Approx-RM: Reducing Energy on Heterogeneous Multicore processors under Accuracy and Timing Constraints

Approx-RM: Reducing Energy on Heterogeneous Multicore Processors under Accuracy and Timing Constraints

MUHAMMAD WAQAR AZHAR, MADHAVAN MANIVANNAN, and PER STENSTRÖM,
Chalmers University of Technology, Sweden

Reducing energy consumption while providing performance and quality guarantees is crucial for computing systems ranging from battery-powered embedded systems to data centers. This article considers approximate iterative applications executing on heterogeneous multi-core platforms under user-specified performance and quality targets. We note that allowing a slight yet bounded relaxation in solution quality can considerably reduce the required iteration count and thereby can save significant amounts of energy. To this end, this article proposes *Approx-RM*, a resource management scheme that reduces energy expenditure while guaranteeing a specified performance *as well as* accuracy target. *Approx-RM* predicts the number of iterations required to meet the relaxed accuracy target at runtime. The time saved generates execution-time slack, which allows *Approx-RM* to allocate fewer resources on a heterogeneous multi-core platform in terms of DVFS, core type, and core count to save energy while meeting the performance target. *Approx-RM* contributes with lightweight methods for predicting the iteration count needed to meet the accuracy target and the resources needed to meet the performance target. *Approx-RM* uses the aforementioned predictions to allocate *just enough* resources to comply with quality of service constraints to save energy. Our evaluation shows energy savings of 31.6%, on average, compared to *Race-to-idle* when the accuracy is only relaxed by 1%. *Approx-RM* incurs timing and energy overheads of less than 0.1%.

CCS Concepts: • **Hardware** → **Power and energy**; **Power and energy**; • **Computer systems organization** → *Architectures*; *Embedded systems*; *Real-time systems*; *Multicore architectures*; • **Computing methodologies** → **Machine learning**; *Artificial intelligence*;

Additional Key Words and Phrases: Energy efficiency, approximate iterative applications, resource management, quality of service, heterogeneous multicore processors, DVFS

ACM Reference format:
Muhammad Waqar Azhar, Madhavan Manivannan, and Per Stenström. 2023. Approx-RM: Reducing Energy on Heterogeneous Multicore Processors under Accuracy and Timing Constraints. *ACM Trans. Arch. Code Optim.* 20, 3, Article 44 (July 2023), 25 pages.
https://doi.org/10.1145/3605214

This research has been supported by the Swedish Research Council under contract 2019-04929, and by the Swedish Foundation for Strategic Research under contract CHI19-0048. Moreover, the European Union has also partially funded this research under the PRIDE project (grant agreement No EU-101051997).
Authors' address: M. W. Azhar, M. Manivannan, and P. Stenström, Department of Computer Science and Engineering, Chalmers University of Technology, 41296 Göteborg, Sweden; emails: {waqarm, madhavan, per.stenstrom}@chalmers.se.

1544-3566/2023/07-ART44 $15.00
https://doi.org/10.1145/3605214

1 INTRODUCTION

Reducing energy consumption is vital in the entire computing ecosystem, from battery-powered embedded systems to **high-performance computer (HPC)** systems. In the **Race-to-idle (RTI)** strategy, the application is run at full speed, i.e., by using the highest frequency until completion, and then idles until the deadline. This over-provisions resources and can consume more energy. Prior works [4–6, 16, 22, 26, 27, 32] have demonstrated the shortcomings in RTI and proposed techniques to reduce energy consumption employing appropriate resource allocation. However, these proposals mainly focus on improving energy/power efficiency under QoS constraints without trading off the accuracy of the computation. Specifying performance requirements in terms of **quality of service (QoS)** allows a runtime resource manager to allocate a minimum of resources, e.g., core type, core count, and **voltage frequency (V-F)**, to reduce energy consumption while also ensuring that QoS targets are satisfied.

Approximate iterative applications (AIAs), e.g., iterative solvers and gradient descent, represent an important class of applications where the solution error reduces and the *solution quality* improves with the execution of each new iteration. Furthermore, such applications terminate when the solution error reaches a specific target, thus guaranteeing that the solution error is bounded at the end of the execution.

Some recent proposals have managed to reduce energy in AIAs by exploiting the tradeoff between computation accuracy and energy reduction and can be broadly grouped into two categories. The first category comprises methods that enable energy reduction while providing accuracy guarantees, but without meeting performance constraints [35, 36] or through trading off accuracy for improving performance [15, 33]. The second category offers methods that meet performance constraints without providing accuracy guarantees [9, 14]. While Kulkarni et al. [21] offer both accuracy and performance guarantees, their technique makes **resource allocation (RA)** decisions based on offline characterization of the workload on all possible hardware configurations. In summary, existing proposals have the following shortcomings. First, they do not target energy reduction under *both* accuracy and performance constraints in an application-agnostic manner. Second, they base resource allocation decisions on *offline* characterization of application behavior, thus being unable to exploit the opportunities on offer due to variance in runtime behavior. Third, these proposals only employ a subset of the configuration space, i.e., V-F states, core types, and core counts, available in commodity multi-core architectures, for energy reduction, and thus can only harness limited energy savings.

We propose *Approx-RM*, an application-agnostic, online framework to reduce energy for approximate iterative applications on heterogeneous multi-core platforms while meeting both accuracy and performance targets. Typically, there is a diminishing return on improving the solution quality as the execution proceeds. *Approx-RM* leverages this by trading a slight but bounded relaxation in the target solution error for a significant reduction in the number of iterations executed. This enables the runtime system to trade a slight accuracy loss for a significant energy reduction.

Approx-RM first predicts the required number of iterations of the application kernel to meet the quality target. It uses this information and runtime application behavioral prediction (i.e., **cycles per instruction (CPI)**, **misses per instruction (MPI)**, energy, etc.) to predict the resource allocation needed to meet the user-specified deadline (or QoS target). In doing so, all of the above-mentioned steps represent a challenge that must be addressed for the whole scheme to work. First, Approx-RM requires an online prediction method to estimate the iteration count needed to meet the accuracy target. Second, an online performance and energy prediction method is needed to estimate the resources needed for a specific configuration. Third, we need a search

heuristic that employs both predictions to efficiently search the configuration space and allocate enough resources to meet the performance and accuracy target. A necessary requirement for all these mechanisms is to have low overheads in order to have sufficient energy savings overall.

Approx-RM addresses the first challenge by using curve fitting to predict the iteration count needed to meet the accuracy target. To address the second challenge, it uses models that leverage on hardware performance counters and onboard energy sensors to predict the execution time and energy consumption of the entire configuration space, including the voltage-frequency range, core types, and core count. These measures are taken, iteration by iteration, to yield significant energy savings while meeting the accuracy and performance constraints. Third, *Approx-RM* presents a resource allocation policy that searches the entire configuration space, in linear time, and uses the estimates of iteration count and application runtime behavior to allocate appropriate resources to harness the full potential of energy savings.

