decisions, making it relevant for qualitative comparison against our approach. The frameworks presented in [22], [26] can handle multiple workloads, however, these methods do not consider application bias, performance requirements, or combination of essential actuation knobs in HMPs (discussed in Section 2.2). For example, the framework in [26] just achieves 3% improvement versus the powersave governor with limitation in considering multi-thread applications. For a fair comparison, we manage DyPO in a way to adapt multi-application scenarios. DyPO prioritizes applications based on performance-per-power and does not consider the dynamic execution of concurrent workloads. However, AdaMD considers the dynamic execution of concurrent applications, using adaptive mapping and dynamic DVFS. AdaMD estimates speed-up i.e., ratio of performance in the big and LITTLE cluster for each application, by collecting data for 500 ms at run-time.

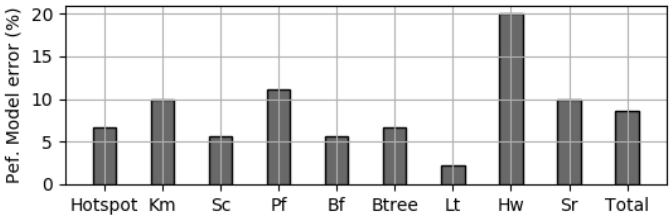

Fig. 5: Error in performance model for various applications.

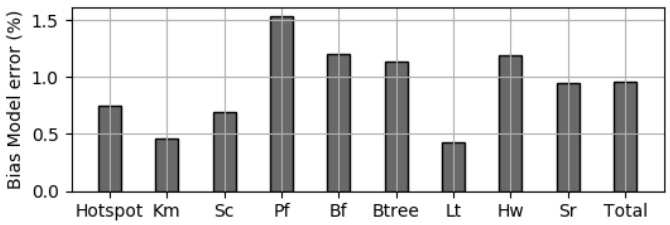

Fig. 6: Error in bias prediction for various applications.

Then, the speed-ups of applications are used as the ratio between the number of big and LITTLE cores that assigned to those. After assigning the cores to the applications by such a ratio, if still there is a free core, it is assigned to the application with the highest speed-up. AdaMD monitors the performance at run-time and increases the frequency level when performances of applications are lower than the required target.

In addition, we compare our method against Linux's *powersave*, *performance*, *ondemand*, *interactive*, and *conservative* governors, which are standard baselines, as used on millions of smartphones [22]. Powersave and performance governors set the frequency to the lowest and highest level to provide low power and high performance, respectively. The other governors adjust the frequency level based on utilization threshold.

4.2 Experimental Results

In this section, we present the experimental results containing i) accuracy of the proposed models, ii)functional analysis of our method during real experiments, and iii) evaluation and comparison of energy and performance trade-off against state-of-the-art approaches.

4.2.1 Accuracy evaluation

As discussed in Section 3, we design two models for performance prediction and bias prediction. The performance model is a classifier that receives features of an application and one configuration point, as a data sample, and predicts whether the input configuration satisfies the application requirement (class 1) or not (class 0). For accuracy evaluation of the performance model, we test our model by using 810 data samples. These 810 data samples are various combination of num_cores, $core\_type$, and $freq$ for 9 applications from the Rodinia benchmark suite. We consider 3 states for num_cores - 1, 2, and 4, 2 states for $core\_type$ - big and LITTLE, and 15 states for frequency levels, which create 90 various configurations. We use 80% of these samples for training. Figure 5 shows the percentage of error for the performance model when using for each application on 90 configurations. For most of the applications, the prediction error is less than 10%, except Hw and Pf. The last bar in Figure 5 shows the error of the performance model by considering all the 810 data samples which is less than 10%. For compensating the performance prediction error, we provide run-time performance monitoring in our framework.

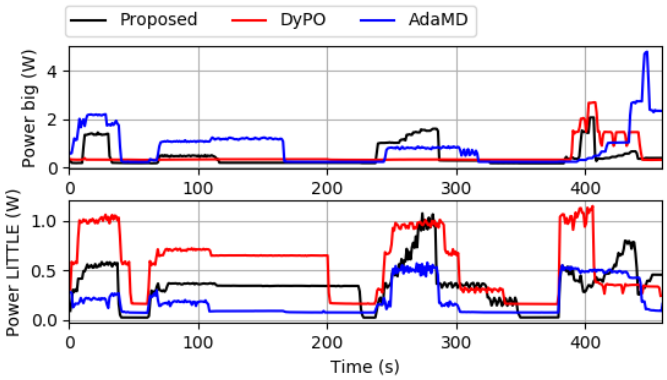

Fig. 7: Power consumption of big and Little clusters, running dynamic workload scenario, using our proposed method, DyPO, and AdaMD.

