Exploiting Approximation for Run-time Resource Management of Embedded HMPs

ZAIN TAUFIQUE, Health Technology, University of Turku, Turku, Finland
ANIL KANDURI, University of Turku, Turku, Finland
ANTONIO MIELE, DEIB, Politecnico di Milano, Milano, Italy
AMIR RAHMANI, Computer Science, University of California, Irvine, California, United States and Institute of Computer Technology, Technische Universitat Wien, Wien, Austria
CRISTIANA BOLCHINI, Dip. Elettronica e Informazione, Politecnico di Milano, Milano, Italy
NIKIL DUTT, Center for Embedded Computer Systems, University of California at Irvine, Irvine, United States
PASI LILJEBERG, IT, University of Turku, Turku, Finland

Run-time resource management (RTM) of multi-programmed workloads on heterogeneous multi-core platforms is challenging due to (i) fixed power budget of the device, (ii) variable performance requirements of the workloads, and (iii) unknown arrival of the applications. Existing RTM solutions lack power-performance coordination, resulting in performance degradation during power actuation or power violations during performance provisioning. Exploiting inherent error-resilience of the applications can address the performance loss incurred in power actuation, by combining run-time approximation with traditional power knobs (including Dynamic Voltage/Frequency Scaling, Task Migration, Degree of Parallelism, and *CPU Quota*). In this work, we present an accuracy-aware resource management framework that jointly actuates run-time approximation and traditional power knobs for efficient power-performance management of multi-programmed and multi-threaded workloads running on heterogeneous mobile platforms. Our strategy configures the accuracy of the applications at run-time to exploit accuracy-performance trade-offs, by considering system-wide power-performance dynamics. We use heuristic estimation models to jointly enforce accuracy configuration and traditional power knobs settings at run-time. We evaluated our framework on real-world embedded mobile platforms, including Odroid XU3 and Asus Tinker Edge R boards to demonstrate the efficiency of our proposed approach across multiple workload scenarios. Our approach achieved 25% lower performance violations against the state-of-the-art run-time resource management policies at the cost of 2.2% accuracy loss across six applications.

CCS Concepts: • **Computer systems organization → Real-time system specification;**

This work is supported by the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska Curie Grant No. 956090 (APROPOS).
Authors' Contact Information: Zain Tauique, Health Technology, University of Turku, Turku, Varsinais-Suomen, Finland; e-mail: zatauf@utu.fi; Anil Kanduri, University of Turku, Turku, Finland; e-mail: spakan@utu.fi; Antonio Miele, DEIB, Politecnico di Milano, Milano, Italy; e-mail: antonio.miele@polimi.it; Amir Rahmani, Computer Science, University of California, Irvine, CA, USA and Institute of Computer Technology, Technische Universitat Wien, Wien, Austria; e-mail: a.rahmani@uci.edu; Cristiana Bolchini, Dip. Elettronica e Informazione, Politecnico di Milano, Milano, Italy; e-mail: cristiana.bolchini@polimi.it; Nikil Dutt, Center for Embedded Computer Systems, University of California at Irvine, Irvine, CA, USA; e-mail: dutt@uci.edu; Pasi Liljeberg, IT, University of Turku, Turku, Finland; e-mail: pasi.liljeberg@utu.fi.

ACM 1539-9087/2025/04-ART39
https://doi.org/10.1145/3723357

Additional Key Words and Phrases: Run-time mapping, DVFS, approximation, and dynamic power

ACM Reference Format:
Zain Taufique, Anil Kanduri, Antonio Miele, Amir Rahmani, Cristiana Bolchini, Nikil Dutt, and Pasi Liljeberg. 2025. Exploiting Approximation for Run-time Resource Management of Embedded HMPs. *ACM Trans. Embedd. Comput. Syst.* 24, 3, Article 39 (April 2025), 30 pages. https://doi.org/10.1145/3723357

1 Introduction

Embedded, mobile, and edge systems are increasingly using Heterogeneous Multi-Processing (HMP) architectures to meet diverse performance requirements of applications within stringent power and energy budgets [27]. HMP systems are characterized by flexible use of their heterogeneous cores by (i) mapping applications on the most suitable cluster(s), and (ii) moving the application between different clusters for power/performance optimization. In particular, such optimization goal is pursued by means of Application Mapping (MAP) and Task Migration (TM) policies and Run-time Resource Management (RTM) techniques, jointly with traditional power knobs, such as Dynamic Voltage/Frequency Scaling (DVFS) [12, 24] and *CPU Quota* assignment [22, 35]. Existing RTM strategies use machine learning models to navigate wider power-performance trade-off space exposed by HMP systems through extensive offline profiling (e.g., [17, 26]), online learning (e.g., [6]), and hybrid models (e.g., [1, 13, 33]). However, the efficacy of these strategies is limited when facing unknown and dynamic workload variations, resulting in limited performance gains and/or performance degradation during power actuation. These limitations can be attributed to: (i) dependence on traditional power knobs with relatively orthogonal power-performance trade-offs, (ii) lack of coordination between power and performance actuation decisions, and (iii) limited considerations on knob actuation feasibility and run-time system dynamics.

On the other hand, mobile and edge HMP devices are increasingly being used for running lightweight machine learning kernels to deliver smart services. Such data-driven applications exhibit tolerance to inaccurate computations given their characteristics such as noisy and redundant inputs, stochastic outputs, and human perception as end result [34]. The approximate computing paradigm leverages the inherent error resilience of applications for performance and energy gains within an acceptable accuracy loss. Approximate computing is widely adopted in embedded processors by combining accuracy trade-offs with traditional power/performance knobs to maximize the performance within stringent power and energy budgets [11, 22]. Some of the existing approaches have used approximation for resource management in both static (e.g., [14, 25, 29]) and dynamic workload scenarios (e.g., [7, 20, 38]). These approaches create a limited and fixed number of approximate versions of an application at design time and opportunistically select an approximate version at run-time for power/performance gains. However, such design time coarse-grained approximation narrows down the accuracy-performance-power trade-off space, resulting in higher accuracy loss or limited gains with approximation. Exploiting accuracy-performance-power trade-offs require disciplined tuning of fine-grained approximation, along with the traditional knobs of DVFS, *CPU Quota* assignment, MAP, and TM.

Given these motivations, we propose run-time dynamic approximation for multi-programmed parallel workloads running on HMP architectures. We design a run-time resource management policy that configures the algorithmic parameters of applications at run-time to generate fine-grained approximate versions. The resource management policy determines run-time approximation and traditional resource configuration knob settings based on concurrent workloads' performance requirements, their error resilience characteristics, and system-wide power consumption. The resource management policy considers the error resilience of applications to

prioritize candidates for approximation among concurrently running applications, minimizing the overall accuracy loss. We design and implement a resource management framework that integrates our proposed policy with Operating System (OS) level interfaces for monitoring power consumption and per-application performance and enforcing resource allocation decisions on MAP, Degree of Parallelism (DoP), TM, DVFS, *CPU Quota* assignment, and run-time approximation. We target heterogeneous edge platforms having asymmetric *big.LITTLE* CPU clusters with non-uniform power and performance characteristics. We evaluated our strategy on two real hardware testbeds including Odroid XU-3 [18] and Asus Tinker Edge R [5] over relevant machine-learning micro-kernels. Our approach achieved 25% lower performance violations against the state-of-the-art run-time resource management policies at the cost of 2.2% accuracy average loss across six machine learning micro-benchmark applications.

A preliminary version of our resource management approach and policy for serial applications is partly presented in [22]. In this work, we design a comprehensive accuracy-aware runtime resource management approach to support multi-programmed and multi-threaded workloads, while dynamically prioritizing applications for exploiting run-time approximation based on their error resilience. Our novel contributions are as follows:

- Modular and scalable framework for run-time resource management of HMP architectures to handle unknown dynamic workload variation, composed of a mixture of concurrently running parallel and serial applications.
- Fine-grained run-time accuracy configuration of applications, and prioritization of applications as candidates for approximation based on error resilience characteristics.
- Run-time resource allocation policy that decides on a combination of run-time approximation, MAP, DVFS, *CPU Quota* assignment, TM, and DoP, by considering performance requirements and available power budget.
- Coordinating power actuation with performance management through analytical estimation models for joint optimization of approximation with other knobs.
- Experimental evaluation of our strategy on two different embedded heterogeneous platforms including Odroid-XU3 [18] and Asus Tinker Edge R [5], with different multi-threaded machine learning micro-benchmark applications.

Manuscript Organization: Section 2 provides background and motivation for our proposed approach, Section 3 presents overview of our run-time management framework infrastructure, Section 4 elaborates our proposed policy, Section 5 presents evaluation of our proposed solution against other relevant strategies, followed by conclusions in Section 6.

2 Background and Motivation

2.1 Knob Actuation Dynamics

We demonstrate the diversity in power-performance characteristics among different knobs through the example of *least squares* micro-kernel executed on an HMP platform (that is Odroid XU-3 described in Section 5). Figure 1(a) shows the normalized performance when actuating power consumption using DVFS (scaled in steps of 100 MHz), and *CPU Quota* assignment (scaled in steps of 5%). DVFS has a significant reduction in power consumption with the quadratic effect of lowering both Voltage/Frequency (VF) levels, whereas scaling *CPU Quota* has almost a linear effect on power and performance. For the same application, Figure 1(b) shows normalized performance with varying levels of DoP (1 to 4 threads), combined with DVFS by scaling the frequency levels. Figure 1(c) shows the performance metrics with migrating the application from high performance (big) cluster to low power (LITTLE) cluster - for both accurate (acc) and approximate (apx) kernels. Upon task migration, the performance of the accurate version of

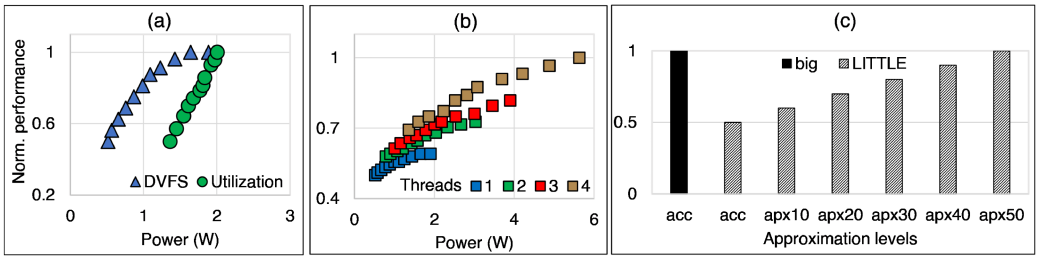

Fig. 1. Knob actuation dynamics. (a) DVFS and CPU Utilization, (b) Parallelism, (c) Task migration and approximation.

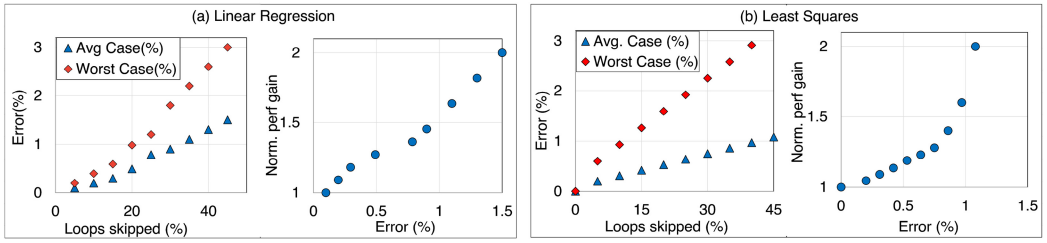

Fig. 2. Accuracy-performance trade-offs for different applications. (a) Linear regression, (b) Least squares.

the application is reduced by about 2.8×, whereas power consumption is significantly lowered by almost 2×. In this scenario, Figure 1(c) also shows normalized performance with different approximate versions, where 10-50% of the workload is reduced using loop perforation. Within the same power consumption, each progressive approximate version provides higher performance on the LITTLE cluster in comparison with the accurate version.

While the aforementioned knobs expose various power-performance configuration options, selecting an optimal knob setting depends also on feasibility. For instance, consider two applications with different performance requirements running on the same cluster. Actuating DVFS for tuning the performance of one application would affect the other application also, being VF settings scaled at the cluster level. Thus, tuning the performance of a single application would require the actuation of an application/process-level knob (*CPU Quota* assignment/approximation) and such a knob setting would not be optimal on a generic power-performance trade-off space of all possible knobs. While the knob actuation dynamics presented in Figure 1 are widely understood, jointly actuating all these knobs at run-time considering dynamic workload variation, power-performance Pareto-space, and knob actuation feasibility is a complex optimization problem. In our work, we build analytical models for the estimation of power consumption and performance with every resource allocation decision, to coordinate among the actuation of different knobs.