Approx-RM is evaluated on an ARM big.LITTLE platform (ODROID XU-3 board with Exonys 5422 [8]) comprising four big and four LITTLE cores organized in two clusters. Results show energy savings of 31.6%, on average, compared to *Race-to-Idle* (e.g., [2, 18, 20]), while accuracy is reduced by only 1%. The contributions of this article are as follows.

- A runtime resource manager, *Approx-RM*, for approximate iterative applications to save energy under accuracy and performance constraints by adapting core types, core count, and DVFS
- A lightweight prediction method, based on curve fitting, to predict the number of iterations to meet the quality target
- A lightweight method to predict the resources needed to meet the performance target using hardware performance counters
- An evaluation of the energy-saving framework that shows average energy savings up to 34% compared to race-to-idle, while keeping overheads low ($< 0.1\%$)

The rest of the article is organized as follows. In Section 2, we provide background and motivational data. Section 3 presents *Approx-RM*. The experimental methodology and implementation-related details are presented in Section 4. Section 5 evaluates *Approx-RM*. Related work is discussed in Section 6 before the article is concluded in Section 7.

2 BACKGROUND AND MOTIVATION

2.1 Heterogeneous Multi-core Platform

We consider **heterogeneous multi-core platforms (HMPs)**, e.g., ARM *big.LITTLE* [23], that comprise different core types with the same ISA having the capability to control the **voltage-frequency (V-F)** level of a set of cores of the same type within a cluster. To strike a tradeoff between performance and energy consumption, a runtime resource manager chooses among a set of *hardware configurations*, each configuration comprising a combination of core type, V-F setting, and core count. Additional details about the experimental platform are provided in Section 4.1.

2.2 Accuracy-energy Tradeoff in Approximate Iterative Applications

This article targets AIAs. The solution error decreases with each iteration of the *computational kernel*, and the application is terminated when the solution error reaches a predefined target. Figure 1 shows a block diagram of an AIA. To further elaborate on this temporal variation in solution error, we show the solution error curve of a few AIA applications in Figure 2. The vertical axis represents the solution error on a logarithmic scale, and the horizontal axis represents the number of iterations.

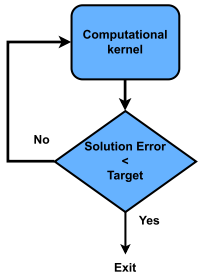


Fig. 1. Structure of an approximate iterative application.

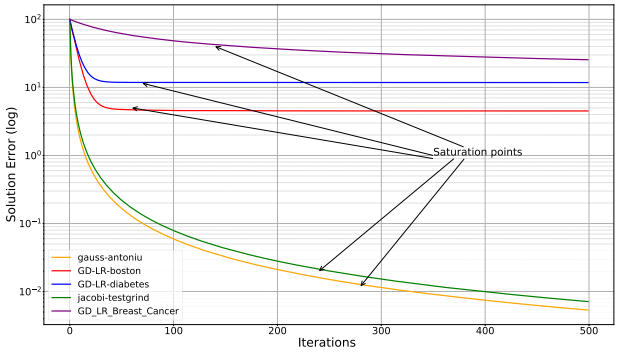


Fig. 2. Behavior of solution error with respect to iteration count for selected workloads.

There are two important observations here. First, the solution error decreases as the application progresses, and various workloads have distinct curves. Second, the solution error's rate of decrease and the saturation point, where the solution error bottoms out, is application and input dependent. For example, *gauss-antoniu*, an iterative solver workload, and *GD-LR-boston* and *GD-LR-diabetes,* gradient descent for **multi-variate linear regression (MVLR)** workloads, show different behavior even though both workloads from the MVLR class have identical hyperparameters, e.g., the learning rate. The reason is that the change and saturation point rate depend on the input data distribution. We will build on the important observation that the improvement in solution quality shows diminishing returns as the application progresses, as highlighted by arrows pointing to the saturation points. It suggests that one can trade a considerable number of iterations for a small relaxation of the accuracy, which can be translated into substantial energy savings.

Note that the solution error is a property of the algorithm and is independent of the resource allocation. Controlled relaxation of the solution quality target by a small margin, e.g., 0.01%, allows the resource manager to trade a slight amount of accuracy for significant energy savings. A reduction in solution quality allows for a decrease in the number of executed iterations. The decrease in the number of iterations allows each iteration to execute more slowly, yielding an opportunity to reduce resource needs to meet the deadline, hence saving energy.

The reduction in the iteration count when the solution error target is increased by a small margin is shown in Figure 3. Here, the vertical axis is the percentage reduction in iteration count required to reach the new solution error target, while the horizontal axis shows the percentage addition in the solution error target. The reduction in iteration count varies widely across applications since different workloads have distinct solution-error curves. For example, *gauss* shows a slight and almost proportional reduction in iteration count. On the other hand, the behavior of gradient descent for multi-variate linear regression shows that the solution error saturates, yielding a considerable decrease in iteration count. Moreover, since the saturation point depends on the input data, different algorithms show distinct behavior. Hence, reducing the required iteration count allows the resource manager to execute each iteration more slowly, thereby allocating fewer resources to save energy.

2.3 Approx-RM Usage

We envision *Approx-RM* as a runtime resource manager that must provide resource allocation as shown in line 2 (highlighted in color) of Algorithm 1. Users need to specify the performance and accuracy targets for the scheme.

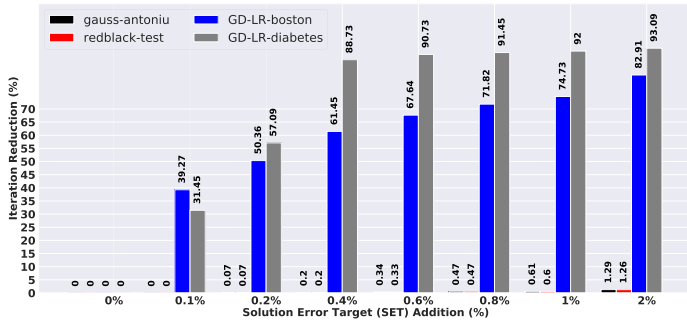


Fig. 3. Relation between reduction in the iteration count and the solution error.

ALGORITHM 1: Compiler directive specification of Approx-RM

```
1: Initialization()
2: #pragma Approx_RM(Deadline,Accuracy)
3: while Error > Solution_Error_Target do
4:     Error ← Applicatin_Kernel()
5: end while
```

A source-to-source compiler can transform this specification to insert inline functional calls as shown in Algorithm 2 and highlighted in color. These calls include functions to read the hardware performance counters before (line 5) and after (line 7) each invocation of the kernel. The performance counter measurements and solution error results are fed into *Approx-RM* to predict and apply resource allocation (line 8).