We also evaluate the prediction error of the bias model which is explained in Section 3.1. The input features of this model are the same as the performance model, and the outputs are bias predictions. Figure 6 shows percentage of error in bias prediction for each application in various configuration points. The bias error is less than 1.6% for all of the applications, which is low enough. The last bar in Figure 6 shows the total error of the bias model by considering 810 data samples that is less than 1%. To this end, our approach provides reliable prediction models leading to optimizing energy consumption under performance constraints better than state-of-the-art approaches.

4.2.2 Functional analysis

For the evaluation purpose, we present the instantaneous power consumption of the *big* and *LITTLE* cores on the system to show the effect of resource management decisions, using DyPO, AdaMD, and our proposed approach. We ran various combinations of 7 different applications (presented in Table 2) from the Rodinia benchmark suite [5], such that there are at least 2 applications that execute concurrently. Figure 7 shows the variation of power during the whole run-time by using a dynamic workload scenario. As shown in Figure 7, AdaMD consumes relatively higher power in the big cluster and lower power in the LITTLE cluster. The AdaMD maps applications to big and LITTLE cores in a ratio of their speedup, for balanced workload sharing between big and LITTLE cores. For example, if the speedup is 2 for an application, AdaMD assigns 2 big and 1 LITTLE core to that application, thus the power consumption of the big cluster is relatively higher. Conversely, in DyPO, the power consumption of the LITTLE cluster is higher due to mapping applications to LITTLE cores for lower energy consumption. In DyPO, the best mapping configuration for every single application is mapping to LITTLE cores, except for Km and Lt, leading to an increase of power for the big cluster at $t = 389s$ and $t = 395s$. Our approach tries to map