2.2 Error Resilience Considerations

Outcomes of traditional power/performance knob actuation are deterministic, enabling analytical modeling of power and performance with specific knob settings. However, approximation is entirely application-specific, such that applications with higher error resilience could achieve higher performance gains with approximation. We demonstrate the variation of error resilience among different applications with an example. Figure 2 shows the accuracy-performance trade-offs of two applications viz., *linear regression* and *least squares*. In each case, loop perforation [34] is used to skip a percentage of input data to generate different approximate versions. We experimented

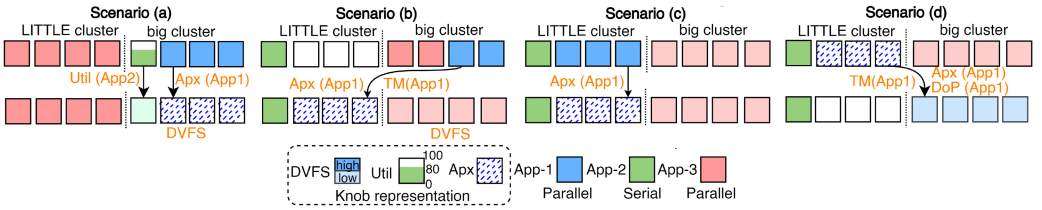

Fig. 3. Exemplar knob actuation scenarios.

with 1,000 different random data sets and collected the relative root mean square error induced with approximation in each case. Figures 2(a) and 2(b) show the average and worst-case errors with approximation across all the data sets. The amount of error induced with reduced workloads is non-linear and is different in both the cases in (a) and (b). For a direct comparison, linear regression has a higher average case and worst-case errors when compared to that of least squares. The average and worst-case accuracy values are exclusively a function of input data and these numbers may vary across different datasets. We ran extensive experiments to show that across diverse input datasets, the average and worst-case errors are bounded by a tolerable accuracy loss (for example 3%). Moreover, in Figure 2(b), the error induced with further approximation from about 25% of skipped loops becomes less significant. The performance gains for both applications are expected to be proportional to the amount of relaxed workload. However, the normalized performance gained per error induced creates discrepancies in error resilience characteristics between applications. Precisely, for the same amount of comparable performance gain, least squares incur a smaller error than that of linear regression. As shown in Figure 2(a), normalized performance per error is sub-linear for linear regression, whereas it is super linear for least squares, assuming an error of 0.25% as a minimum. This example demonstrates the diversity among error resilience of different applications, an aspect to be considered for making appropriate accuracy trade-off decisions. For instance, in a dynamic workload scenario with both applications running, it is efficient to select least squares as the priority candidate for approximation. This could prompt approximating least squares while provisioning linear regression with other system resources such as *CPU Quota*. Therefore, exploiting approximation for performance/power management has to consider two aspects viz., (i) performance gained with approximation, which depends on the amount of relaxed workload, and (ii) error induced with approximation, which depends on input data and application characteristics. To enable such disciplined tuning and choice of accuracy configuration, we characterize the error resilience of applications through profiling and use these heuristics at run-time for appropriate resource allocation to maximize performance while minimizing inaccuracy and power consumption.

2.3 Example Scenarios

Knob actuation nuances, feasibility, and run-time system dynamics present complex scenarios for resource allocation decisions. We demonstrate such scenarios and possible knob actuation decisions through a motivation example in Figure 3. In this example, we run two parallel (App-1 and App-3) and one serial (App-2) applications with diverse characteristics on an HMP platform.[1]

Scenario (a): App-3 is running on the *LITTLE* cluster, and the power intensive App-1 and App-2 are running on the *big* cluster at high VF levels. Lowering the VF levels will degrade the performance of both applications. Hence, we reduce the VF level and simultaneously approximate App-1 proportionally to the performance loss and scale the *CPU Quota* of App-2 from 80% to 100%

[1]Experimental setup is presented in details in Section 5.

to improve its performance. DVFS reduces power consumption quadratically, creating a power headroom that can be utilized intuitively to scale the *CPU Quota* of App-2.

Scenario (b): App-2 is running on the *LITTLE* cluster, App-1 and App-3 are running on *big* cluster; power consumption of the system is high. Similar to scenario (a), lowering VF could degrade the performance of applications. Hence, we migrate App-1 to the low power *LITTLE* cluster with 3 idle cores and approximate the application proportionally to the performance loss. We select App-1 as the candidate for joint invocation of TM and approximation since it achieves higher performance gain with approximation than App-3. We lower the VF levels of the *big* cluster and use the power headroom created with TM and DVFS to scale out (DoP) the App-3 to 4 cores for mitigating performance loss due to lowering VF levels.

Scenario (c): All the cores are occupied, power consumption is under control, and App-1 is underperforming. The *LITTLE* cluster is running at full throttle VF levels, the *CPU Quota* of App-1 is at 100%, and TM has been recently invoked (scenario (b) – migrating App-1 to *LITTLE*), reflecting an exhaustion of invoking traditional knobs. In this scenario, we approximate App-1 by configuring the accuracy proportionally to meet the target performance.

Scenario (d): App-1 (approximated) and App-2 are running on the *LITTLE* cluster, App-3 has completed its execution, and the power consumption is under control. Completion of App-3 frees up cores on the *big* cluster and also creates power headroom to upscale resources. Hence, we migrate App-1 to the *big* cluster and re-configure the execution to accurate mode since the performance requirements of App-1 can be met with scaled-up resources on the *big* cluster without sacrificing accuracy.

2.4 Related Work

Run-time Resource Management. RTM techniques focus on objectives of minimizing power/energy consumption and maximizing performance within a power cap by using a combination of traditional knobs such as DVFS, TM, *CPU Quota* assignment, DoP for resource management (e.g., [2, 9, 12, 24]). State-of-the-art RTM techniques determine optimal resource configuration knob settings using (i) heuristics (e.g., [9, 12, 24]), or (ii) machine learning predictions (e.g., [1, 6, 13, 26]). Existing heuristic-based approaches have limited efficiency in handling unknown and dynamic workloads and inevitably sacrifice performance when power consumption and/or workload intensity is higher [2, 24]. Advanced RTM approaches built machine learning models using techniques such as imitation and reinforcement learning to predict optimal resource configuration settings (e.g., [17, 26]). However, these approaches rely on exhaustive offline profiling with high-design space exploration overhead. Recent strategies have used run-time data collection to build online learning models for determining resource configuration settings [1, 6], while some strategies have combined both offline and online learning models [13, 33]. RTM strategies train machine learning models on data where applications are run individually. However, when multiple applications are run concurrently, decisions made by the models for an application could become sub-optimal/infeasible and/or override the decisions made for other application(s). This behavior limits the efficiency of machine learning-based RTM strategies in handling unknown and concurrent workloads, as well as dynamic workload variation. Further, aforementioned RTM strategies are constrained by the limitations of traditional knobs with orthogonal power-performance characteristics.

A crucial aspect in the RTM decision process is the estimation of power and performance of any new configuration before enforcing it. Estimation models save the policy from the hassle of enforcing the knob actuation, monitoring the results, and fine-tuning the complete actuation

decision based on the monitored results. Recent works [15, 30, 31] have proposed machine learning models for predicting power, and performance against knob actuation. For instance, [30] uses a pre-trained Deep Neural Networks (DNN) model to predict the performance impact of actuating *Task Migration* of multi-threaded applications on *S-NUCA* many-core architectures. SmartBoost [31] uses online learning to predict the impact of processor frequency on application performance and device power. STAFF [15] also provides online learning-based power and performance prediction of unknown applications on heterogeneous multi-core architectures. These machine learning-based state-of-the-art power and performance prediction methods have design time training and run-time prediction overheads. We have designed lightweight models for fast estimation of power and performance values. Our estimation models provide heuristics-based hints to the RTM policy to make accurate knob actuation decisions without requiring highly accurate power and performance values.

Run-time Approximation. The run-time approximation has been widely used in resource-constrained systems in combination with traditional power knobs: to address performance loss incurred during power actuation [21, 38], for dynamic thermal management [7, 29], and to provide Quality of Service (QoS) guarantees [25]. Further, approximation benefits have been maximized through optimal scheduling of a combination of approximate-accurate tasks for performance guarantees [38] and thermal management [7]. However, all of these aforementioned approaches reactively toggle through a fixed number of approximation levels (levels 1-4) without coordination between power/performance/accuracy decisions. These approaches thus cannot fine-tune approximation as per run-time power/performance demands, consequently leading to significant accuracy loss with aggressive approximation. Capri [37] uses offline learning to determine fine-grained approximation levels by considering performance; however, this approach does not consider system-level resource allocation.

Recent works [23, 39–41, 43, 44] have also focused on RTM of DNN workloads on heterogeneous multi-core architectures. We did not consider DNN workloads for the proposed work because they restrict the actuation of the run-time approximation knob due to fixed weights and network design. Tango [40] explores the approximation of DNN inference on HMPs by selecting a model from a pool of 2-3 pre-trained models at run-time. However, this method does not allow run-time approximation of a single DNN kernel and is limited to DNN models released with sub-model variants of non-uniform numbers of parameters. Alternatively, we consider classical Machine Learning (ML) applications that provide fine-grain control over their power-performance characteristics against the joint action of the approximation and typical power knobs.

Table 1 summarizes the most relevant state-of-the-art RTM approaches, highlighting the novelty of our proposed strategy in fine-grained run-time approximation, coordinated power/performance decision-making, and handling of dynamic workloads. Some of these RTM techniques lack accuracy-awareness, while others use ad-hoc and static knobs for exploring a limited sub-set of accuracy trade-offs, typically for a given single application. On the other hand, our proposed approach combines generic fine-grained run-time approximation with traditional power knobs, by re-configuring applications' accuracy and DoP at run-time based on power/performance demands.

3 Resource Management Framework

Our resource management framework consists of interfaces between hardware, OS, and application layers, and a run-time controller to enforce decisions of the resource management policy. Figure 4 shows an overview of our resource management framework, elaborated in the following.

Hardware platform. We consider widely used ARM *big.LITTLE* asymmetric multi-core architecture as the baseline hardware [28]. It comprises two clusters – a set of high-performance

Table 1. Qualitative Comparison of the Proposed with Respect to the Existing Approaches

	Multi-program	Unknown workload	Knobs					Apx levels	Coordinated
			DVFS	CPU Quota	TM	Apx	DoP		
[25]	✓	✘	✓	✘	✓	✓	✘	2	+/-
[21]	✓	✓	✓	✓	✘	✓	✓	2	+/-
[7]	✓	✘	✓	✘	✓	✓	✓	2	+/-
[38]	✓	✘	✓	✓	✓	✓	✓	5	✘
[26]	✓	✓	✓	✘	✘	✘	✓	✘	+/-
[17]	✓	✓	✓	✘	✓	✘	✓	✘	+/-
[1]	✘	✘	✓	✘	✘	✘	✓	✘	+/-
[29]	✘	✘	✓	✘	✘	✓	✓	4	✘
[6]	✓	✓	✓	✘	✓	✘	✓	✘	+/-
[44]	✓	✘	✓	✘	✓	✘	✓	✘	✘
[43]	✓	✓	✘	✘	✘	✘	✘	✘	✘
[40]	✓	✓	✓	✘	✓	✓	✘	Config.	✘
our	✓	✓	✓	✓	✓	✓	✓	Config.	✓

✓: supported, ✘: not supported, +/-: partially supported.

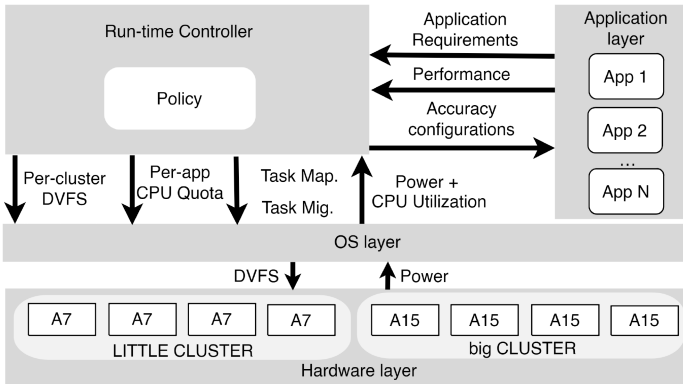

Fig. 4. The resource management framework.

power-hungry *big* cores and a set of low-performance power-saving *LITTLE* cores. Each cluster has an onboard power sensor for reading run-time power consumption and support for setting appropriate VF levels. The platform is constrained by Thermal Design Power (TDP), the conservative upper bound on power consumption for thermal safety.

Operating System. The OS running on top of the hardware platform is the Linux version targeted for big.LITTLE architecture, with a 3-layered scheduler featuring:

— *HMP scheduler* (also called Global Task Scheduling [3]) to migrate tasks (process/threads) between the *big* and *LITTLE* clusters, based on computational requirements,
— *Balancer* to uniformly distribute tasks among the ready queues of the cores of the same cluster, and
— *Priority-based round-robin algorithm* used on each core by picking tasks from the corresponding ready queue.

The OS exposes an interface to control the scheduler at run-time to orchestrate the execution of applications and resource allocation through the following mechanisms:

— *Application mapping* for binding the execution of an application/process on a specified set of cores, and
— *CPU Quota assignment* for setting a specific percentage of time for which the assigned cores are dedicated to executing an application/process.

The proposed policy enforces resource allocation decisions by overriding the Linux scheduler to map an application on a selected cluster (by modifying HMP scheduler) and a subset of cores within the cluster (by modifying the balancer) and to set the *CPU Quota* for the various applications (by modifying round-robin algorithm priorities). When mapping a parallel application, we set the number of threads always as equal to the number of assigned cores, thus from here on, we use application mapping to implicitly refer to DoP as well. Moreover, an exclusive mapping of at most one application per core is adopted, to guarantee execution isolation to achieve the best performance, as in [2, 35]. The OS exposes the driver interface to access the hardware sensors and knobs, and also enables reading per-core CPU utilization percentage, which can be implicitly translated to the assigned *CPU Quota*. Voltage and frequency levels are organized in pairs of fixed values. For DVFS knob actuation, the driver allows setting the frequency value alone while the corresponding voltage level is automatically selected.