ALGORITHM 2: Compiler transformation to insert inline routines to read PMC and resource prediction using Approx-RM

```
1: Initialization()
2: Resource_Allocation ← Baseline_Resource_Allocation
3: Approx_RM_Initilization(Deadline,Error_Target)
4: while Error > Solution_Error_Target do
5:     Start = Approx_RM_ROI_Start(Resorce_Allocation)
6:     Error ← Applicatin_Kernel()
7:     End = Approx_RM_ROI_End()
8:     Resorce_Allocation = Approx-RM_Resource_Allocator(Start,End,Error)
9: end while
```

3 APPROX-RM

3.1 Overview

Figure 4 shows how Approx-RM interacts with the hardware platform and the application. Approx-RM assumes that the application outputs a solution error after each iteration and performance counter readings are available at runtime. Approx-RM uses this information first to predict the application duration, i.e., the number of iterations needed to meet the accuracy targets set by the user. Then it establishes the time allocation per iteration to meet the program deadline. Application duration, execution time, and energy predictions are used to select a configuration, i.e., core type, core count, and voltage-frequency pair, that minimizes energy.

After the completion of each iteration, the solution error and execution parameters are fed into Approx-RM for future predictions. Execution parameters comprise **instruction count (I)**, CPI,

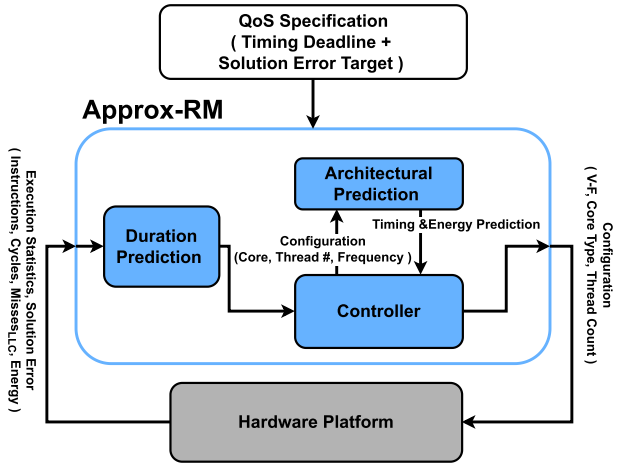


Fig. 4. Approx-RM and system framework.

misses per kilo instruction (MPKI), and **energy consumption (E)**. Approx-RM uses the solution error target to predict the number of iterations needed. Furthermore, the history of execution parameters is used to predict the execution time and the energy consumption. This process continues until the completion of the application.

Approx-RM predicts three quantities: (1) the application duration using the history of the solution error; (2) the computational demand, i.e., the instruction count, for future iterations based on instruction counts of previous iterations; and (3) timing and energy for specific hardware configuration using execution parameters for that configuration. Finally, Approx-RM makes timing and energy predictions for a set of configurations after each iteration to decide on the best hardware configuration to use in the next iteration.

3.2 Resource Management Algorithm

Approx-RM is invoked after each iteration to determine a hardware configuration for the next iteration, and this procedure continues until the QoS target regarding the quality is met. The pseudo-code depicted in Algorithm 3 establishes the set of hardware configurations that meet timing requirements and have minimum energy for each combination of core type and core count and then picks the one with the minimum energy. This configuration is then used to execute the next iteration. The first step is to predict the duration of the application (line 9). This step predicts how many iterations are needed to reach the target solution error. It sets the Pred_Accurate flag to indicate that the prediction of the duration is within a confidence interval (details are provided in Section 3.3). If the prediction is accurate enough, i.e., Pred_Accurate is *true* (line 10), one of the most energy-efficient configurations (lines 11–24) is identified; otherwise, *Approx-RM* chooses a baseline configuration.

First, the time needed to meet the performance target for the remaining iterations is computed (line 11). Then, Approx-RM evaluates all combinations of core types and core count in the configuration space (lines 13 and 14) and predicts the minimum voltage and frequency for each combination of core type and core count (line 15) using the Predict_Voltage_Frequency() function. If the predicted frequency is outside of the allowed frequency range (line 16), it is converted into the allowed range, i.e., Find_Legal_Voltage_Frequency(). Since the resulting frequency can violate the timing requirement, the execution time and energy consumption are predicted using the Predict_Time() (line 18) and Predict_Energy() (line 17) functions, respectively.

ALGORITHM 3: Resource management algorithm

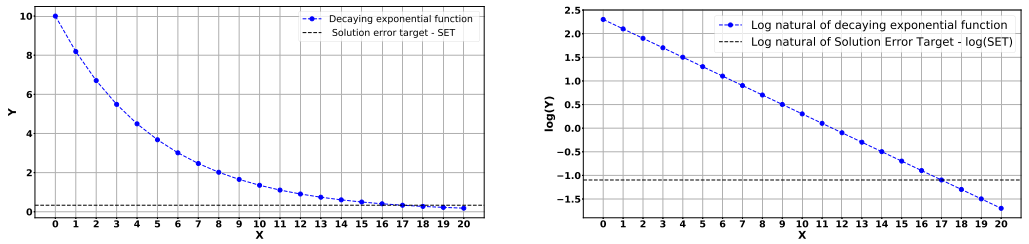
```
1: Notations:
2: D_Itr: Deadline per iteration
3: D: Program deadline relative to start time
4: Time : Current time
5: Core_Types : Core types in the heterogeneous multi-core platform
6:
7: function Approx-RM_Resource_Allocator(Start,End,Error)
8:     Insert_PMC_Data(Start,End)
9:     Insert_Error_Data(Error)
10:    Time ← get_System_Time()
11:    Predicted_Configuration ← Approx-RM(Error,Time)
12:    return Predicted_Configuration
13: end function
14:
15: function Approx-RM(Error,Time)
16:    Predicted_Configuration = Configuration_Baseline
17:    [Iteration_Pred, Pred_Accurate] ← Predict_Duration(Error)
18:    if Pred_Accurate = TRUE then
19:        D_Itr = (D−Time) / (Itr_Pred − Itr_Executed)
20:        Energy_Min ← ∞
21:        for all Core ∈ Core_Types do
22:            for all Core_Count ∈ Core_Combinations do
23:                Voltage_temp, Freq_temp ← Predict_Voltage_Frequency(Core, Core_Count, D_Itr)
24:                Voltage_Legal, Freq_Legal ← Get_Legal_Voltage_Frequency(Freq_temp, Proc)
25:                Energy_Pred ← Predict_Energy(Core, Core_Count, Voltage_Legal, Freq_Legal)
26:                Time_Pred ← Predict_Time(Core, Core_Count, Freq_Legal)
27:                if ( Time_Pred ≤ D_Itr & Energy_Pred < Energy_Min) then
28:                    Predicted_Configuration = [Core, Core_Count, Voltage_Legal, Freq_Legal]
29:                    Energy_Min ← Energy_Pred
30:                end if
31:            end for
32:        end for
33:    end if
34:    return Predicted_Configuration
35: end function
```

Next, the execution time is compared with the deadline per iteration, establishing whether the energy is minimum (line 19). If so, this configuration is marked as being the one that consumes the least energy.