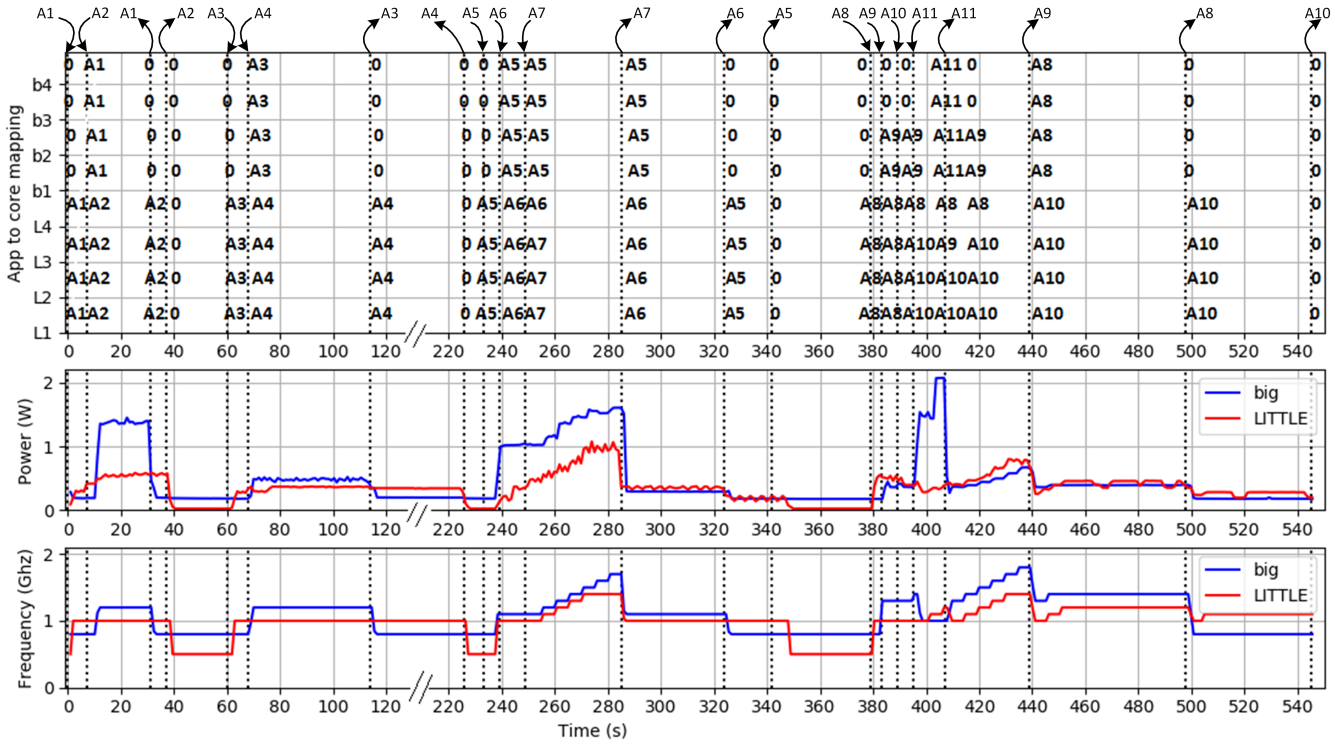

Fig. 8: Frequency, power consumption, and application to core mapping by using our framework when a dynamic workload scenario is running. Applications arrive and leave in an unknown manner. The dotted line indicates the arrival or leaving of applications. L1-L4 are LITTLE cores and b1-b4 are big cores.

the applications to the best configuration either LITTLE or big cores. Thus, the power consumption of LITTLE cores in our approach is lower than DyPO, which maps most of the applications to LITTLE cores. Our approach uses the big cores, whenever LITTLE cores are busy, leading to higher power consumption for the big cluster compared to DyPO.

For a detailed analysis, we zoom in on the power consumption of the big and LITTLE clusters, using our approach. Figure 8 shows details of frequency, power consumption, and application to core mapping at the whole run-time. We demonstrate the times when an event occurs (applications arrive or leave) by vertical dotted lines and directional arrows, which show the arriving or leaving of applications. By occurring each event, our resource management technique decides for task migration, task mapping, and DVFS, after calculating *Weighted Bias*. In Figure 8, A2 arrives when A1 is running, which leads to migration of A1 to the big cluster. The same scenario happens for A3 and A4. Due to the lower weighted biases for A1 and A3, A2 and A4 have a relatively higher priority for mapping to the LITTLE cluster. By assigning each application to each cluster, our framework increases the frequency of that cluster for performance satisfaction, thus the power of that cluster increases (as shown in power and frequency plots). When A5, A6, and A7 execute concurrently, our framework combines the bias lists of these 3 applications and selects the configuration with maximum combined bias, which is shown in Figure 8. With the new configuration, the requirements of applications are not satisfied due to the congestion and interference, thus the frequency of the big and LITTLE clusters is increased step by step, resulting in power increasing. By the completion of each application, the mapping configuration for the rest of the applications is updated. A8, A9, A10, and A11 create 4 concurrent applications scenario, which leads to updating configuration after the arrival of each application and increasing power consumption of big cluster up to 2 W. After completion of A11, the new configuration does not satisfy the performance requirements of the 3 remained applications, thus our framework increases frequency step by step (from $t = 407s$ to $t = 439s$). As shown in Figure 8, our approach adapts to dynamic workload scenarios and handles multiple concurrent applications at run-time for minimizing total energy consumption.

4.2.3 *Mapping configurations*

The decisions of our resource management strategy are guided by *Weighted Bias*, while in DyPO, decisions for single application scenarios are guided by *Bias* i.e., performance-per-power. The application with higher *Weighted Bias* has a higher impact on energy consumption, which should thus be assigned a higher priority for resource allocation and energy optimization. Decisions on AdaMD guided by speedup and similar to DyPO does not consider any weight for applications. For comparing the mapping strategies between DyPO, AdaMD, and our method, we present run-time mapping configuration with these three approaches at $t = 0$, $t = 7$, and $t = 31$ in Table 3. The table shows how mapping configuration changes by the arrival of A2 and completion of A1 in three different frameworks. At $t = 7$, by A2 arrival, DyPO searches for the best configuration based on the bias. The best configuration is 4 LITTLE for A2, thus DyPO assigns 4 LITTLE cores to A1 and A2 simultaneously,

TABLE 3: Run-time mapping configurations with different approaches. Mapping configuration is shown as a per-cluster vector, where AN represents application-N being mapped to a core and 0 represents idle core.