Workload and applications. The considered workload is representative of many use cases of intelligent and smart edge systems, such as environmental monitoring, surveillance, and activity-based applications (e.g., health monitoring and fitness tracking); such applications execute multiple machine learning tasks that perform clustering, classification, and recognition processes on data streams from cameras or input sensors. In these contexts, the computing system is tightly connected with the surrounding environment; for instance, applications may be triggered by external events (e.g., perceived motion can initiate object tracking in a surveillance system), resulting in a variable and unpredictable workload; simultaneously, the working context influences the performance requirements for running applications (e.g., the number of moving objects or poor environmental lighting may necessitate additional processing efforts for object tracking).

According to this scenario, we characterize the workload to be composed of multiple applications entering and leaving the system in an unknown and unpredictable sequence, representing a high degree of variability. We consider embedded machine learning kernels with streaming input data sets, where the core compute-intensive block is repeated within a loop per every batch of inputs. Due to their streaming nature, we adopt *throughput in a given time window* as the primary performance metric. Applications' source code is assumed to be enhanced with the HeartBeat Application Programming Interface (API) [19], which measures the amount of data processed by an application in a given time window, represented as *heartbeats* per unit of time (hb/t). Figure 5 shows the general code skeleton of applications in our proposed framework. The application notifies a *heartbeat* at the end of each iteration of the application kernel loop (Line 9). Each application has user-defined throughput requirements in terms of minimum and maximum admissible levels (tp_min and tp_max) to guarantee the QoS; a throughput band is specified instead of a single target value to tolerate possible run-time performance fluctuations on embedded devices. Throughput requirements are registered by the application with a HeartBeat API call in the program preamble (Line 4). The last parameter is a lookup table used for configuring run-time approximation, which will be discussed in Section 4.1.

Software knobs and approximation. We use configuration of DoP and run-time approximation as software knobs for tuning the power/performance of multi-threaded approximable applications. We adapt the implementation of applications (as shown in Figure 5) such that software knob settings viz., (i) Degree of Parallelism (DoP) – number of threads to spawn, and (ii) the desired

```
1  int main() {
2    controller_t* c;
3    //... initialization phase ...
4    c = attachCtrl("applX", tp_max, tp_min, approx_tab);
5    //iterative processing of data chunks
6    for(i=0; i<num_of_chuncks; i++){
7      p = getSWParams(c);
8      applKernel(p.threadN, p.apxKnobValue);
9      sendHeartbeat(c);
10   }
11   detachCtrl(c);
12 }
```

Fig. 5. Application source code skeleton.

percentage of accuracy level are received at the beginning of the application main loop through the Heartbeat API (Line 7). The software knob settings are transmitted to the compute kernel (Line 8) as function parameters, enabling the application to self-configure at each execution cycle. The kernel function is implemented to spawn the required number of threads and to invoke application-specific approximation based on the received knob values (e.g., the number of loops to be skipped when loop perforation is used).

We introduced DoP as a new actuation knob (in comparison with our preliminary policy [22]), which increases the exhaustive resource configuration space with dynamic core folding and unfolding decisions. Dynamically varying DoP alters the run-time mapping configuration of the system – significantly affecting the performance of the other concurrently running applications, system-wide power consumption, and consequently, accuracy configuration levels of currently running applications. Hence, actuating DoP requires renewal of all the actuation decisions by gauging the power-performance-accuracy trade-offs.

Run-time controller. The run-time controller (Figure 4) is implemented as a process running on top of the OS to enforce resource management decisions of the policy. In this work, we adopt the run-time controller implementation presented in [22], which is similar to state-of-the-art RTM solutions [2, 13, 33]. The controller is connected to both the OS interfaces – to control the hardware and the OS mechanisms for task execution and applications – to monitor performance and control approximation through the Heartbeat library. Internally, the controller implements a feedback control loop, which is triggered at every *control cycle*. We implemented the *control cycle* as a parameterizable component [2, 22], to enable fine-tuning of the controller invocation for different workloads and platforms. In this work, we empirically set the *control cycle* to 1 second, based on the observed heartbeat rate (performance metric) of the benchmark applications. At each control cycle, the controller (i) monitors the status of the system (at hardware, OS, and application levels), (ii) decides how to control the system to pursue a set of system-level and application-level goals and requirements (e.g., applications' performance vs. system's power consumption), and (iii) actuates on knobs based on the decisions taken. The run-time controller enables the policy to make intelligent resource allocation decisions, followed by enforcement of the resource configuration settings determined by the policy through monitoring and feedback control.

4 The Proposed Policy

We here present the proposed resource management policy, which relies on run-time approximation and traditional power knobs. The decision-making strategy uses estimation models to

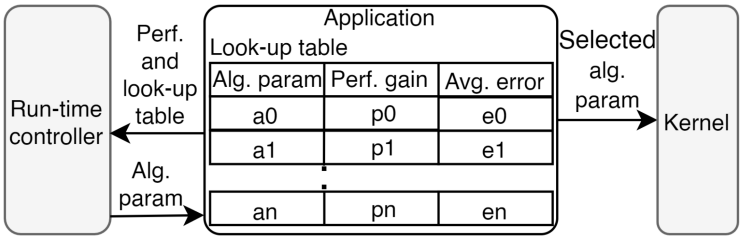

Fig. 6. The mechanism of application level approximation using a lookup table of performance gain at each approximation level.

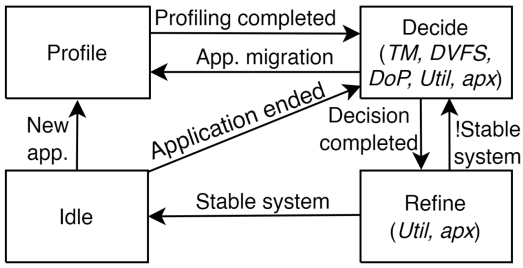

Fig. 7. FSM-based workflow of the designed policy.

coordinate decisions to: (i) meet applications' performance requirements, (ii) honor system's power budget, and (iii) minimize accuracy loss due to approximation.

4.1 Run-time Approximation Control

We define an empirical model that correlates performance gain and accuracy loss at different levels of fine-grained approximation for each application. To this end, every single application is profiled at design time (as described in Section 2.2) to measure the performance gain and the average-case error for relevant settings of the approximation knob w.r.t. the non-approximated golden run. Since the approximation error is data-dependent, multiple runs are executed for a given configuration by varying input data to measure the average-case error. The result of this design-time analysis is the pair of plots shown in Figure 2 that are specific for every single application. Such data are stored in a lookup table having the percentage error as the key, the performance gain, and the approximation knob value as values. As shown in Figure 6, the table is saved in the application code and transmitted to the controller during the initialization phase (through the `approx_tab` parameter at Line 4 in Figure 5). Then, the table is used in the policy to identify for every single application the entry having the desired trade-off between average-case error and performance gain (as will be discussed in Section 4.3) and the corresponding approximation knob value is sent back to the application to be enforced (Lines 7–8 in Figure 5).

4.2 Policy Workflow

The proposed policy is designed as an FSM with four states viz., (i) *Idle*, (ii) *Profile*,(iii) *Decide*, and(iv) *Refine* (Figure 7). Its workflow is described as follows.

Idle. The policy is in the *idle* state when the platform is *stable*, i.e., (i) initially when no applications are running on the platform, or (ii) performance requirements of the currently running application(s) are met, and the power budget is honored. When no applications are running, the policy reduces the static power consumption by setting the frequency of all the clusters to the

lowest available level. In this state, the policy continuously monitors relevant events of workload variation, which can perturb the *stable* system state. Primary workload variation events include (i) entry of a new application, (ii) termination of a currently running application, and (iii) migration of a currently running application between clusters. The policy reaches the optimal resource configuration settings by the end of the *Refine* phase and enters the *Idle* phase. The *Idle* phase implies that no additional resource optimization maneuvers are feasible. Although a shift in the workload scenario can make resource reallocation possible; However, this automatically transitions the policy out of the *Idle* phase to the *Profile* phase. Alternatively, when a currently running application terminates (or migrates between clusters), the policy transitions into the *decide* state for re-allocating resources freed up by the terminated/migrated application.

Profile. The policy enters the *profile* state upon encountering a workload variation event, viz., entry of a new application, or migration of a currently running application between clusters. The goal of the *profile* state is to measure an initial baseline throughput of an application when executed on a specific number of cores on a selected cluster. This profiling information will be used in the *decide* state for estimating the performance of the same application under potentially different resource configurations. The profiling activity lasts for a specific number of control periods to obtain a stable throughput measure. Newly arriving applications are initially mapped on the *LITTLE* cluster by default to minimize sharp rises in power consumption with an unknown workload. If no applications are currently running on the selected cluster and the frequency level is at the lowest (from the *Idle* state), DVFS is actuated to set a pre-defined profiling frequency level. When all the cores of the *LITTLE* cluster are already occupied by some applications at full CPU Quota, the new application is mapped on the *big* cluster. When no core(s) is available on both *LITTLE* and *big* clusters, the policy de-allocates one core of a currently running parallel application and allocates it for profiling the new application. A similar de/re-allocation of core(s) is used during the migration of an application between clusters, specifically when the target cluster is fully busy. Also, migrating an application between clusters may not require profiling, provided that the application was already profiled on a given cluster upon its entry.

Decide. The policy enters the *decide* state when: (i) a new application has arrived and been profiled, transitioning from the *profile* state, (ii) a running application ends, transitioning from the *idle* state, or (iii) a previously selected resource configuration violates either the power budget or performance requirement(s), transitioning from the *refine* state. The goal of the *decide* state is to determine a resource configuration that can maintain/restore the system's *stable* condition, i.e., performance requirements met and power budget honored. The policy determines this resource configuration through a heuristic algorithm (detailed in Section 4.3), which jointly actuates application-to-core mapping, per-cluster DVFS, application migration between the two clusters, per-application *CPU Quota* assignment, per-application DoP, and per-application approximation.

Intuitively, the policy enters the *decide* state when an application performance requirements are not met or when the power budget is violated. The heuristic algorithm initially attempts to tune traditional power/performance knobs to recover the *stable* condition of the system. When the traditional knobs have reached their limitation (i.e., performance requirements cannot be met without potential power violation, or power budget cannot be honored without performance degradation), the algorithm opportunistically invokes run-time approximation. The run-time approximation knob is invoked as a last resort when all the traditional knobs are exhausted. We have designed the policy to quickly recover an application from approximation to avoid further accuracy loss. The decision strategy uses power/performance models (presented in detail in Section 4.4) to estimate the resultant throughput and power consumption of a selected resource configuration. Power/performance estimation models are commonly adopted in the literature

(e.g., [2]) to prune resource configuration space in a single control period – for evaluating other possible alternative knob settings without physically enforcing them. The power/performance models take instantaneous power consumption, per-application throughput (measured during the *profile* state), and an arbitrary resource configuration as inputs and return the estimated performance and power consumption with the selected resource configuration. The heuristic algorithm determines a suitable resource configuration by iteratively tuning the knob settings using the estimation models. The policy finally enforces the selected resource configuration knob settings. Moreover, each cluster that is not running any application is put to its minimum frequency level to reduce static power consumption. If the updated resource configuration includes a migrating application not profiled previously on the target cluster, the policy transitions to the *profile* state – to iterate on the throughput measurement and decision process. Otherwise, the policy transitions to the *refine* state to fine-tune the selected resource configuration.

Refine. The working configuration identified during the *decide* state has been defined in a coarse-grained way by using estimation models. The goal of the *refine* state is, therefore, to perform a fine-grained tuning of such a configuration in a closed-loop fashion by exploiting the available performance sensors and throughput monitors. This state deals with individual performance over/under-compensation of the running applications. We use *CPU Quota* and approximation as the actuation knobs for fine-tuning. The policy mitigates performance over/under-compensation by scaling the *CPU Quota* proportionally to the performance requirements. If the application is already running at 100% CPU utilization, the policy invokes run-time approximation to meet the performance requirements. The precedence of *CPU Quota* followed by the approximation knob is implicitly aimed at minimizing the possible overall accuracy loss. The approximation level is scaled at the granularity of a *step* that is specifically defined within an application, while *CPU Quota* can be scaled in steps of 1%. If the power budget is violated during the tuning, the policy returns to the *Decide* state to identify a new working configuration. Finally, the policy moves to the *Idle* state when all performance requirements are met without exceeding the overall power budget. As a final note, we assume a single application to enter or leave the system at a time, and a subsequent event to occur only after the policy is returned to the *idle* state. Nonetheless, the proposed approach can handle subsequent event "overlaps". If a new application arrives and is still servicing a previous application, the workflow restarts from the *profile* state, while the termination of another application requires to re-perform the *decision* state. The policy executed the applications in isolation and each application thread is mapped on a single core to exploit the full compute power of the multicore architecture and to avoid incurring performance penalties due to resource contention. Allocating one core to multiple application threads causes additional scheduling overhead without ensuring any performance gain [35]. Moreover, we consider the system as overloaded when a new application arrives and all the processing resources are preoccupied. In such cases, our policy does not reject the new application and allocates the busy cores to multiple applications, which may degrade the performance of all the running applications.