3.2.1 Time Complexity of Approx-RM. The Approx-RM algorithm only exhaustively traverses the core count combinations of a specific core type. For instance, in the eight-core platform (comprising four A15 and four A7 cores) used in the evaluation, the total number of evaluated configurations is eight, i.e., 4-big, 3-big, 2-big, 1-big, 4-LITTLE, 3-LITTLE, 2-LITTLE, and 1-LITTLE. Thus, its time complexity is linear, i.e., O(n), where n represents the total number of cores in a system that is a product of core types, i.e., $Core_{Types}$, and the number of cores of a specific type, i.e., $Core\_Count_{Combinations}$, assuming a symmetric system.

3.3 Application Duration Prediction

Application duration is predicted by storing the old samples of the solution error and then employing model fitting to predict the future behavior of the curve and, consequently, the application duration. The duration prediction is based on two components: (1) curve fitting and (2) using the curve-fitting model to predict the duration.



(a) Decaying exponential curve (b) Natural log of decaying exponential curve

Fig. 5. A generic decaying exponential curve and its linear transformation using a logarithmic function.

The first step in the duration prediction is to assume an appropriate curve fitting to the input data, i.e., solution error. Here, the decaying exponential curve that best matches the behavior of the solution error is assumed, and a pictorial representation is shown in Figure 5(a). Equation (1) provides the mathematical model, where a and b represent the intercept and slope, respectively. The model output, i.e., y, represents the solution error, and x represents the iteration number. Thus, replacing y with $\text{Error}_{\text{Target}}$ and re-arranging the equation leads to Equation (2), which predicts the iteration count based on the solution target, i.e., $\text{Error}_{\text{Target}}$ set by QoS. The *Error history* buffer only stores the last K samples of the solution error. Thus, Equation (2) predicts the duration from the current instant until the application finishes. Consequently, we add the completed iteration count, i.e., $\text{Iterations}_{\text{complete}}$, to finally derive Equation (3), which predicts the application duration. We add a margin, say 10%, to the number of predicted iterations to safeguard against under-prediction.

A solution error history buffer of size K is implemented using **First-In-First-Out (FIFO)**. The reason is that storing all samples of the solution error can lead to considerable overheads. The duration prediction is performed when the buffer contains all the new K samples of error history after K iterations. This helps to keep the overheads sufficiently low. In summary, Equation (3) predicts the application duration. However, a fundamental problem remains: the determination of the intercept and the slope of the decaying exponential model. This is discussed next.

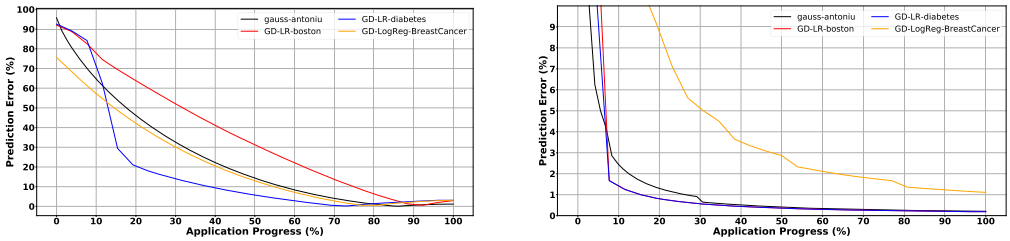
$$y = a * e^{b*x} \quad (1) \qquad x = \frac{\ln(\text{Error}_{\text{Target}}) - \ln(a)}{b} \quad (2)$$

$$\text{Duration} = \text{Iterations}_{\text{complete}} + \frac{\ln(\text{Error}_{\text{Target}}) - \ln(a)}{b} \quad (3)$$

3.3.1 Model Fitting. The purpose of employing curve fitting in Approx-RM is to find the intercept a and the slope b of the curve that plots solution error over time. To use linear regression, the curve must be linearized. This can be done by applying the natural logarithm as shown in Equations (4) to (6). Rearrangement of the equation yields Equation (6), which is in a linear form. ($y = mx + c$) suitable for applying linear regression. Here, $\overline{y}$ and $\overline{a}$ represent the natural logarithm of y and a, respectively. Figure 5(a) shows the decaying exponential curve, and Figure 5(b) shows the logarithm of the same data. As can be seen, the logarithmic transformation converts the data representing the decaying exponential behavior to linear form, thus enabling the use of methods such as the least-square method to find line parameters. The intercept and slope can be inserted into Equation (3) to predict the application duration.

$$\ln(y) = \ln(a * e^{b} * x) \quad (4) \qquad \ln(y) = \ln(a) + b * x \quad (5) \qquad \overline{y} = \overline{a} + b * x \quad (6)$$

To find line parameters, we employ a low-overhead averaging method. The slope is determined by using the first and last sample in the solution error history (see Equation (7)), and the intercept



(a) Prediction error of duration of future iterations

(b) Prediction error of duration of executed iterations

Fig. 6. Prediction error for future and executed iterations using the decaying exponential model.

is equal to the first sample in the error history buffer (see Equation (8)).

$$\text{slope} = \text{a} = \frac{\text{Error}[0] - \text{Error}[k]}{k} \quad (7) \qquad \text{Intercept} = \text{b} = \text{Error}[0] \quad (8)$$

3.3.2 Confidence Interval (Model Fitting). The prediction accuracy of application duration improves as the execution progresses. If resource allocation is applied at intervals where duration prediction is under-predicted, it can cause a violation of the deadline. Thus, a mechanism is needed that can determine the accuracy of the prediction (i.e., confidence interval) as depicted by the $\text{Pred}_{\text{Accurate}}$ flag on lines 9 and 10 in Algorithm 3.

To elaborate on the problem, we have plotted the *prediction error* (y-axis) as a function of application progress (x-axis) in Figure 6(a). As explained earlier, the prediction error decreases as the workload execution proceeds. Furthermore, the duration is under-predicted at the start due to under-fitting and over-predicted at later stages in execution due to over-fitting. This is due to the fact that we employ a very simple model to predict the duration so that the overheads can be controlled. In this context, it is crucial to apply the resource allocation optimizations at an instant where the prediction accuracy is high. Otherwise, it could result in an allocation of fewer resources than required and could lead to missing the timing deadline.