Instance	t = 0		t = 7		t = 31	
Cluster	LITTLE	big	LITTLE	big	LITTLE	big
DyPO	A1 A1 A1 A1	0 0 0 0	A1A2 A1A2 A1A2 A1A2	0 0 0 0	A2 A2 A2 A2	0 0 0 0
AdaMD	A1 0 0 0	A1 A1 0 0	A1 A2 0 0	A1 A1 A2 A2	A2 0 0 0	A2 A2 A2 0
Proposed	A1 A1 A1 A1	0 0 0 0	A2 A2 A2 A2	A1 A1 A1 A1	A2 A2 A2 A2	0 0 0 0

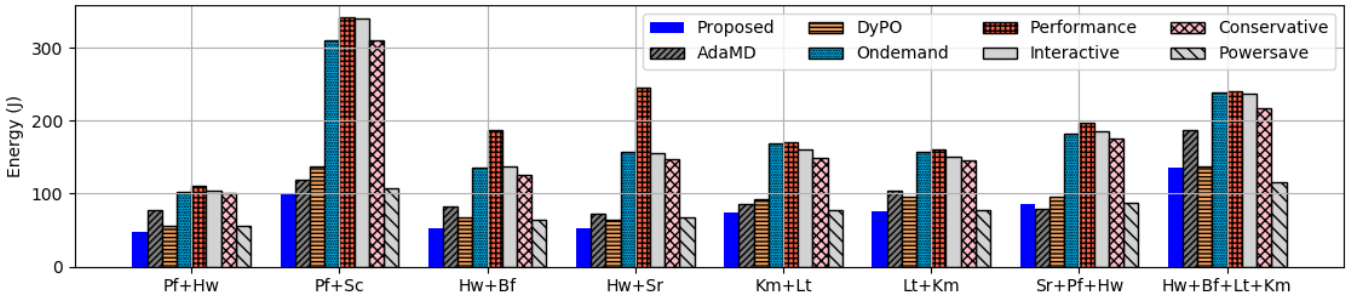

Fig. 9: Energy consumption of system over various dynamic workload scenarios, using different frameworks.

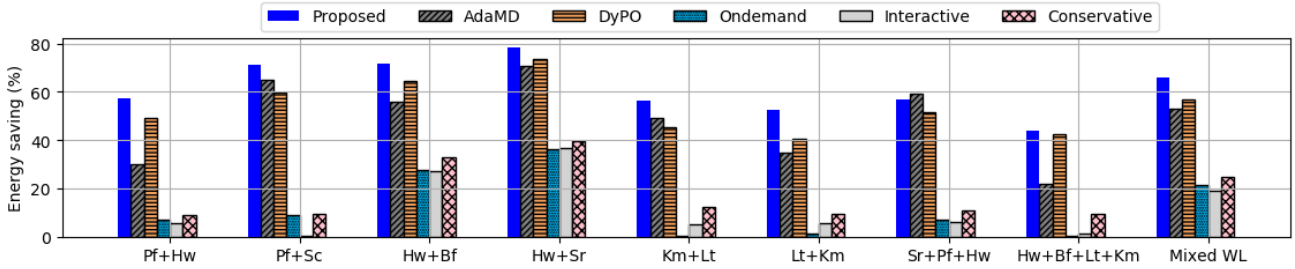

Fig. 10: Energy savings achieved by our approach compared to different approaches for various dynamic workload scenarios.

which leads to overloading LITTLE cores and decreasing performance of A1, and A2. While running both A1 and A2 on 4 LITTLE cores leads to lower power consumption, increasing execution time of these to applications result in higher energy consumption. On the other hand, our method by calculating weighted bias finds mapping A1 to 4 big and A2 to 4 LITTLE is more efficient. AdaMD estimates speed-up which is 2 for both A1 and A2, leading to assigning 2 big cores and 1 LITTLE core for each application. In AdaMD, when an application leaves the system and cores are free, the controller assigns one more core to the application with the highest speedup, thus when A1 leaves the system at $t = 31$, A2 is mapped to 3 big cores and one LITTLE core. Our resource allocation strategy chooses a resource configuration suitable for minimizing the energy consumption of the prioritized application, while any other concurrent applications may be re-allocated to sub-optimal configurations, the overall energy consumption is minimized.