4.3 The Decision Strategy

The algorithmic flow of determining and enforcing knob actuation settings in the *Decide* phase of the policy is presented in Algorithm 1. The algorithm initially calculates the required performance $\texttt{perf}_{\texttt{req}}$ of an application as the average of the target performance range, based on the specified target performance range [`tp_min` and `tp_max`], as mentioned in Section 3 (Lines 1–2). Then, the decision strategy modularly handles system-wide power violations (Lines 3–15) and per-application performance violations (Lines 17–50). Decisions are taken by exploiting the power and performance estimation models later discussed in Section 4.4.

ALGORITHM 1: Decide phase workflow.

Inputs: Apps: List of currently running applications
Variables: U: Total utilization of all the cores, P: Power consumption
Constants: TDP: Power budget limit, MIG_LOSS: Task migration performance loss threshold
Body:

```
1: for (app_i ∈ Apps) do
2:     app_i.perf_req ← (app_i.tp_min + app_i.tp_max)/2
3: if (P ≥ TDP) then
4:     for app_i in Apps ∈ big do
5:         perf_Loss ← app_i.perf_LITTLE / app_i.perf_req
6:         if (perf_Loss ≤ MIG_LOSS) then
7:             set_APX(app_i, perf_Loss)
8:             decide_TaskMigration(app_i, LITTLE)
9:             enforce_knobs_and_exit(Profile); //Exit current decision phase
10:    f_temp ← (TDP/P) * f_curr
11:    decide_DVFS(f_temp, big)
12:    for (app_i in Apps ∈ big) do
13:        app_i.perf_est ← estimate_Perf(F_temp, app_i.Q)
14:        if (app_i.perf_est < tp_min) then
15:            set_APX(app_i, app_i.perf_est / app_i.perf_req)
16: else
17:    Apps.sort (perf_curr/Q)
18:    dvfs_Done ← false
19:    for (app_i ∈ Apps) do
20:        f_est ← (app_i.perf_req / app_i.perf_curr) * f_curr
21:        if (app_i.perf < app_i.tp_min) then
22:            if (app_i.is_Parallel && (app_i.cluster.free_Cores > 0)) then
23:                decide_DoP(app_i.cluster.free_Cores)
24:            else if (estimate_Power(f_est) < TDP && !dvfs_Done) then
25:                decide_DVFS(f_est, app_i.cluster)
26:                dvfs_Done ← true
27:                for (app_j ∈ app_i.cluster) do
28:                    app_j.perf_est ← estimate_Perf(f_est, app_j.Q)
29:                    app_j.perf_gap ← app_j.perf_est / app_j.perf_req
30:                    if (app_j.perf_est < app_j.tp_min) then
31:                        if (app_j.Q < 1) then
32:                            set_CPUQuota(app_j, app_j.perf_gap)
33:                        else
34:                            set_APX(app_j, app_j.perf_gap)
35:                    else if (app_j.perf_est > app_j.tp_max) then
36:                        if (app_j.curr_Apx_Level > 0) then
37:                            set_APX(app_j, app_j.perf_gap)
38:                        else
39:                            set_CPUQuota(app_j, app_j.perf_gap)
40:            else if ((app_i ∈ LITTLE) && (free_Cores_Big>0)) then
41:                if (estimate_Power((TDP/P) * f_curr, app_i.Q) < TDP) then
42:                    decide_TaskMigration(app_i, big)
43:                    enforce_knobs_and_exit(Profile);
44:            else
45:                set_APX(app_i, app_i.perf_curr / app_i.perf_req)
46:        else if (app_i.perf_curr > app_i.tp_max) then
47:            if (app_i.curr_Apx_Level > 0) then
48:                set_APX(app_i, app_i.perf_curr / app_i.perf_req)
49:            else
50:                set_CPUQuota(app_i, app_i.perf_curr / app_i.perf_req
51: enforce_knobs_and_exit(Refine);
```

Power Violation. In case of a power violation, the policy prioritizes recovery actions on the *big* cluster, contributing significantly to the overall power consumption. The policy initially considers `TaskMigration` of an application from the *big* cluster to the *LITTLE* cluster, subject to the availability of free cores on the *LITTLE* cluster (Lines 4–9). We define a heuristic parameter `MIG_LOSS`,

which represents the threshold of performance degradation when an application is migrated from *big* to the *LITTLE* cluster. MIG_LOSS is a design variable that we set based on the performance gain ceiling of the considered applications at the *LITTLE* cluster using approximation (in our case we set MIG_LOSS threshold as 10%). Hence, we ensure that the performance loss due to migration to *LITTLE* cluster can be minimized using approximation. Among the currently running applications, the policy calculates the performance loss with task migration perf_Loss as the ratio of maximum performance on the *LITTLE* cluster (recorded during *Profile* phase) to the required performance $\texttt{perf}_{\texttt{req}}$. The policy then chooses an application whose performance loss upon task migration is less than the MIG_LOSS threshold (Line 6). The policy proactively approximates the migrating application (joint actuation of TM+APX) to compensate for the potential performance loss incurred in migrating from *big* to *LITTLE* cluster. The policy sets the approximation level of the application as proportional to the calculated performance loss (Line 7). Finally, the policy enforces the knob actuation settings and transitions to the *Profile* state to record the application performance on the LITTLE cluster (Lines 8–9).

"When a new application arrives and the *LITTLE* cluster is busy, the policy cannot profile its performance on the *LITTLE* cluster. For each new application, we initialize the default value of MIG_LOSS to 100. This high value of MIG_LOSS ensures that *Task Migration* fails for any application without profiling data on the *LITTLE* cluster. When *Task Migration* fails in the absence of an ideal candidate, and the policy jointly actuates DVFS and application approximation to reduce power violations." The policy actuates DVFS by scaling down the current CPU frequency $\texttt{f}_{\texttt{curr}}$ of the *big* cluster as proportional to the available power headroom $\frac{\texttt{TDP}}{\texttt{P}}$. Actuating DVFS on the *big* cluster based on one application can impact the performance of the other applications running on the same cluster. Hence, the policy estimates the possible performance degradation of all the applications running on the *big* cluster through the performance estimation model from Equation (3) (Lines 12–13). If the estimated performance $\texttt{perf}_{\texttt{est}}$ of an application with the updated frequency levels is lower than $\texttt{tp}_{\texttt{min}}$, the policy approximates the application as proportional to the ratio of the estimated performance $\texttt{perf}_{\texttt{est}}$ to the required performance $\texttt{perf}_{\texttt{req}}$ (Lines 14–15). Finally, the policy enforces the updated DVFS settings on the *big* cluster and exits the current decision cycle to enter the *Refine* phase (Line 51).

We empirically observe that power consumption on the chosen embedded platforms (Asus Tinker R and Odroid-XU3) is largely a function of cluster and frequency level (as also discussed in [2, 22]). Further, application-specific knobs viz., *CPU Quota* result in relatively higher performance degradation with power actuation (also shown in Figure 1), while *Approximation* incurs accuracy loss. Hence, in case of power violations, our strategy sets the knob actuation precedence in the order of TM, DVFS, DoP, *CPU Quota*, and approximation.

Performance Violation. When there is no power violation, the policy exclusively handles applications' performance. The policy sorts applications in ascending order of utilization, i.e., the performance achieved per *CPU Quota* allocated (Line 17). The target cluster frequency $\texttt{f}_{\texttt{est}}$ is determined (Line 20) based on the application with worst-case utilization, Apps[0]. Indeed, tuning the cluster frequency based on the application with the worst utilization minimizes overall power consumption. The policy sets the dvfs_Done flag as *false* at the start of the decision cycle, indicating that no DVFS actuation has yet been applied (Line 18).

Within the sorted list, the policy first handles the *under-performing* applications, i.e., applications with measured performance $\texttt{app}_{\texttt{i}}\texttt{.perf}$ below the required performance $\texttt{tp}_{\texttt{min}}$ (Line 21). For parallel applications, the policy finds free cores on the given application's cluster to increase the application's DoP (Lines 22–23). We do not use any prediction model for DOP as it shows non-linear performance behavior against the allocated CPU cores and has application-specific power

and performance characteristics [36]. The policy invokes the DoP knob to allocate free cores to the application and transitions to the *Refine* phase (Line 51). If no free cores are available, the policy actuates the DVFS knob to address the performance gap. Before actuating DVFS, the policy determines the potential power consumption with the estimated frequency f_{est} settings, using the power estimation model from Equation (1) (Line 24). If the estimated power is less than TDP and DVFS knob has not been invoked in the current decision cycle, the policy decides to actuate the DVFS knob and sets the dvfs_Done flag as true (Lines 25–26). DVFS is a cluster-wide knob which affects the performance of all the applications that are running on a given cluster. Therefore, the policy estimates the performance $perf_{est}$ (from Equation (3)) of all the applications running on the cluster with the updated frequency settings (Lines 27–39).

For each application, the policy calculates the performance gap $perf_{gap}$ as a ratio of the estimated performance $perf_{est}$ to the required performance $perf_{req}$. For applications with estimated performance $perf_{est}$ less than the required performance $perf_{req}$, the policy first invokes the *CPU Quota* knob to increase the quota Q proportional to the estimated performance gap $perf_{gap}$ (Lines 31–32). If actuating *CPU Quota* is not feasible (i.e., Q is already 100% or increasing Q results in TDP violation), the policy invokes the approximation knob. The policy refers to the approximation look-up table (Figure 6) to select the least possible approximation level which provides enough performance to address the performance gap $perf_{gap}$ (Line 34). For applications with estimated performance $perf_{est}$ greater than the required performance $perf_{req}$ (Line 35), the policy scales down resources to provide performance precisely within the requirements. In this case, the policy prioritizes recovering from approximation, followed by lowering *CPU Quota* as proportional to the performance gap $perf_{gap}$ (Lines 36–39).

If DoP and DVFS have reached their limitations and the under-performing application is running on the LITTLE cluster, the policy considers TaskMigration to the *big* cluster (Lines 40–43). The policy estimates the power headroom available for migration using the power estimation model from Equation (1) (Line 41) and migrates the application to the *big* cluster (Lines 42–43). The policy transitions to the *Profile* phase to record the performance of the migrated application with the updated mapping configuration (Line 43). If all the power knobs have reached their limitations, the under-performing application is approximated proportional to the ratio of the current performance $perf_{curr}$ to the required performance $perf_{req}$ (Line 45).

For *over-performing* applications, the policy actuates either approximation or *CPU Quota* to scale down the performance to the required level. It follows the precedence of invoking approximation (to minimize the accuracy loss), followed by *CPU Quota* downscaling (Lines 47–48). For an application that is already approximated, it sets the accuracy level proportional to the ratio of the current performance $perf_{curr}$ to the required performance $perf_{req}$ (Line 48). In case the application is not approximated, the policy scales down the *CPU Quota* proportional to the ratio of the current performance $perf_{curr}$ to the required one $perf_{req}$ (Line 50). Finally, the policy moves into the *Refine* phase (Line 51).

4.4 Power and Performance Models

The policy adopts analytical power and performance models to estimate system-wide power consumption and per-application performance. The models used here have been borrowed from the literature (e.g., [2, 22, 38]) and experimentally tuned on the selected target architecture. These estimation models enable the policy to predict the potential impact of a selected resource configuration without actually enforcing the decision. This allows the policy to pro-actively fine-tune resource allocation decisions, avoiding the penalty of reactively oscillating between resource over/under-provisioning. Some of the existing strategies focus exclusively on accurately predicting the power and performance values using pre-trained ML models [16, 32]; however, achieving such prediction

accuracy is not our focus in this work. We considered heuristics-based estimation models for predicting power and performance, given their lower design and run-time overheads than ML based solutions. It should be particularly noted that the estimation models are not our policy's decisions; rather our RTM policy uses the estimation models only as intermediate heuristics to fine-tune the actual RTM decisions towards optimality.

Power estimation. It computes the power consumption of each cluster i as a function of the configured frequency level, f_i (and corresponding voltage level set by the OS), and the CPU utilization with the applications that are mapped on that cluster, U_i. Estimated power is expressed as:

$$P_i = a(f_i) \cdot U_i + b(f_i) \tag{1}$$

where $a()$ and $b()$ are two tabulated functions defining for each frequency level of cluster i a pair of corresponding empirically derived coefficients. Moreover, since the CPU utilization of the cluster is also unknown in advance, it is estimated as the sum of the *CPU Quota* (Q_j) that will be set to each one of the n_i applications we aim at running on cluster i. Estimated utilization is expressed as:

$$U_i = \sum_{j=0}^{n_i} Q_j \tag{2}$$

It is worth noting that the *CPU Quota* assigned of the single application Q_j intrinsically captures also the information on how many cores the application is using; in fact, $Q_j = 1$ represents a single core completely used, while larger values represent the assignment of multiple cores. The overall power is estimated as the sum of individual per-cluster power consumption.