In this regard, we propose a solution that uses the slope and intercept computed by the duration prediction mechanism to predict the count of already executed iterations. Since we know the number of completed iterations, the prediction error can be computed and referred to as the *error-history prediction*. This can be used to establish the efficacy of the curve-fitting parameters and hence the accuracy of predicted duration. To elaborate on the proposed solution, we have plotted the prediction error (y-axis) of executed iterations as a function of the execution progress (x-axis) in Figure 6(b) for several workloads. As can be seen, the error history prediction decreases as the execution proceeds. Moreover, a common threshold for clusters of workloads can be empirically found. Thus, we divide applications into three clusters and perform offline analysis, where the error-history prediction for the instant where prediction error for future iterations reaches below 30% is recorded. The measured value of the error-history prediction for all workloads in a cluster is averaged to find a common value for the cluster.

Since the model fitting method predicts the iteration count from the start of execution until the current iteration, the solution error for the first iteration is stored, and the solution error for the last sample in the error history buffer is used as the error target.

Please note that the graphs shown above are only for demonstration purposes. In reality, the runtime system has no prior knowledge. The prediction error of all executed iterations is computed at runtime at each instant whenever the future duration is predicted. Once it reaches a target, the $\text{Pred}_{\text{Accurate}}$ flag is set. Before this flag is set, the resource manager uses the baseline configuration,

and once the flag is set, the resource manager activates resource allocation. The earlier resource allocation is activated, the more energy can be saved, although it can result in a greater violation of the timing deadline and vice versa.

3.3.3 Duration Predictor Applicability. The duration prediction mechanism employed in this work assumes a monotonic behavior of solution error. Consequently, the duration prediction mechanism employed is also very simple and will only be able to predict error curves with such behavior accurately. However, our resource allocation scheme is modular and can incorporate any other sophisticated prediction mechanism to predict the non-monotonic behavior of solution error and save energy in those cases as well. However, it must be noted that advanced prediction techniques would incur higher overheads.

3.4 Execution Time and Energy Prediction

3.4.1 Workload Prediction. The timing and energy consumption of any application depend on workload size. As in the case of iterative applications, the same kernel is executed in each iteration. In this context, we assume instruction count (I) as a proxy for workload size. However, it has been shown in the prior art that input data can cause variation [17] in instruction count for successive executions of a kernel because of various control flows within the code. Nevertheless, these variations are often subtle from one iteration to the other [6], contributing to significant variations over a multitude of iterations. Therefore, monitoring the instruction count at the granularity of iterations can capture the trend. This can be used to predict the instruction count for future kernel executions and has shown to be a reliable method of predicting execution time [4–6] in the prior art. Such a method could suffer from under- or over-prediction in case of very high variation. In this context, there are two critical insights. First, since the RM repeatedly checks against the mispredictions by comparing the remaining time and deadline, the effect will be automatically curtailed. Second, it has been shown in the prior art that variations from iteration to iteration, even in the case of different control flows within the program, are generally small [6] and predictable. Thus, we record the instruction count for the "h" loop iteration in a FIFO buffer and employ averaging to predict the instruction count in future loop iterations.

3.4.2 Timing Prediction Model. The timing and energy prediction for a given workload, i.e., instruction count, depend on the chosen configuration. This article employs a simple execution-time model depicted in Equation (9), where T_{Itr}, I, F, CPI_0, MPI_{LLC}, and MP_{LLC} represent the execution time per iteration, instruction count, frequency, cycle per instruction base, misses per instruction for the **last-level cache (LLC),** and miss penalty for the LLC, respectively. Equation (9) represents the execution time as a combination of compute and memory components and is generically applicable to any computing system. We chose this simple model as it gives enough accuracy while keeping the runtime overheads of Approx-RM in check. We argue that since the configuration space is quantized, the added accuracy might not always translate into additional energy savings but certainly add to the overheads. In the context of Approx-RM, Equation (9) is used to predict the execution time per iteration.

$$T_{Itr} = D_{Itr} = \frac{I \times CPI_0}{F} + I \times MPI_{LLC} \times MP_{LLC} \quad (9) \qquad F = \frac{CPI_0 \times I}{D_{Itr} - I \times MPI_{LLC} \times MP_{LLC}} \quad (10)$$

Furthermore, to prune the configuration space, we need a model to predict the minimum frequency at which an iteration can execute while meeting the soft deadline per iteration. Thus, we re-arrange it to derive a frequency prediction model. Since an iteration must finish execution before its allocated time, the execution time T_{itr} can be equated with a deadline per iteration D_{Itr}. Furthermore, solving the equation for frequency F leads to Equation (10). The Predict_Frequency

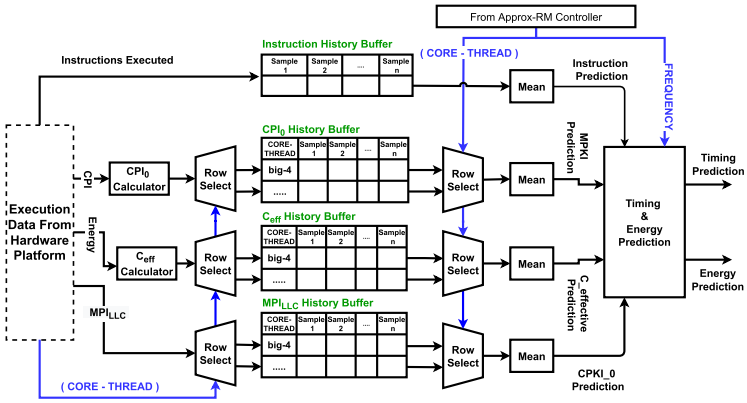


Fig. 7. Overview of the prediction mechanism for CPI_0, C_{eff}, MPI_{LLC}, and I and consequently timing and energy for future iterations.

function implements Equation (10), where the input values of I, CPI_0, and MPI_{LLC} are predicted using the equation/model detailed in Section 3.4.4.

3.4.3 Energy Prediction Model. The energy prediction used in function PREDICT_ENERGY() is given by Equation (11), where α, C, V, F, and T_{pred} represent the activity factor, the capacitance, the voltage, the frequency, and the predicted execution time, respectively.

$$E = \alpha \times C \times V^2 \times F \times T_{pred} \quad (11) \qquad E = C_{eff} \times V^2 \times F \times T_{pred} \quad (12)$$

This equation can be further simplified by replacing the product of the activity factor α and the capacitance C with the *effective capacitance* C_{eff} [19], resulting in Equation (12). The capacitance is a property of the circuit, and the activity factor primarily depends on the interaction of the application program with the micro-architecture. We use the *effective capacitance* values measured from completed iterations to predict the value for future iterations in the same application. Thus, C_{eff} is predicted for every application at runtime, where details are given in Section 3.4.4. In this context, it is important to note that energy consumption depends on the predicted execution time. Execution time, in turn, depends on the chosen V-F pair calculated in the last step.

3.4.4 Timing and Energy Predictor Architecture. The timing and energy prediction model relies on the prediction of I, MPI_{LLC}, C_{eff}, and CPI_0. Therefore, the predictor design is based on storing the history of these parameters that are averaged to predict future values. A block diagram of the predictor is shown in Figure 7.