4.2.4 Energy and performance evaluation

Figure 9 shows a comparison of energy consumption of the system, by using different approaches under various concurrent workload scenarios. All the workload scenarios are dynamic and applications arrive and leave the system in an unknown manner. The blue bar shows the energy consumption of the system using our approach, which is the lowest in most of the combinations of concurrent workloads. Although *powersave* approach provides relatively low energy consumption, it does not respect performance requirements and increases the execution time. For the first two workload scenarios, DyPO assigns both applications to 4L which is the best configuration for a single execution of each one. Such mapping may result in lower power consumption but higher execution time, leading to higher total energy consumption compare to our method. On the other hand, AdaMD uses less number of cores compared to our method, however, Figure 9 shows its mapping strategy is not optimal. The power consumption by AdaMD is high due to using the big cores which results in even higher energy consumption compare to DyPO in the first workload. In the third and fourth workloads, the best configurations for singular execution are 4L for *Hw* and 1L for *Bf* and *Sr*. Thus, when *Hw* is running singular, our method map it to 4L, and when the second application arrives *Hw* releases one core for that. Therefore, the total energy consumption in our approach is less than the other approaches. In dynamic workload scenarios, the order of arriving applications also affects energy consumption and must be considered by resource management. DyPO does not consider arriving order of applications, thus for both *Km+Lt* and *Lt+Km* workloads the selected configurations are the same, and the energy consumption is not optimal. However, our method and AdaMD

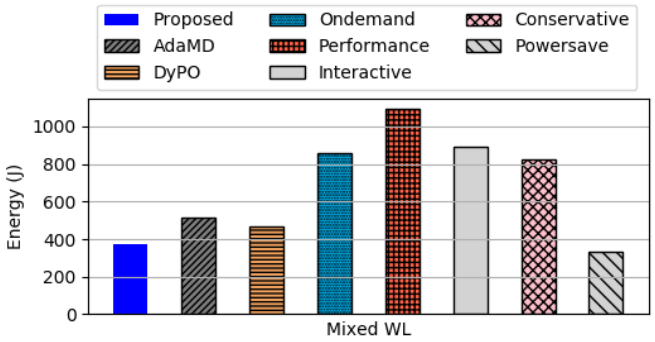

Fig. 11: Energy consumption of system over mixed dynamic workload scenario, using different frameworks.

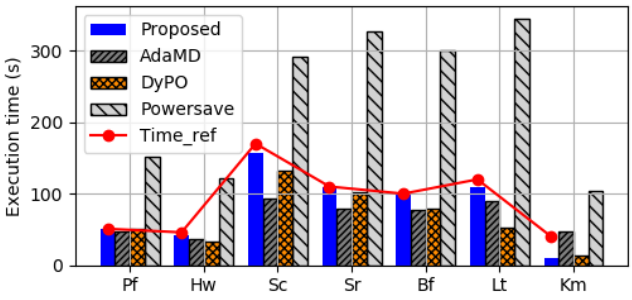

Fig. 12: Execution time of running applications in dynamic workload scenario, using various frameworks.

consider such factor and manage the resources based on that. AdaMD maps the applications to the available cores based on their arriving orders. Our method by considering weighted bias and migration of applications at run-time can handle the dynamic workloads, leading to lower energy consumption in both cases. Our method maps each application to its best configurations at first, and when the second application arrives, changes the configuration. As shown in Figure 9, AdaMD results in lower energy consumption for 3 concurrent applications compared to our method, however, it leads to significantly higher energy consumption for 4 concurrent applications scenario.

To this end, our method provides lower energy consumption compared to the other related works considering the dynamic arrival of applications. We create a mixed workload scenario (Mixed WL) by combining 11 applications (as in Figure 8) to evaluate the approaches under more complex workloads. The energy consumption under such a workload scenario is shown in Figure 11. Our framework, which is shown in the blue bar has the lowest energy consumption. Thus, while the other approaches may provide lower or equal energy consumption under some workload scenarios, our method offers the lowest energy for complex workloads.

For better evaluation, we present percentage of energy saving for each approach against *performance* governor in Figure 10. Our approach (blue bar) provides the highest energy saving in most of the workload combinations. Our framework saves energy consumption up to 12% ,27%, and 61% more than *DyPO*, *AdaMD*, and Linux governors, respectively. While in the mixing of 3 concurrent applications, AdaMD has higher energy saving, in *Mixed WL*, which is a

Fig. 13: Execution time- Energy trade-off

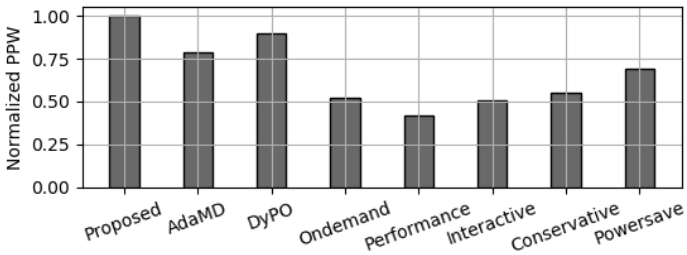

Fig. 14: PPW (Performance per watt) comparison

more complex workload scenario AdaMD has lower energy saving. In summary, our approach provides higher energy saving under dynamic workload scenarios compare to the state-of-the-art approaches.