Performance estimation. A rough performance model generally used in the literature for iterative closed-loop approaches, as the one here presented, considers the application throughput to be a linear function of the assigned *CPU Quota* and cores' frequency; as discussed above, the number of cores used by the application is included in the *CPU Quota*. Thus, for a given application j running at given cores' frequency f_{j_old} and *CPU Quota* Q_{j_old} and having a measured throughput equal to tp_{j_old}, we can estimate the new performance value tp_{j_new} when varying the frequency to f_{j_new} and *CPU Quota* to Q_{j_new} as:

$$tp_{j\_new} = \frac{f_{j\_new} \cdot Q_{j\_new}}{f_{j\_old} \cdot Q_{j\_old}} \cdot tp_{j\_old} \tag{3}$$

In previous work [2], this model is also extended with a factor to estimate the performance variation when migrating the application to a different cluster. We do not adopt such an approach since it presents a quite low accuracy; we rather prefer to rerun the profiling after application migration.

5 Experimental Evaluation

5.1 Experimental Setup

Controller prototype. We have implemented the proposed RTM framework as a middleware in C++, with modular blocks of the run-time controller, resource allocation policy, monitors, and OS interface. The RTM framework runs as a user-space process in a Linux OS environment. It uses Linux drivers to access onboard power sensors and set DVFS levels, and the CGroups library to enforce task/thread-to-core mapping and *CPU Quota* assignment. We enhance applications with the *HeartBeat* library for expressing high-level performance requirements and measuring run-time performance and integrate the *HeartBeats API* with the controller. The framework uses a Linux-shared memory mechanism to enable communication between the controller and each running

Table 2. Characteristics of the Selected Benchmark Applications

Application	Exec. model	Input	Approximation technique	Odroid XU3		Tinker Edge R	
				TPmin (hb/s)	TPmax (hb/s)	TPmin (hb/s)	TPmax (hb/s)
knn	Parallel	25k points/25 test cases	loop perf. and task skip	2	4	6	8
fe	Parallel	204k points	loop perf.	11	15	9	10
thd	Serial	204k points	loop perf.	3	7	5	7
kM	Parallel	50k points/3 clusters	relaxed conv.	2	4	4	6
lesq	Parallel	1M pairs	loop perf.	3	5	4	5
lr	Serial	1M points	relaxed conv.	3	6	4	5

application. This enables the controller to read applications' performance requirements and approximation lookup table, measure run-time performance, and set approximation and DoP levels of the application at run-time. The controller runs in a continuous loop and is invoked over a configurable *control period*. For experimentation, we set the *control period* to 1s and the number of iterations for the *profile* state to 5s. Finally, it is worth noting that the programming logic of the proposed RTM controller is portable; to deploy on a target board it requires only minor changes to the strings containing the names and paths of the files referring to the specific hardware drivers.

Platforms. We evaluate the proposed controller on two widely used embedded platforms viz., Odroid XU3 [18] and Tinker Edge R [5]. Both these platforms expose diverse sets of core-level heterogeneity and power/performance characteristics, allowing rigorous evaluation of our proposed RTM strategy. The Odroid XU3 [18] board hosts Samsung Exynos 5422 SoC, integrating an ARM *big.LITTLE* processor with 4 ARM A7 cores (*LITTLE* cluster) and 4 ARM A15 cores (*big* cluster). The *big* and *LITTLE* clusters operate over frequency levels of 200MHz - 2000MHz and 200MHz - 1400MHz, respectively, with the possibility of setting frequency levels in steps of 100MHz. To avoid thermal violations, we set the TDP to 5*W* [33]. As for the operating system, we used Linux Ubuntu 20.04. The Tinker Edge R board hosts a Rockchip RK3399Pro SoC with hexacore ARM *big.LITTLE* architecture, which includes 4 ARM A53 cores (*LITTLE* cluster) and 2 ARM A72 cores (*big* cluster). The *big* and *LITTLE* clusters allow a 200MHz step frequency settings with respective frequency levels of 400MHz - 1400MHz and 400MHz - 1800MHz. We also set the TDP of 5W for the Tinker board. We used Linux Debian 10 as the operating system, which is the latest distribution supported by Tinker. We deployed our proposed RTM framework on both the aforementioned platforms. We measured the standalone computational overhead of the run-time controller as 1.43% on one *LITTLE* core of Odroid XU-3 and 1.82% on one *LITTLE* core of Asus Tinker Edge R platforms.

Workloads. We selected data-intensive micro-kernels that are widely used in machine learning pipelines and signal/image processing as workloads for evaluation. These include: (i) Least Squares (lesq) [10], (ii) K-Nearest-Neighbours (knn) [10], (iii) kMeans (kM) [8], (iv) Linear Regression (lr) [8], (v) Feature Extraction (FE) [42], and (vi) Total Harmonic Distortions (THD) [4]. We adapted and re-implemented the original source of the benchmark applications to support configurable run-time approximation, and thread-level parallel execution using *pthreads*. We consider streaming applications that repeat their execution in a brief cycle, and their execution phase variations are collected during the Profile phase. We set the profiling duration to 5 seconds, sufficient to capture the applications' power and performance metrics. Table 2 summarizes the characteristics of the benchmark applications, and approximation techniques used for each application. The approximation table of each application is defined once at compiled time after offline profiling of the error percentage against the performance gain. Performance measured in terms of *hb/s* is often prone to minor fluctuations, leading to a misrepresented enumeration

of performance being met versus not met. To avoid making unnecessary reactive decisions on performance, we define the target performance as a range between minimum throughput TP_{min} and maximum throughput TP_{max}. The policy parses the required performance as the average of TP_{min} and TP_{max}; this avoids any possible false positive triggers on performance being met or not met. TP_{min} is determined by running the application standalone on the big cluster at 80% of the max frequency and TP_{max} at the max frequency. We ran 3-4 applications simultaneously to mimic the scenario of a practical mobile workload use case. In real-world use cases, the number of foreground high-performance user-space applications is not more than 3-4 applications.

Evaluation metrics. We consider, for each application: (i) *rate of performance requirements being met*, i.e., percentage volume of time during which the throughput is in the $[TP_{min}, TP_{max}]$ range w.r.t. the overall execution time, (ii) *average power consumption*, and (iii) *accuracy loss*, i.e., average accuracy degradation accumulated during its execution.

Comparison w.r.t. state-of-the-art approaches. We compare our proposed approach against two state-of-the-art resource management strategies for HMP viz., *AdaMD* [6] and *DAgger* [26]. Both *AdaMD* and *DAgger* are highly relevant for comparison since they encompass: run-time adaptability for unknown workloads, optimal DVFS and DoP configuration, and meeting performance requirements within power and/or energy constraints. *AdaMD* and *DAgger* collect extensive offline profiling data to train machine learning models to predict DVFS and DoP settings while considering performance and energy metrics. We adapted *AdaMD* and *DAgger* to our proposed framework by (i) enhancing their implementation with Heartbeats API, (ii) configuring the policies to meet performance requirements within an acceptable range, and (iii) considering potential TDP violations. We performed extensive offline profiling to collect each application's power and performance numbers at different resource configurations to gather the training data for the state-of-the-art strategies. Linux versions on both Odroid XU-3 and Tinker boards provide access to standard governors including *Performance, Powersave, Ondemand, Conservative, and Interactive*. We compare our approach against the most relevant *Ondemand* (*HMP_O*) and *Interactive* (*HMP_I*) Linux governors as a standard baseline. These governors monitor CPU usage statistics over a given window to scale CPU frequency proportionally, with *Ondemand* governor reacting to CPU busy time and *Interactive* governor being driven by load intensity.

5.2 Experimental Results

We evaluate our approach against the aforementioned strategies over different dynamic workload scenarios, selecting kernels from the benchmark applications in Table 2. For practical feasibility, our experiments include (a) running two applications concurrently and (b) running three applications concurrently. In both cases, we use a combinatorial mix of serial and parallel workloads to create diverse workload variations. Finally, we also experiment with (c) an ensemble run, i.e., running multiple (ranging between 1-6) randomly selected applications concurrently. We run the experiments on both platforms including Odroid XU3 board and Tinker Edge R board.

5.2.1 Experiments on Odroid XU3 Board. We performed different experiments by running two, three, and multiple concurrent applications. The three experiments are discussed in the following.

(a) Running 2 applications concurrently. In this scenario, the second application dynamically arrives 20 seconds after initializing the first application. We experiment with eight different workload combinations, covering a mix of serial and/or parallel applications (viz., 2 serial apps, 2 parallel apps, 1 serial and 1 parallel apps) running concurrently.

Figure 8(a) shows the performance requirements met (%) for the mentioned eight workload combinations and for all the chosen strategies. It should be noted that the proposed policy meets

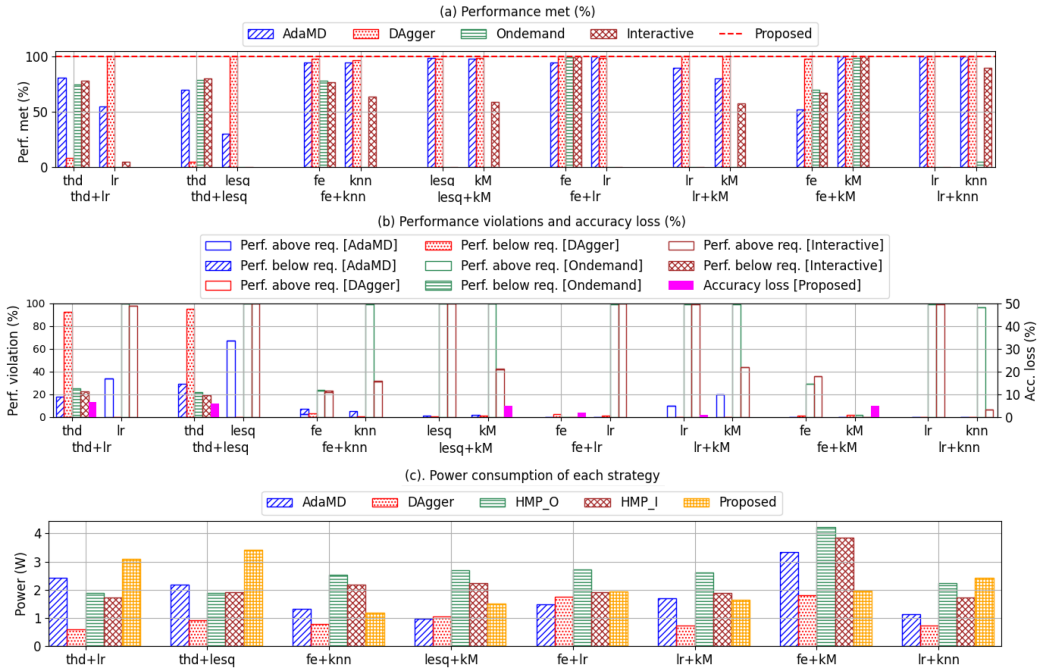

Fig. 8. Comparison of the performance violations, performance met, and accuracy loss for all five strategies when two applications are running concurrently in eight different combinations.

performance requirements in all the workload combinations (shown in red horizontal dashed line). In contrast, other relevant strategies fail to meet the performance requirements (either over-compensate or under-compensate) when handling compute/power-intensive workload combinations.

AdaMD fails to meet the performance requirements of application(s), particularly in the workload combinations of *thd+lr*, *thd+lesq*, *lr+kM*, *fe+kM*. In the first two cases *AdaMD* allocates cores on *big* cluster based on offline profiled data, indicating that *big* cluster is optimal for *thd*, *lr*, and *lesq* applications when running standalone. *AdaMD* thus maps all the applications on the same cluster, thus leading to resource contention, particularly with the compute-intensive *thd* application affecting the performance of other applications running on the same cluster. In this scenario, scaling DVFS and DoP to cater to the under-performing application is not feasible with power constraints, resulting in performance violations. *DAgger* meets the performance constraints of the applications in most workload combinations. However, in *thd+lr* and *thd+lesq* workload combinations, *DAgger* meets the performance requirements of *thd* only for 5% of the total execution time. In both these combinations, *DAgger* maps *thd* to the *big* cluster to meet its high-performance requirements. However, it applies DVFS settings as per *lr* to achieve energy conservation, which leads to performance violations of *thd*.

Both the standard governors aggressively scale the CPU frequency and reactively update the mapping configuration to meet the performance requirements. This results in performance overcompensation in most workload combinations, as shown in Figure 8(b). However, the performance is below the requirements with power-hungry workloads such as *thd+lr* and *thd+lesq*. In these cases, aggressive DVFS is warranted by TDP constraint, demonstrating the limitations of exclusively using traditional power knobs.

In the aforementioned intensive workload combinations, our proposed strategy can meet the performance of *thd* by reducing the frequency level to avoid power violations and by approximating *thd* at a 6% accuracy loss. The effectiveness is further demonstrated in Figure 8(b), showing the performance violations (performance below TP_{min} and above TP_{max}) achieved with other strategies versus accuracy loss due to approximation with the proposed strategy. Our policy opportunistically trades off accuracy at a minimal level, coordinating power/performance decisions to prevent performance violations.

Figure 8(c) shows the average power consumption with different strategies for the selected workload combinations. Both *AdaMD* and *DAgger* have relatively lower power consumption in general due to conservative resource allocation based on the first arriving application. Both governors have relatively higher power consumption with reactive frequency scaling to meet the performance requirements. Our strategy has relatively higher power consumption since it exploits the available power budget to meet the performance requirements within the TDP limit.