The predictor performs two essential tasks. First, it inserts the old sample into the history buffers, and second, it predicts the execution behavior of iterations. As for the first task, the measured values of I, CPI_0, MPI_{LLC}, and C_{eff} are stored in the history tables using the FIFO principle after the completion of each iteration. The values of CPI_0, MPI_{LLC}, and C_{eff} can be different for each combination of core type and core count. Thus, the history of these parameters is stored in separate rows in the history tables for every combination of core type and core count. As all core types have the same ISA, the instruction count is the same. CPI_0 and C_{eff} are derivative values, so they are computed on the fly using Equations (13) and (14), respectively, where $MP_{LLC(cycles)}$ and T_{exe} represent the miss penalty for LLC misses in cycles and the execution times of the iterations, respectively.

$$CPI_0 = CPI - MPI_{LLC} \times MP_{LLC(cycles)} \quad (13) \qquad C_{eff} = \frac{E}{V^2 \times F \times T_{exe}} \quad (14)$$

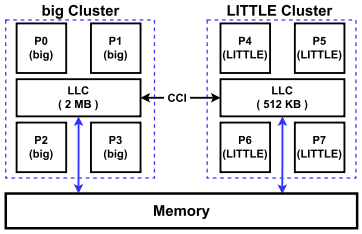


Fig. 8. Hardware platform.

Frequency(M-Hz)	2000	1900	1800	1700	1600	1500	1400	1300
big Voltage (mV)	1300	1225	1175	1137.5	1100	1062.5	1037.5	1025
LITTLE Voltage (mV)	-	-	-	-	-	-	1250	1200
Frequency(M-Hz)	1200	1100	1000	900	800	700	600	500
big Voltage (mV)	1000	975	950	925	900	-	-	-
LITTLE Voltage (mV)	1150	1112.5	1075	1037.5	1000	962.5	925	900

Fig. 9. Voltage/frequency data for Exynos-5422 chipset.

Second, it predicts the next iteration's execution time and energy consumption. First, the assigned core count and core type from the Approx-RM controller are used to select appropriate rows from the history tables employing multiplexers. Next, an averaging unit computes the mean from all the valid samples in the row. This mean value is used to predict timing and energy using Equation (9) and Equation (12). The resource management controller repeatedly invokes the timing and energy predictions for various configurations (see Algorithm 3) to find a suitable configuration. Finally, the instruction count and the execution parameters, measured while executing on the predicted configuration, are fed back to the prediction history buffer.

4 EXPERIMENTAL METHODOLOGY

4.1 Hardware Platform and Configuration Space

We use as a hardware platform a heterogeneous multicore system with the capability to experiment with different hardware configurations with an interesting range of tradeoffs concerning performance and energy efficiency. To this end, we use *ODROID-XU3*, based on *Exonys 5422* [8] **System-on-Chip (SoC)** employing the ARM big.LITTLE [23] architecture. Figure 8 shows the platform, where four big and four LITTLE processors are arranged in two homogeneous clusters. Processors within a cluster share an LLC, and a **cache coherent interconnect (CCI)** connects the LLCs of both clusters to allow for fast data transfer between the clusters. The SoC features four big (i.e., CORTEX A15 performance-oriented out-of-order) and four LITTLE processors (i.e., A7 energy-efficient, in-order). Each processor has a 32-KB private L1 cache. The *big cluster* has a 2-MB shared LLC, while the *LITTLE cluster* has a 512-KB shared LLC. The available V-F states are shown in Figure 9. Furthermore, lmbench [25] is used to measure the miss penalty (i.e., 48 nono-seconds) for LLC misses.

4.2 Simulation Methodology

A hybrid simulation methodology is employed to evaluate the Approx-RM scheme. As a first step, we record an *execution data-trace*, at the granularity of an iteration, for each application by executing on the real hardware at a fixed configuration. We repeated this experiment on all combinations of V-F, processor type, and processor count to generate the execution traces for each configuration in the system. The traces are used to replay the real execution at a certain configuration. The measured values include the instruction count, cycles, LLC misses, and energy consumption.

In the second step, we simulated the dynamic behavior where configurations can change during execution. The execution data traces feed into a simulator that employs all schemes (one at a time) as a decision maker in the resource manager to pick data from one of the execution data traces. This method simulates the dynamic behavior without the need to execute the workload repeatedly on hardware. The objective is to emulate application execution on real hardware using three resource managers: Approx-RM, Oracle, and Race-to-Idle.

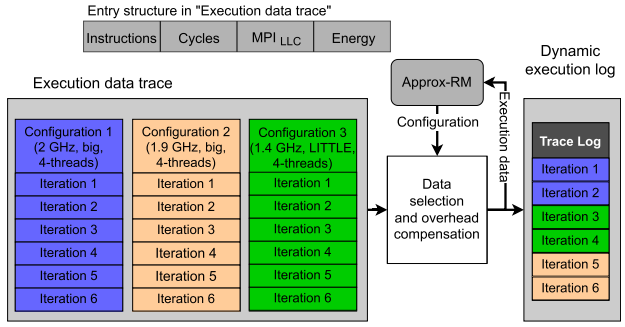


Fig. 10. Simulation methodology.

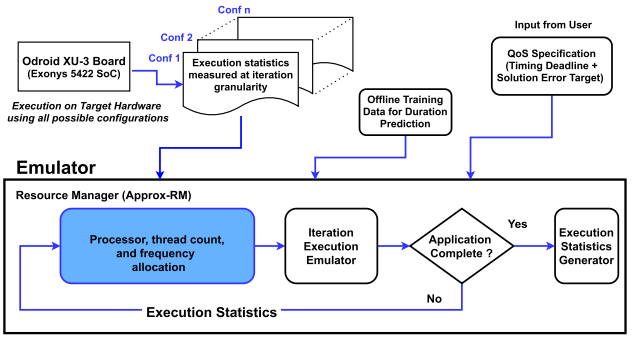


Fig. 11. Experimental setup (adapted from [5]).

An example is shown in Figure 10. Here, Iteration-1 executes on configuration-1 (big processor, four threads, 2 GHz), and execution data for Iteration-1 at said configuration is selected from the execution data-trace. The overhead associated with V-F switching, processor migration, and Approx-RM is added before recording data in the *dynamic execution log* and fed back to Approx-RM for prediction purposes as shown in Figure 4. Finally, Approx-RM uses this data to predict the resource allocations for future iterations. As shown in Figure 10, Approx-RM predicts to execute Iteration-3 using the following setup: the LITTLE processor, four threads, 1.4 GHz. Hence, the data for Iteration-3 at the predicted configuration is selected from the execution data-trace. This data is recorded in the dynamic execution log and fed into Approx-RM. This process repeats until the completion of the application. For each iteration, the data for a predicted configuration is read from the execution data trace and recorded in the *dynamic execution log*. At the end of the application, the dynamic execution log is aggregated to compute the result for the whole execution.