Figure 12 shows the average execution time of each application that is used in *Mixed WL* scenario, using various resource allocation approaches. The red circles show the execution time target for each application, the execution time higher than the red circles is not acceptable. The blue bar in Figure 12 shows our approach, which meets the execution time targets for all the applications in the Mixed WL scenario while minimizing the energy consumption. Figure 12 shows *powersave* governor neglects applications target requirements and leads to the execution time up to 3 times higher than our approach. Thus, our approach provides an optimum trade-off between the execution time of applications and energy consumption.

Figure 13 shows the trade-off between energy consumption and execution time and where each approach stands on this trade-off. The ideal solution would be at the bottom left corner of this plot. Therefore, the closer each approach to the bottom left corner, the better its overall utility. The figure shows *powersave* approach offers the lowest energy consumption by compromising the execution time. The *performance* approach has the highest energy consumption with a little improvement in execution time compare to *interactive*, *conservative* and *ondemand* approach. *AdaMD* and *DyPO* provide lower energy consumption compare to the aforementioned governors while still satisfy the performance requirements, as shown in Figure 12. Our approach provides a suitable trade-off that offers lower energy consumption than *AdaMD* and *DyPO* while meeting the execution time targets.

Another metric for the evaluation of frameworks that focus on energy optimization is performance-per-watt (PPW). The higher PPW shows relatively higher performance and lower power consumption. Figure 14 shows comparison of

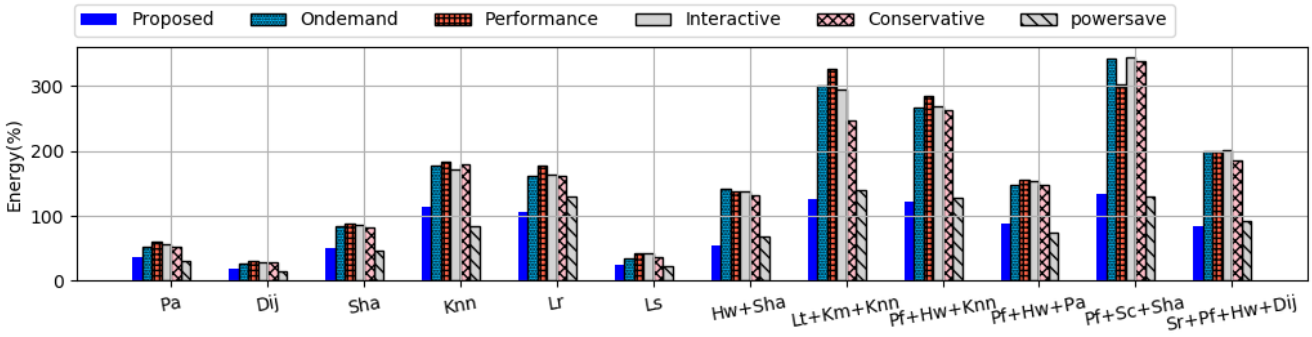

Fig. 15: Energy consumption of system over various workload scenarios with unknown applications, using different frameworks.

normalized PPW in various approaches under *Mixed WL*. In this Figure, the PPW for each approach is normalized to 1, using the highest value for PPW. Our approach provides the highest PPW, which is 22% higher than AdaMD, and 11% higher than DyPO.

Handling unknown applications: We evaluate our framework by some unknown applications without any initial information. We compare the energy consumption of the system under such workload scenarios against the Linux governors, which is shown in Figure 15. Dypo and AdaMD do not support unknown application scenarios, thus could not optimize energy consumption under such kind of workload scenarios. Figure 15 shows energy consumption for various workload scenarios from 1 unknown application executing individually to a combination of known and unknown applications. As shown in Figure 15, Our framework still offers the lowest energy consumption. Although Powersave governor may provide low energy consumption, it cannot guarantee performance requirements.

4.3 Run-time overhead

We estimate the run-time overhead of our framework by considering i) CPU and memory usage, ii) execution time overhead, and iii) energy overhead.