For example, *AdaMD* and *DAgger* incur lower power consumption for *thd+lr* and *thd+lesq*, causing significant performance violations. In the same workload combinations, the proposed approach is characterized by relatively higher power consumption, still within the TDP constraint, while meeting the performance requirements. In other workload combinations such as *fe+kM*, *AdaMD* has higher power consumption by running both the applications on the same cluster, while both the governors also have higher power consumption by increasing the frequency to compensate for the performance. In this case, the proposed approach combines task migration and approximation on *kM*, meeting performance requirements while minimizing power consumption.

(b) Running 3-applications concurrently. In this scenario, the first application starts at t=0, the second and third ones arrive at t=20s, and t=40s, respectively. We experiment with eight different workload combinations (viz., 2 serial and 1 parallel app, 2 parallel, and 1 serial app) running concurrently, creating diverse workload scenarios of variable power and performance demands. Figure 9(a) shows the performance requirements met (%) for individual applications in each workload combination with different strategies. Our proposed policy meets the performance requirements in all of the combinations (shown in red dashed line). In contrast, the other strategies can only meet the performance constraints of two combinations viz., *fe+knn+lr* and *lr+lesq+kM*. Both *AdaMD* and *DAgger* are unable to meet the performance of multiple applications in the workload combinations *knn+thd+lesq*, *fe+thd+lesq*, *fe+thd+kM*, *fe+kM+lr*, and *fe+lr+lesq*. Also, the standard governors fail to meet the performance requirements of at least one application in most workload combinations. It should be noted that the governors typically over-compensate for one application on an arbitrary cluster while inevitably degrading the performance of the application(s) on the other cardinal cluster due to TDP constraints. With *AdaMD* and *DAgger*, the first arriving parallel application is mapped on the *big* cluster, while the second and third ones are mapped on the *LITTLE* cluster. This generally leads to performance degradation of subsequently arriving application(s). Specifically, the subsequently arriving *thd* application suffers (e.g., *knn+thd+lesq*, *fe+thd+lesq*, and *fe+thd+kM*), since the *LITTLE* cluster is inadequate in meeting its performance requirements. In this scenario, the proposed policy (i) maps *thd* application on *LITTLE* cluster and approximates 3-6% to cater the performance requirements, (ii) migrates it to the *big* cluster when free cores are available and restores accuracy levels, and (iii) actuates DVFS to ensure that *thd* performance is met under the power budget.

With an intensive workload of three concurrent applications, our approach exploits accuracy trade-offs to meet performance requirements under power constraints, while the other strategies inevitably degrade the performance of one or more applications. Figure 9(b) shows for each workload combination the performance violations (%) for the different strategies versus accuracy loss

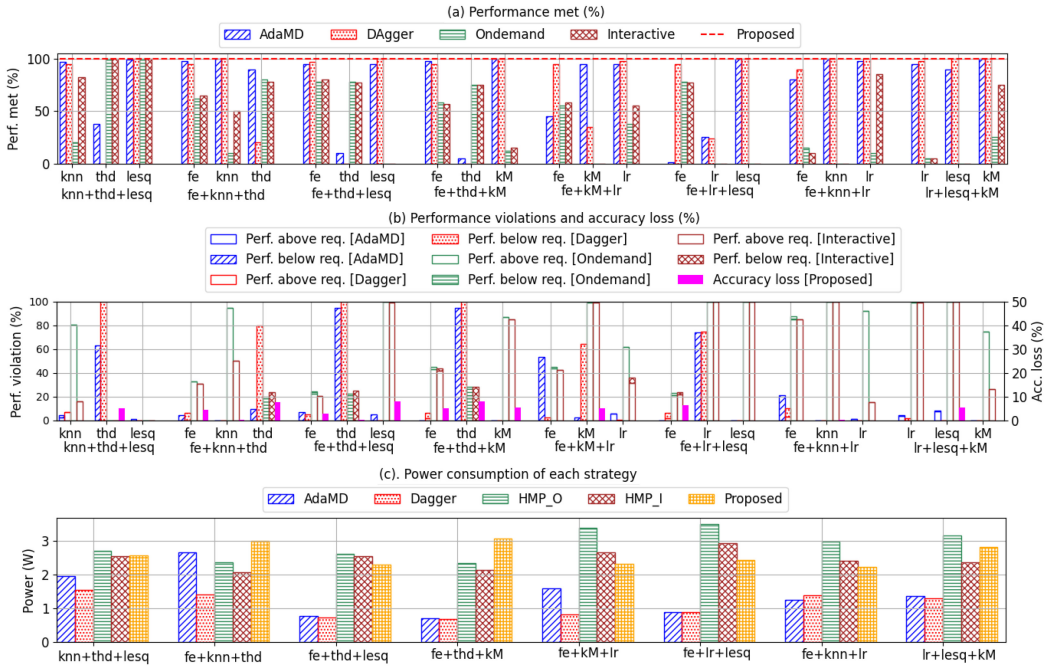

Fig. 9. Comparison of the performance violations, performance met, and accuracy loss for all five strategies when three applications are running concurrently in eight different combinations.

for the proposed approach. Figure 9(c) shows the power consumption of each strategy. The proposed approach efficiently uses the available power budget to meet the performance requirements without causing any violation. Both *AdaMD* and *DAgger* have relatively lower power consumption guided by offline profiling, however, resulting in significant performance violations. The standard governors, on the other hand, have a higher power consumption with reactive DVFS scaling, yet resulting in both performance over- and under-compensation.

(c) Running multiple concurrent applications. In this scenario, we created a dynamic workload by concurrently running 1-6 randomly selected applications. Figure 10(a) shows the run-time performance of each application with different strategies, along with the minimum and maximum performance requirements (red dashed lines). For the proposed strategy, the execution period where the approximation is dynamically invoked is also highlighted.

The workload begins at t=0s with the entry of *knn* application, which is mapped on the *big* cluster by all strategies based on performance demands. At t=45s, *fe* arrives, and all strategies map it on the *LITTLE* cluster with free cores. While the other strategies violate the performance requirements of *fe* on the *LITTLE* clusters, the proposed strategy proactively approximates *fe* (shown in the pink shaded box in Figure 10(a)) to meet the performance requirements. At t=60s, *knn* finishes execution, freeing up resources on the *big* cluster. At this point, the proposed strategy migrates *fe* to the *big* cluster and recovers *fe* from approximation to restore accurate execution. Similarly, *AdaMD* maps *fe* on the *big* cluster to achieve performance, while *fe* continues under-performing on the *LITTLE* cluster for *DAgger* because it does not support task migration. At t=80s, *thd* arrives; the proposed strategy and *AdaMD* map it on the *LITTLE* cluster, while *DAgger* maps it on the *big*. The proposed policy once again invokes approximation to meet the performance requirements of *thd*, while *AdaMD* violates the performance requirements. At t=105s, *fe* completes execution

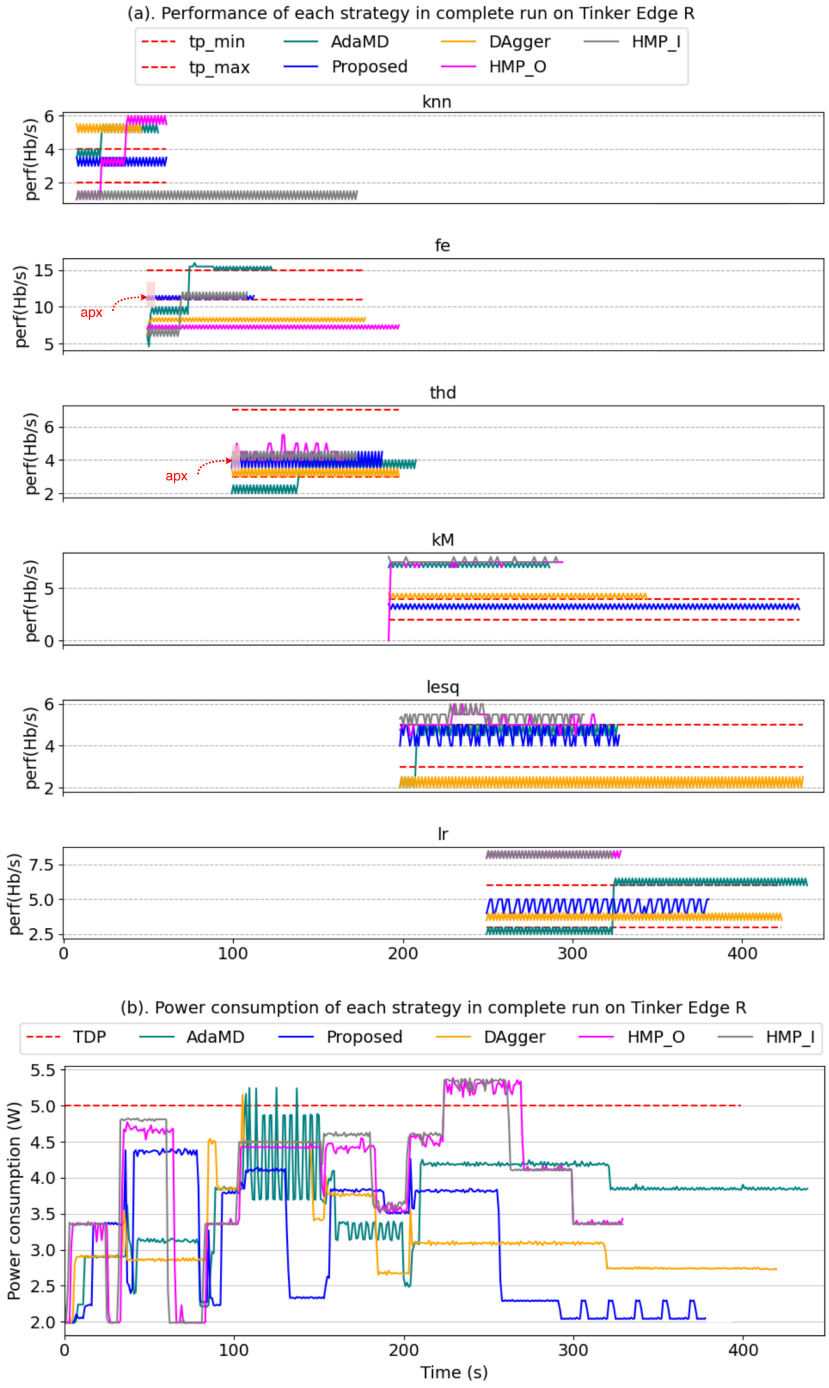

Fig. 10. Per-application performance with different strategies under dynamic workload combination on Odroid XU3.

Table 3. Efficiency of the Proposed Strategy against State-of-the-art for Benchmark Applications

Strategy	Average power (W)	Power Viol. (%)	Performance viol. (%)						Accuracy loss (%)					
			knn	fe	thd	kM	lesq	lr	knn	fe	thd	kM	lesq	lr
AdaMD	3.7	1.14	0	34	37	0	14	49	0	0	0	0	0	0
DAgger	3.2	0.25	0	100	5	0	100	0	0	0	0	0	0	0
HMP_O	3.86	13.94	30	100	0	0	0	0	0	0	0	0	0	0
HMP_I	4.03	11.52	100	25	0	0	0	0	0	0	0	0	0	0
Proposed	4.05	0	0	0	0	0	0	0	0	0	6	7	0	0

with the proposed and *AdaMD* strategies, freeing up the *big* cluster. Both strategies migrate *thd* to the *big* cluster to meet its performance, while the proposed policy also recovers from approximation. At t=190s, *kMeans* arrives; the proposed strategy maps it on four free cores of *LITTLE*, while *DAgger* and *AdaMD* maps it on three cores of *big* cluster. Since both *thd* and *kMeans* are running on *big* cluster with *AdaMD* and *DAgger*, a load unbalance is created due to unequal performance requirements. Thus, high CPU frequency to compensate the performance of power-intensive *thd* overcompensates the performance of *kMeans*. At t=220s, *thd* finishes executing the proposed strategy, while *lesq* arrives into the system. The proposed strategy maps *lesq* on the free cores of the *big* cluster, while *DAgger* and *AdaMD* map *lesq* on free cores of *LITTLE* where the application underperforms. At t=190s, *thd* and *kMeans* finish execution with *AdaMD*, and it migrates *lesq* to the *big* cluster to meet its performance. Finally, *lr* arrives at t=255s, the proposed policy: (i) reduces the DoP of *lesq* to three big cores to make room for the *lesq*, (ii) increases CPU frequency of *big*, (iii) adjusts *CPU quota* of both applications, and (iv) maps *lr* on the free core on the *big* cluster. *AdaMD* maps *lr* on a free core of *LITTLE* and waits for *lesq* to finish execution on the big cluster to migrate *lr* to *big*. Finally, *kMeans* finishes executing with *DAgger*, and it maps *lr* on the free *big* core. The governors map the applications on both clusters and increase the CPU frequencies aggressively to meet the applications' performance requirements. For *HMP_O*, *fe* is mapped on *LITTLE* cluster where its performance suffers throughout the execution. *HMP_I* maps *knn* on the *LITTLE*, also leading to performance violations.