4.3 Experimental Setup

The simulation framework is shown in Figure 11. The source code is instrumented to measure the execution statistics at the iteration granularity by inserting routines to read out data from the hardware performance counters before and after each iteration. The first step is to execute the modified applications on the ODROID-XU3 board on all the configurations. There are 13 V-F settings for the big cluster and 10 for the LITTLE cluster, and two, three, and four are assumed as thread count settings. This results in $13 * 3 = 39$ configurations for the big cluster and $10 * 3 = 30$ configurations for the LITTLE cluster, amounting to 69 configurations. This execution data trace

Table 1. Workloads and QoS Specification

Workload	Name	Iteration Count	Deadline (sec)	Solution Error Target	Average Time per Iteration (msec)	Description	Problem Type
W1	gauss-test	1,473	32.6	529.3	22.13	Gauss–Seidel method	Iterative Solvers (IS)
W2	gauss-testgrind2	1,469	29.9	531.51	20.3		
W3	gauss-testgrind2-regions	1,493	676.7	0.05	453		
W4	gauss-anotniu	1,304	1.52	0.003	1.16		
W5	redblack-test	1,493	50.52	0.01	33.8	Blocked Red-Black	
W6	redblack-test2	1,500	9.72	250.23	6.48		
W7	jacobi-test	1,802	252.71	542.3	141.3	Jacobi method	
W8	jacobi-testgrind	1,845	1,095.41	0.026	593.71		
W9	GD-LR-boston	550	0.88	21.89	1.6	Multi-variate Linear Regression (MVLR)	Gradient Decent (GD)
W10	GD-LR-diabetes	550	0.53	2870.1	0.96		
W11	GD-LR-California	550	31.89	0.52	57.9		
W12	GD-LgR-Cancer	550	1.12	0.17	2.03	Logistic Regression (LgR)	

is then used for runtime simulation. The solution error target and the timing deadline are assumed to be provided by the user.

The simulation uses the execution data-trace to compute the dynamic schedule. First, Approx-RM predicts the configuration for the next iteration. Next, the *Task Execution Simulation* block records the execution of the task from the real execution data-trace in the *Dynamic execution log*. Here, the overheads associated with Approx-RM, processor switching, and DVFS are also added to the execution time and energy consumption of the iteration. Resource allocation for each iteration is established until all iterations have been completed. Once the application is completed, the Dynamic execution log is aggregated by the *Execution Statistics Generator* to compute the evaluation statistics.

4.4 Benchmarks

Our proposed resource management scheme applies to workloads exposing two properties: (1) an iterative execution model and (2) an approximate nature of the algorithm. Since no benchmark suite entirely comprises such applications, we must pick benchmarks from various suites. We have chosen workloads from two sources. First, we use *iterative solvers* for heat diffusion from the **BSC Application Repository (BAR)** [7]. While they are written in OmpSs, we have modified them to comply with OpenMP 4.0, including syntax transformation, adding OMP parallel constructs, and replacing task dependency clauses with barriers. The resulting code is published on the web [3]. Second, we use a microkernel for gradient descent for multi-variate linear regression, implemented in C++ and parallelized using OpenMP. We use publicly available datasets from *sklearn* API [29]. The workloads used are listed in Table 1.

In the context of chosen workloads, we can observe a diverse set of properties. For example, there are workloads with big iteration times, i.e., W3, and some with very short execution times, i.e., W4, W9, and so forth. Similarly, workloads have diverse iteration counts as well.

4.5 QoS Specifications

We use the **Upper Bound Execution Time (UBET)** to determine the QoS specifications for the workloads. UBET is the measurement-based estimation of worst-case execution time [1] and is a reasonable method in soft real-time systems. In this context, the workload's main loop encapsulating kernel is instrumented and executed using a baseline configuration (i.e., big, four threads at 2 GHz) for several runs. The measured execution time per iteration is analyzed to establish the **highest observed execution time (HOET)** across iterations and used as UBET. The deadline is set using Equation (15).

$$\text{Deadline} = D = \text{UBET}_{\text{iteration}} \times \text{Iteration Count} \tag{15}$$

Table 2. Overheads of Key Components in Approx-RM

Mechanism	Configuration Prediction	Duration Prediction
Timing Overhead	2k cycles	1k cycles
Energy Overhead	1,472 nJ	716 nJ

Finally, the solution error target is the solution error of the last iteration. The application duration is set to reduce the solution error by a factor of 100,000. However, for the multi-variate linear regression class of workloads, the solution error will not reduce that much due to variance in input training data. Hence, we let these workloads execute 500 iterations. Table 1 provides the details of the iteration count, deadline, and solution error target for all workloads.

4.6 Evaluated Techniques

Approx-RM is quantitatively compared to the following schemes.

4.6.1 Race-to-Idle (RTI). The Race-to-Idle scheme executes the workload at the fastest configuration and then powers down until the deadline. It executes on a big core with four threads at 2 GHz. It serves as a reference for other schemes.

4.6.2 Oracle. Oracle is used to compare the potential of energy savings and efficacy of the proposed Approx-RM. Oracle has perfect knowledge of the duration of the application, execution, time, and energy consumption. Therefore, it exhaustively searches the configuration space for every iteration to find the optimal configuration from an energy efficiency perspective that meets the QoS specifications.

4.6.3 Oracle-Base. The Oracle-Base scheme employs RTI for the reduced number of iterations. This scheme has perfect knowledge of the iteration count required to meet the relaxed SET. In this scheme, the energy reduction is entirely due to a reduction in iteration count.

4.7 Implementation and Deployment

We envision the Approx-RM to be a runtime resource manager. Depending on utilization, it can be implemented in hardware or software. In the context of this article, we assume that it is a software component that is invoked after the completion of each iteration to make resource allocation decisions. Note that different components of Approx-RM are computed depending on the state as depicted in Algorithm 3. For example, the duration prediction in Approx-RM is invoked whenever the error history buffer is filled with new samples.

Approx-RM, along with all its sub-components, is implemented in C++ and tested on the target hardware platform. This allows us to establish the execution time and energy overheads of the scheme as shown in Table 2.

In order to establish the runtime overheads, an important observation is that the various components of Approx-RM do not execute at every iteration. Thus, we need to record the overhead of a particular component whenever it is invoked within the Approx-RM scheme. The energy consumption and execution time of Approx-RM, including all sub-components, are added to the energy consumption and execution time, respectively, of the workload's execution with the Approx-RM scheme. Thus, savings presented in the evaluation incorporate these overheads, and overheads are presented separately.

4.8 Overhead Estimation

Other overheads, such as DVFS switching, performance-counter reading, and cluster switching, are also incorporated into the results. DVFS overhead is estimated using values presented in [28].