CPU and Memory. The memory usage of our framework is $O((N_{core}.N_{freq})^{N_{App}})$, where N_{core} is the number of cores in each cluster, which is 4 in our platform, N_{freq} is the maximum number of frequency levels, which is 18 in our platform, and N_{App} is the number of concurrent applications. Whenever a new application arrives, we require temporary storage with the above-mentioned order to generate the combined bias list and make a resource management decision. We monitor the CPU usage of our framework by using `htop` command in the Linux. The CPU usage is variable depends on the stage of the execution of our framework. For example, when a new application arrives, or an application leaves the system, the *Configuration Selection* is called, and the CPU usage by our framework increases for at most 2.2 s. The maximum CPU usage by our framework is 23% of one big core, which is 3% of all the available cores. During the execution of applications, our framework uses 11% of one big core, which is 1% of all the available cores.

Execution Time. The major part of execution time of our framework is related to *Configuration Selection* (as presented in the Figure 4). The execution time of *configuration selection* is variable depends on the number of concurrent applications and the number of feasible configurations for those applications. The space of feasible configurations and the configurations that meet the target requirements of the applications varies at run-time. The smaller space requires less exploration time. The time complexity of our framework is $O((N_{core}.N_{freq})^{N_{App}})$. We measure the execution time of our framework for various workload scenarios from 2 to 4 concurrent applications and the results show time interval varies from 0.09 - 2.2 s which is negligible compare to the applications execution time (from 75 to 546 s).

Energy. For calculating the energy overhead, we monitor the instantaneous power consumption of our framework. The power consumption of our framework is considerable only when *Configuration Selection* is called. When *Configuration Selection* executes, the power consumption of our framework increases up to 0.2 Watt for at most 2.2 s. The number of calling *Configuration Selection* depends on the workload scenario, for example in 2 concurrent workload scenario it is called 2 times and in the Mixed WL, it is called 22 times. If we consider the highest energy consumption (0.2×2.2) for all the 22 times, the total energy consumption of our framework would be $0.2 \times 2.2 \times 22 = 9.68J$, and the energy overhead is 9.68 divided by the total energy consumption of the system, which is 420 J in Mixed WL. Therefore, the energy overhead is 2.3%, which is considered in the reported experimental results.

5 Conclusions

We proposed a resource management strategy for minimizing energy consumption of multi-programmed workloads by characterizing each application's power-performance profile off-line. We model the applications' weighted bias from the off-line profiling and use that for the on-line management of concurrent applications. In on-line management, we combine applications' biases to find the best configuration that provides the lowest energy consumption while honoring the performance requirements of concurrent applications. We evaluated the proposed approach on a real

embedded platform against other relevant resource management strategies. Our approach provides lower energy consumption while satisfying the performance requirements of the standard embedded benchmark suite.

REFERENCES

[1] A. Aalsaud et al. Model-free runtime management of concurrent workloads for energy-efficient many-core heterogeneous systems. In *PATMOS*, 2018.

[2] Ali Aalsaud, Rishad Shafik, Ashur Rafiev, Fie Xia, Sheng Yang, and Alex Yakovlev. Power–aware performance adaptation of concurrent applications in heterogeneous many-core systems. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pages 368–373, 2016.

[3] M. A. Al-hayanni et al. PARMA: Parallelization-Aware Runtime Management for Energy-Efficient Many-Core Systems. *IEEE Transactions on Computers*, 2020.

[4] K. R Basireddy et al. AdaMD: Adaptive mapping and DVFS for energy-efficient heterogeneous multi-cores. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[5] Sh. Che et al. Rodinia: A benchmark suite for heterogeneous computing. In *Proc. of IISWC*, 2009.

[6] R. Cochran et al. Pack & Cap: adaptive DVFS and thread packing under power caps. In *MICRO*, 2011.

[7] AM Coutinho D. et al. Performance and Energy Trade-Offs for Parallel Applications on Heterogeneous Multi-Processing Systems. *Energies*, 2020.

[8] Y. Ding et al. Generative and multi-phase learning for computer systems optimization. In *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.

[9] B. Donyanavard et al. Sparta: Runtime task allocation for energy efficient heterogeneous manycores. In *CODES+ ISSS*, 2016.

[10] U. Gupta et al. DyPO: Dynamic pareto-optimal configuration selection for heterogeneous MpSoCs. *ACM TECS*, page 123, 2017.