Table 3 shows performance violations and accuracy loss of all the applications with each strategy. The average performance violation with other strategies is around 32.5% for *knn*, 64.75% for *fe*, 10.5% for *thd*, 28.5% for *lesq*, 12.25% for lr, and overall average performance violation of 24.75%. Our strategy meets the performance requirements in all cases with a minimal overall average accuracy loss of 2.2% per application across the entire workload (accuracy loss of 6% and 7% for *fe* and *thd*, respectively). Figure 10(b) shows the run-time power consumption for each strategy, with the *TDP* set to *5W*. The proposed strategies, *AdaMD* and *DAgger,* do not incur any power violations, while both governors result in numerous power violations. *DAgger* has relatively lower power consumption with conservative resource allocation but results in significant performance violations. *AdaMD* proportionally scales DVFS settings for power actuation, inevitably sacrificing performance under intensive workload scenarios. Both *HMP_O* and *HMP_I* have high power consumption, yet resulting in frequent performance violations of both over- and under-compensation. The power consumption of the proposed approach is relatively lower, which is reflected in performance requirements being met in all scenarios at a minimal accuracy loss.

5.2.2 Experiments on Tinker Egde R Board. Similar to the previous evaluation, we performed different experiments by running two, three, and multiple concurrent applications. The three experiments are discussed in the following.

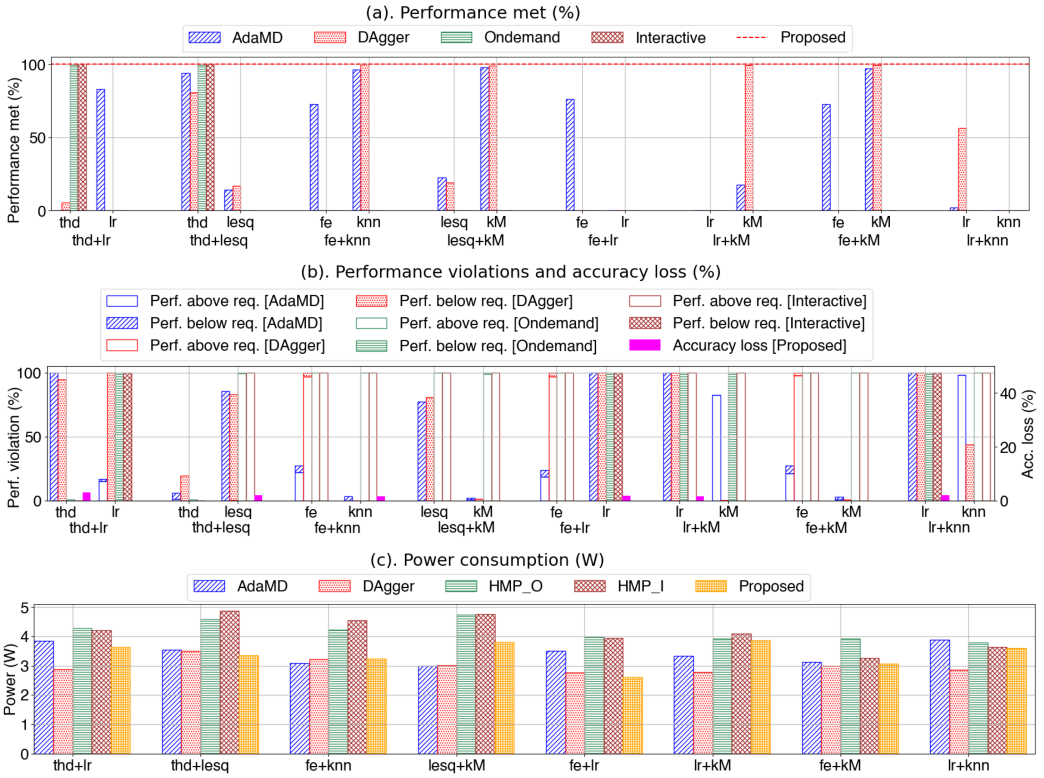

Fig. 11. Comparison of the performance violations, performance met, and accuracy loss for all five strategies when two applications run concurrently in eight different combinations on Tinker board. Note that applications with 0% performance met rate in (a) have 100% violations in (b).

Running 2-applications concurrently. We repeated the same experiments of the eight combinations of two concurrently running applications on the Tinker Edge R board. Figure 11 shows the power and performance of the five strategies including performance met (%) in Figure 11(a), performance violations in Figure 11(b), and power consumption for each combination in Figure 11(c). In most combinations where an application is under-performing, all state-of-the-art strategies map both the applications on *big* cluster causing DVFS high enough to violate the TDP. These strategies scale down the DVFS to avoid the power violations leading to the performance of either application. Since there are only two *big* cores available in the Tinker Edge R board, the state-of-the-art strategies are unable to exploit DoP to gain performance.

The proposed strategy takes advantage of the CPU quota and approximation to meet the performance constraints at a cost of 1/3.5% accuracy loss.

Running 3-applications concurrently. We evaluated the same combinations of the three concurrently running applications on the Tinker Edge R board. The performance met of each combination is shown in Figure 12(a), the performance violations are shown in Figure 12(b), and the power consumption is shown in Figure 12(c). Typically, the state-of-the-art strategies map the first two applications on the *big* cluster to gain performance and map the third application on the *LITTLE* cluster. *AdaMD* and DAgger leverage the DoP of the third application to meet the performance on LITTLE given that the application is parallel. However, *AdaMD* is a run-time strategy and actuates DVFS by continuously monitoring the performance of the applications. However, running three

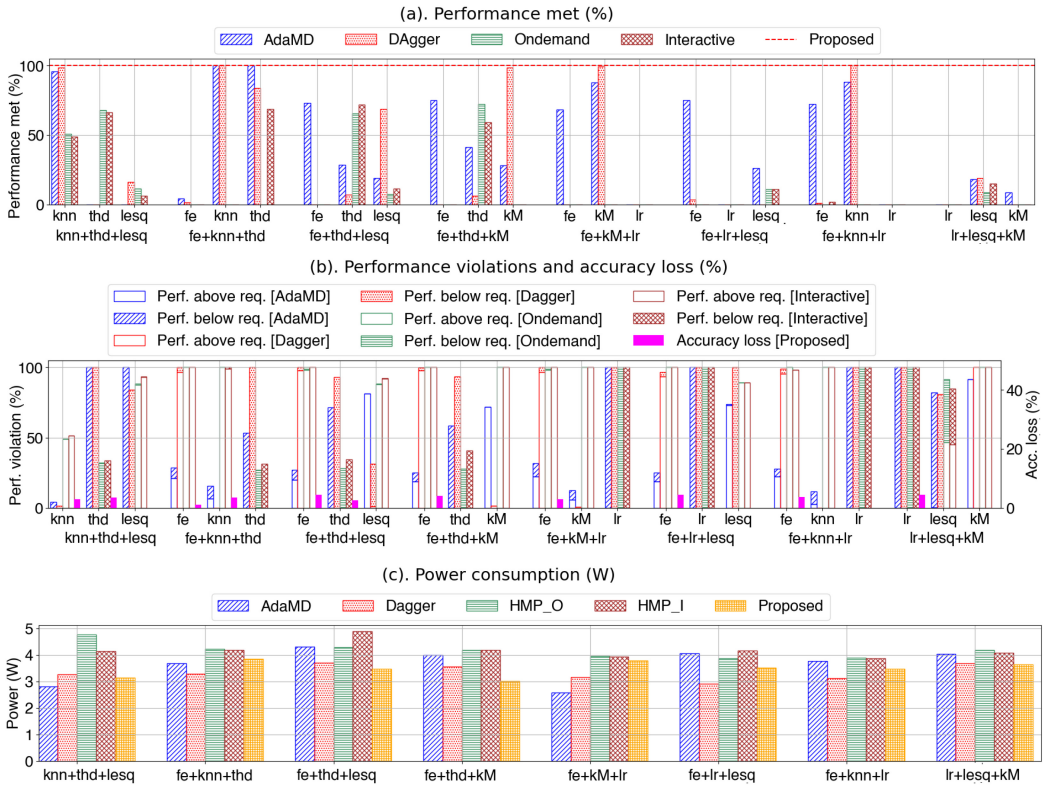

Fig. 12. Comparison of the performance violations, performance met, and accuracy loss for all five strategies when three applications run concurrently in eight different combinations on Tinker board. Note that applications with 0% performance met rate in (a) have 100% violations in (b).

applications on the Tinker board leads to very high power consumption and these state-of-the-art strategies reduce the DVFS to avoid the violations. The governors report the highest power consumption for each combination. However, the proposed strategy (i) balances the load by adjusting the CPU quota of the applications at the *big* clusters, (ii) increases the DoP of the application at the *LITTLE* cluster, and (iii) approximates the application on the *LITTLE* cluster if the application is unable to meet the performance. The overall accuracy loss in the proposed strategy varies between 1-3.75% on average.

Running multiple concurrent applications. We evaluated the concurrently running application experiment on the Tinker board for each strategy. Figure 13(a) shows the performance of each application, Figure 13(b) shows the power consumption of each strategy, and Table 4 shows the summary of power-performance violation and accuracy loss of each strategy. AdaMD maps the first arriving *knn* and *thd* applications on the *big* cluster, and *fe* on 4 *LITTLE* cores. This mapping configuration causes performance lag of *fe*. When *fe* completes execution, *kMeans* arrives and occupies the free *LITTLE* cores. While *lesq* arrives and gets one big core. When *kMeans* completes its execution, *knn* arrives and gets the free LITTLE core leading to under-performance. Eventually, *lesq*, *thd*, and *lr* suffer performance loss due to resource constraints. DAgger maps *knn* on 1B, *fe* on 4L, and *thd* on 1B, where *thd* under-performs. Later, after these applications complete execution, *lesq* gets 1B, *kMeans* gets 3L, and *lr* gets 1L where *lr* under-performs. The proposed strategy maps

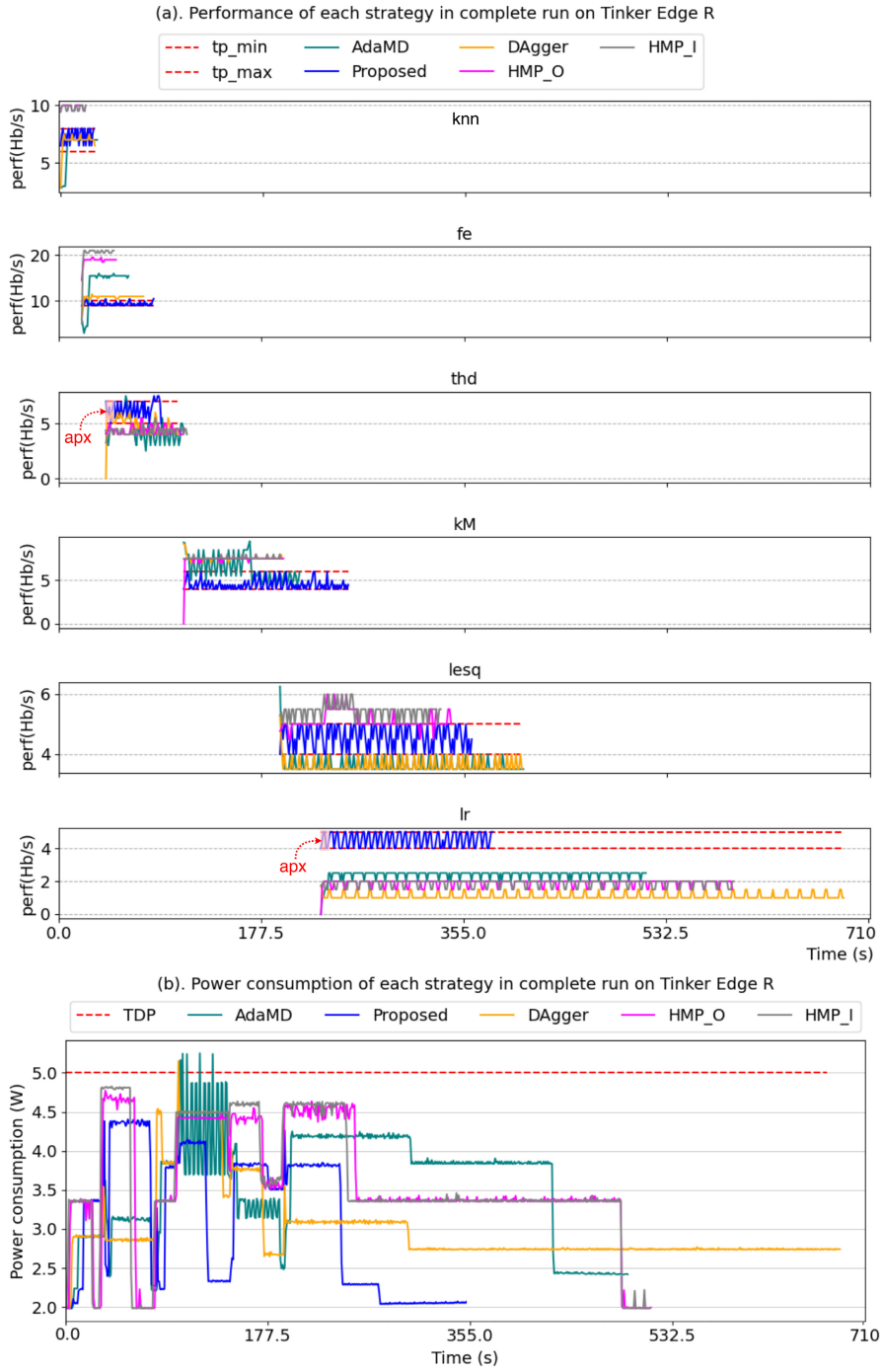

Fig. 13. On Tinker board: (a) Per-application performance with different strategies under dynamic workload combination. (b) Run-time power consumption with different strategies under dynamic workload combination.