[11] U. Gupta et al. STAFF: online learning with stabilized adaptive forgetting factor and feature selection algorithm. In *Proc. of DAC*, 2018.

[12] U. Gupta et al. A deep q-learning approach for dynamic management of heterogeneous processors. *Computer Architecture Letters*, 2019.

[13] M. R Guthaus et al. MiBench: A free, commercially representative embedded benchmark suite. In *Workload Characterization*, 2001.

[14] Hardkernel. ODROID-XU.

[15] A. Kanduri et al. Approximation-aware coordinated power/performance management for heterogeneous multi-cores. In *Proc. of DAC*, 2018.

[16] S. K Mandal et al. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[17] N. Mishra et al. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *ACM SIGPLAN Notices*, 2015.

[18] N. Mishra et al. CALOREE: Learning control for predictable latency and low energy. *ACM SIGPLAN Notices*, 2018.

[19] T. Muck et al. Adaptive-Reflective Middleware for Power and Energy Management in Many-Core Heterogeneous Systems. In *Many Core Computing: Hardware and Software, IET*, 2019.

[20] A. Pathania et al. Power-performance modelling of mobile gaming workloads on heterogeneous MPSoCs. In *Proceedings of the 52nd Annual Design Automation Conference*, 2015.

[21] B. K Reddy et al. Inter-cluster Thread-to-core Mapping and DVFS on Heterogeneous Multi-cores. *TMSCS*, 2017.

[22] B. K. Reddy et al. Online concurrent workload classification for multi-core energy management. In *Proc. of DATE*, 2018.

[23] H. Rexha et al. Core level utilization for achieving energy efficiency in heterogeneous systems. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2017.

[24] E. Shamsa et al. Goal-Driven Autonomy for Efficient On-chip Resource Management: Transforming Objectives to Goals. In *Proc. of DATE*, 2019.

[25] D. Shingari et al. DORA: optimizing smartphone energy efficiency and web browser performance under interference. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018.

[26] S. Tzilis et al. Energy-Efficient Runtime Management of Heterogeneous Multicores using Online Projection. *ACM TACO*, 2019.

[27] Matthew J Walker et al. Accurate and stable run-time power modeling for mobile and embedded cpus. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

Elham Shamsa Elham Shamsa received the B.Sc. degree in computer engineering-hardware in 2012, and the M.Sc. degree in computer architecture in 2014, from Amirkabir University of Technology, Tehran, Iran. She is currently a Ph.D. student at the University of Turku, Finland. Her current research interests include system on-chip resource management, multi-objective resource management techniques, high-performance energy-efficient architectures, and self-aware systems.

Anil Kanduri received M.Sc (Tech) in Embedded Computing in 2014, and PhD (Tech) in Computer Systems in 2018, from University of Turku, Finland. He is currently a Senior Researcher at the department of Future Technologies, University of Turku. His research interests are in on-chip resource management, run-time power and performance modeling and analysis, system software for heterogeneous processors, and alternative computing methods and architectures.

Pasi Liljeberg received MSc and PhD degrees in electronics and information technology from the University of Turku, Turku, Finland, in 1999 and 2005, respectively. He received Adjunct professorship in embedded computing architectures in 2010. Currently he is working as a full professor in University of Turku in the field of Embedded Systems and Internet of Things. At the moment his research interest are computer architectures, fog computing, energy efficient architectures, approximate and adaptive computing, biomedical engineering, health technology and Internet of Things. In that context he has established and leading the Internet-of-Things for Healthcare (IoT4Health) research group. Liljeberg is the author of more than 260 peer-reviewed publications. He is a member of the IEEE.

Amir M. Rahmani is the founder of Health SciTech Group (healthscitech.org) at the University of California, Irvine (UCI). He is an Assistant Professor of Computer Science and Nursing (joint appointment) at the UCI, and is also a life-time adjunct professor (Docent) in the Department of Future Technologies of University of Turku, Turku, Finland. His research is in Internet-of-Things (IoT), e-health, wearable sensor design, embedded systems, bio-signal processing, health informatics, and big health data analytics. He has received numerous research excellence awards (e.g., 2x from Nokia Foundation), fellowships (Marie Curie Fellowship and Faculty Innovation Fellowship), is the co-author of more than 200 peer-reviewed publications, and is the associate editor-in-chief of ACM Transactions on Computing for Healthcare. He is a senior member of the IEEE.