Table 4. Efficiency of the Proposed Strategy against State-of-the-art for Benchmark Applications

Strategy	Average power (W)	Power Viol. (%)	Performance viol. (%)						Accuracy loss (%)					
			knn	fe	thd	kM	lesq	lr	knn	fe	thd	kM	lesq	lr
AdaMD	3.63	1.03	18.18	14.29	61.19	0	79.52	100	0	0	0	0	0	0
DAgger	3.0	0.14	6.45	3.63	53.22	0	75.75	100	0	0	0	0	0	0
HMP_O	3.59	0	0	0	83.82	1.23	0	100	0	0	0	0	0	0
HMP_I	3.61	0	0	3.42	92.85	0	0	100	0	0	0	0	0	0
Proposed	3.32	0.0	0	0	0	0	0	0	0	0	4	0	0	3.75

knn, *thd* on the big clusters, *fe* on the *LITTLE* cluster. *thd* is approximated with 4% accuracy loss. *KMeans* gets 4L, *lesq* and *lr* are mapped on the *big* cluster. *lr* is approximated at 3.75% accuracy loss to meet the performance.

6 Conclusions

We presented a run-time resource management strategy for handling dynamic workloads on embedded HMP architectures. Our approach coordinates power/performance decisions by integrating run-time approximation with traditional power knobs. We evaluated our RTM strategy against other relevant strategies using real hardware testbeds of the *Odroid XU3*, and *Tinker Edge R* over dynamic workloads. Our approach ensures performance guarantees and honors the power budget within an average accuracy loss of 2.2% over diverse workload scenarios, outperforming the other state-of-the-art solutions. Future work will focus on extending the proposed approach to consider, (i) advanced heterogeneous multi-core architectures supported hardware accelerators including Graphics Processing Units, GPUs, and Neural Processing Units, NPU, (ii) diverse workload applications supported by DNN and transformer inference, (iii) online design space exploration through reinforcement learning, and (iv) multi-objective policy-based resource scheduling.

References

[1] A. Aalsaud, R. Shafik, A. Rafiev, F. Xia, S. Yang, and A. Yakovlev. 2016. Power–aware performance adaptation of concurrent applications in heterogeneous many-core systems. In *Proc. of Intl. Symp. on Low Power Electronics and Design (ISLPED'16)*. 368–373.

[2] D. Angioletti, F. Bertani, B. Bolchini, F. Cerizzi, and A. Miele. 2019. A runtime resource management policy for OpenCL workloads on heterogeneous multicores. In *Proc. of Design, Automation & Test in Europe Conf. & Exhibition (DATE'19)*. 1385–1390.

[3] ARM. 2013. Global Task Scheduling. https://developer.arm.com/documentation/den0013/d/big-LITTLE/Software-execution-models-in-big-LITTLE/Global-Task-Scheduling. (2013). (Accessed on 02/16/2023).

[4] A. Arranz-Gimon, A. Zorita-Lamadrid, D. Morinigo-Sotelo, and O. Duque-Perez. 2021. A review of total harmonic distortion factors for the measurement of harmonic and interharmonic pollution in modern power systems. *Energies* 14, 20 (2021).

[5] ASUS. 2024. Tinker Edge R. https://tinker-board.asus.com/series/tinker-edge-r.html. (2024).

[6] K. R. Basireddy, A. K. Singh, B. M. Al-Hashimi, and G. V. Merrett. 2020. AdaMD: Adaptive mapping and DVFS for energy-efficient heterogeneous multicores. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2020), 2206–2217.

[7] S. Chakraborty, S. Saha, M. Själander, and K. Mcdonald-Maier. 2021. Prepare: Power aware approximate real-time task scheduling for energy-adaptive QoS maximization. *ACM Trans. on Embedded Computing Systems (TECS)* 20, 5s (2021), 1–25.

[8] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Proc. of IEEE Intl. Symposium on Workload Characterization (IISWC'09)*. 44–54.

[9] S. Conoci, P. Di Sanzo, B. Ciciani, and F. Quaglia. 2018. Adaptive performance optimization under power constraint in multi-thread applications with diverse scalability. In *Proc. of ACM/SPEC Intl. Conf. on Performance Engineering (ICPE'18)*. 16–27.

[10] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray. 2013. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research* 14, 1 (2013), 801–805.
[11] H. J. Damsgaard, A. Grenier, D. Katare, Z. Taufique, S. Shakibhamedan, T. Troccoli, G. Chatzitsompanis, A. Kanduri, A. Ometov, A. Y. Ding, and N. Taherinejad. 2024. Adaptive approximate computing in edge AI and IoT applications: A review. *Journal of Systems Architecture* 150 (2024), 103114. https://doi.org/10.1016/j.sysarc.2024.103114
[12] E. Del Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini. 2016. Workload-aware power optimization strategy for asymmetric multiprocessors. In *Proc. of Conf. on Design, Automation & Test in Europe (DATE'16)*. 531–534.
[13] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt. 2016. SPARTA: Runtime task allocation for energy efficient heterogeneous manycores. In *Proc. of Intl. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS'16)*.
[14] W. El-Harouni, S. Rehman, B. S. Prabakaran, A. Kumar, R. Hafiz, and M. Shafique. 2017. Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE'17)*. 1384–1389.
[15] U. Gupta, M. Babu, R. Ayoub, M. Kishinevsky, F. Paterna, and U. Y. Ogras. 2018. STAFF: Online learning with stabilized adaptive forgetting factor and feature selection algorithm. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.
[16] U. Gupta, M. Babu, R. Ayoub, M. Kishinevsky, F. Paterna, and U. Y. Ogras. 2018. STAFF: Online learning with stabilized adaptive forgetting factor and feature selection algorithm. In *Proc. of Design Automation Conf. (DAC'18)*. Article 177, 6 pages.
[17] U. Gupta, C. A. Patil, G. Bhat, P. Mishra, and U. Y. Ogras. 2017. DyPO: Dynamic Pareto-optimal configuration selection for heterogeneous MpSoCs. *ACM Trans. on Embedded Computing Systems* 16, 5s (2017).
[18] Hardkernel Co. 2015. ODROID. http://www.hardkernel.com (2015). Accessed: 2022-09-01.
[19] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. 2010. Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments. In *Proc. of Intl. Conf. on Autonomic Computing (CAC'10)*. 79–88.
[20] A. Kanduri, M.-H. Haghbayan, A. M. Rahmani, P. Liljeberg, A. Jantsch, H. Tenhunen, and N. Dutt. 2017. Accuracy-aware power management for many-core systems running error-resilient applications. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 25, 10 (2017), 2749–2762.
[21] A. Kanduri, M.-H. Haghbayan, A. M. Rahmani, P. Liljeberg, A. Jantsch, H. Tenhunen, and N. Dutt. 2016. Approximation knob: Power Capping meets energy efficiency. In *Proc. of ACM/IEEE Intl. Conf. on Computer-Aided Design (ICCAD'16)*. 1–8.
[22] A. Kanduri, A. Miele, A. M. Rahmani, P. Liljeberg, C. Bolchini, and N. Dutt. 2018. Approximation-aware coordinated power/performance management for heterogeneous multi-cores. In *Proc. of Design Automation Conf. (DAC'18)*. 68:1–68:6.
[23] A. Karatzas and I. Anagnostopoulos. 2023. OmniBoost: Boosting throughput of heterogeneous embedded devices under multi-DNN workload. In *Proc. of ACM/IEEE Design Automation Conf. (DAC'23)*. 1–6.
[24] H. Khdr, S. Pagani, E. Sousa, V. Lari, A. Pathania, F. Hannig, M. Shafique, J. Teich, and J. Henkel. 2017. Power density-aware resource management for heterogeneous tiled multicores. *IEEE Trans. on Computers* 66, 3 (2017), 488–501.
[25] X. Li, L. Mo, A. Kritikakou, and O. Sentieys. 2023. Approximation-aware task deployment on heterogeneous multicore platforms with DVFS. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 42, 7 (2023), 2108–2121.
[26] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande, and U. Y. Ogras. 2019. Dynamic resource management of heterogeneous mobile platforms via imitation learning. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 27, 12 (2019), 2842–2854.
[27] A. Miele, A. Kanduri, K. Moazzemi, D. Juhasz, A.-M. Rahmani, N. Dutt, P. Liljeberg, and A. Jantsch. 2019. On-chip dynamic resource management. *Foundations and Trends® in Electronic Des. Automation* 13, 1-2 (2019), 1–144.
[28] P. Greenhalgh, ARM. 2011. Big.LITTLE Processing with ARM Cortex™-A15 & Cortex-A7 – White Paper. (2011).
[29] D. Palomino, M. Shafique, A. Susin, and J. Henkel. 2016. Thermal optimization using adaptive approximate computing for video coding. In *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE'16)*. 1207–1212.
[30] M. Rapp, A. Pathania, T. Mitra, and J. Henkel. 2020. Neural network-based performance prediction for task migration on S-NUCA many-cores. *IEEE Trans. Comput.* 70, 10 (2020), 1691–1704.
[31] M. Rapp, M. B. Sikal, H. Khdr, and J. Henkel. 2021. SmartBoost: Lightweight ML-driven boosting for thermally-constrained many-core processors. In *2021 58th ACM/IEEE Design Automation Conference (DAC'21)*. IEEE, 265–270.
[32] M. Rapp, M. Bakr Sikal, H. Khdr, and J. Henkel. 2021. SmartBoost: Lightweight ML-Driven boosting for thermally-constrained many-core processors. In *Proc. of ACM/IEEE Design Automation Conf. (DAC'21)*. 265–270.

[33] E. Shamsa, A. Kanduri, P. Liljeberg, and A. M. Rahmani. 2021. Concurrent application bias scheduling for energy efficiency of heterogeneous multi-core platforms. *IEEE Trans. on Computers* 71, 4 (2021), 743–755.
[34] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. 2011. Managing performance vs. accuracy trade-offs with loop perforation. In *Proc. of ACM Symp. and European Conf. on Foundations of Software Engineering (ESEC/FSE'11)*. 124–134.
[35] A. K. Singh, A. Prakash, K. R. Basireddy, G. V. Merrett, and B. M. Al-Hashimi. 2017. Energy-efficient run-time mapping and thread partitioning of concurrent OpenCL applications on CPU-GPU MPSoCs. *ACM Trans. on Embedded Computing Systems* 16, 5s (2017), 147:1–147:22.
[36] E. Del Sozzo, G. C. Durelli, E. M. G. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini. 2016. Workload-aware power optimization strategy for asymmetric multiprocessors. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE'16)*. IEEE, 531–534.
[37] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali. 2016. Proactive control of approximate programs. In *Proc. of Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16)*. 607–621.
[38] C. Tan, T. S. Muthukaruppan, T. Mitra, and L. Ju. 2015. Approximation-aware scheduling on heterogeneous multi-core architectures. In *Proc. of Asia and South Pacific Design Automation Conf. (ASP-DAC'15)*. 618–623.
[39] Z. Taufique, A. Miele, P. Liljeberg, and A. Kanduri. 2024. Adaptive workload distribution for accuracy-aware DNN inference on collaborative edge platforms. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC'24)*. IEEE, 109–114. https://doi.org/10.1109/ASP-DAC58780.2024.10473987
[40] Z. Taufique, A. Vyas, A. Miele, P. Liljeberg, and A. Kanduri. 2024. Tango: Low latency Multi-DNN inference on heterogeneous edge platforms. In *2024 IEEE 42nd International Conference on Computer Design (ICCD'24)*. 300–307. https://doi.org/10.1109/ICCD63220.2024.00053
[41] Z. Taufique, A. Vyas, A. Miele, P. Liljeberg, and A. Kanduri. 2024. HiDP: Hierarchical DNN partitioning for distributed inference on heterogeneous edge platforms. *Proc. Design, Automation and Test in Europe Conf. (DATE)* (2025), 1–6. arXiv preprint arXiv:2411.16086.
[42] Z. Taufique, A. Kanduri, M. A. Bin Altaf, and P. Liljeberg. 2021. Approximate feature extraction for low power epileptic seizure prediction in wearable devices. In *Proc. of IEEE Nordic Circuits and Systems Conf. (NorCAS'21)*. 1–7.
[43] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra. 2019. High-throughput CNN inference on embedded ARM Big.LITTLE multicore processors. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2019), 2254–2267.
[44] Y. Wu, Y. Gong, Z. Zhan, G. Yuan, Y. Li, Q. Wang, C. Wu, and Y. Wang. 2023. MOC: Multi-objective mobile CPU-GPU co-optimization for power-efficient DNN inference. In *Proc. of ACM/IEEE Intl. Conf. on Computer Aided Design (ICCAD'23)*. 1–10.

Received 10 October 2024; revised 5 February 2025; accepted 9 March 2025