Table 3. Target Values for Prediction Error History to Ascertain the Confidence Interval

Workloads	Cluster	Error History Prediction
W1-W8	1	0.4
W9-11	2	0.9
W12	3	3

Table 4. Configurations Used in Online Training at the Start of Execution

Iteration	1	2	3	4	5	6
Core	big	big	big	LITTLE	LITTLE	LITTLE
Thread	4	3	2	4	3	2
Frequency	2 G-Hz	2 G-Hz	2G-Hz	1.4G-Hz	1.4 G-Hz	1.4 G-Hz

In this context, we considered a simplified worst-case scenario, where maximum up-scaling (i.e., V-F increase) overhead is shown to be 18 μsec and delay for down-scaling (i.e., V-F decrease) is fixed to 10 μsec (i.e., **phased lock loop (PLL)** delay). However, we expect that this overhead is much lower in newer process nodes and on-chip DC-DC converter designs. Furthermore, the value of DVFS overhead selection is not a limiting factor for our approach. The overhead of reading performance counters is measured to be 100 cycles per counter. The cluster switching overhead is dominated by the time required to move the working set from the LLC of one cluster to the other LLC of the other cluster. Here, we make the pessimistic assumption that all data in the LLC of the big cluster is required to transfer to the LLC of the LITTLE cluster or vice versa. Since at most 512 KB is moved (i.e., the size of LITTLE cluster LLC), we assume 35K cycles for a cluster switch (70K cycles to migrate 1,024 KB according to Markovic [24] on similar CMP). Finally, the approx-RM scheme does not introduce any hardware overhead, as performance counters and DVFS are already available in commodity processors.

4.9 User-provided Inputs and Design Parameters

4.9.1 Design Parameters. The solution-error history buffer has 20 entries. The history buffer size "h" for timing and energy prediction is set to five as this provides acceptable accuracy and responsiveness with low overheads.

The clusters for the confidence interval in duration prediction and target for error history prediction are shown in Table 3.

There are six unique processor-thread combinations in the given hardware platform: the processor types (two) times the thread count options (three; two to four threads). Thus, the number of rows in the history tables for predicting execution parameters is six. Moreover, Approx-RM executes the first six iterations on pre-defined configurations to facilitate energy and timing prediction, as shown in Table 4.

4.10 Evaluation Metrics

The evaluation is conducted concerning energy savings, slack, the prediction accuracy of the timing, and energy prediction. Equation (16), Equation (17), and Equation (18) are used to compute these, respectively.

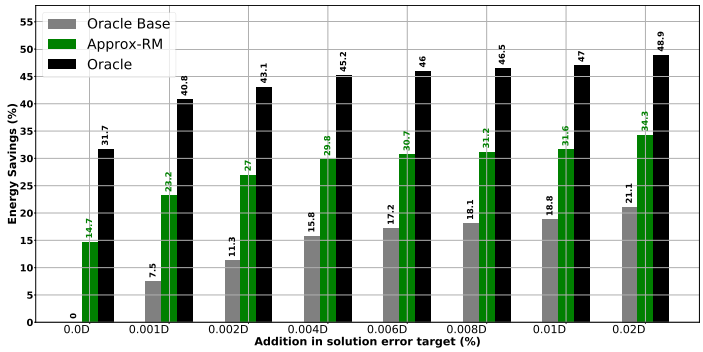
$$\text{Energy Savings}(\%) = 100 \times \frac{\text{Energy}_{\text{RTI}} - \text{Energy}_{\text{scheme}}}{\text{Energy}_{\text{RTI}}} \tag{16}$$

$$\text{Slack}(\%) = 100 \times \frac{\text{Deadline} - \text{Time}_{\text{scheme}}}{\text{Deadline}} \tag{17}$$

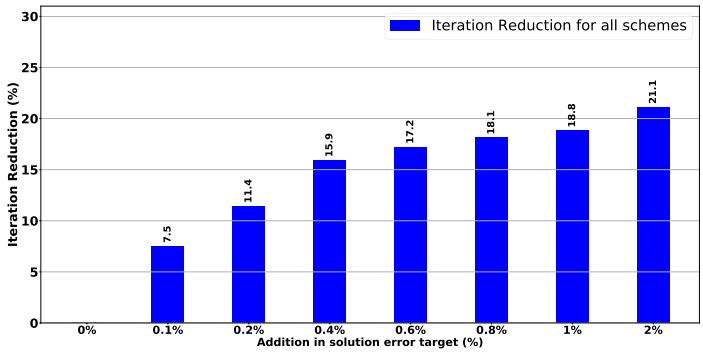
$$\text{Accuracy}(\%) = 100 \times \left(1 - abs\left(\frac{\text{Value}_{\text{Actual}} - \text{Value}_{\text{Predicted}}}{\text{Value}_{\text{Actual}}}\right)\right) \tag{18}$$

5 EVALUATION

We evaluate *Approx-RM* in this section. First, average energy savings across workloads for various settings of a **solution error target (SET)** are presented in Section 5.1. Then, we dig deep into



(a) Average energy savings with addition in SET across all workloads



(b) Average reduction in iterations count across all workloads

Fig. 12. Average energy savings across all workloads (a) and the average reduction in iteration count (b).

per-workload energy savings for a particular SET in Section 5.3. Then, we analyze the computational deadlines and slack usage in Section 5.4. Next, a discussion of the accuracy of timing and energy predictions is presented in Section 5.5. Finally, Section 5.6 analyzes the impact of overheads.

5.1 Average Energy Savings

First, we evaluate the average energy savings and the average percentage reduction in iteration count, for all the workloads, as a function of the percentage relaxation of the SET in Figure 12(a) and Figure 12(b), respectively. The vertical axis in Figure 12(a) represents the energy savings compared to the energy consumed by Race-to-Idle, using Equation (16). The vertical axis in Figure 12(b) shows the percentage reduction in iteration count compared to the total number of iterations. The horizontal axis in both charts represents the percentage reductions in SET.

Figure 12(a) depicts energy savings for *Oracle-Base*, *Oracle*, and *Approx-RM*. The average energy savings for Oracle-Base, Oracle, and Approx-RM are 0%, 31.7%, and 14.7%, respectively, with no addition in SET. The energy savings for Oracle-Base are zero because there is no reduction in iteration count. On the other hand, for a 1% addition in SET, the energy savings increase to 18.8%, 47%, and 31.6% for Oracle-Base, Oracle, and Approx-RM, respectively. This increase in energy savings is because of an 18.8% reduction in iteration count. The iteration count reduction has two effects that contribute to energy savings. First, fewer iterations are executed as depicted by *Oracle-Base*. Second, execution time slack is generated because of the higher allocation of available time till the deadline for each iteration. As SET increases, the number of iterations required to reach the new

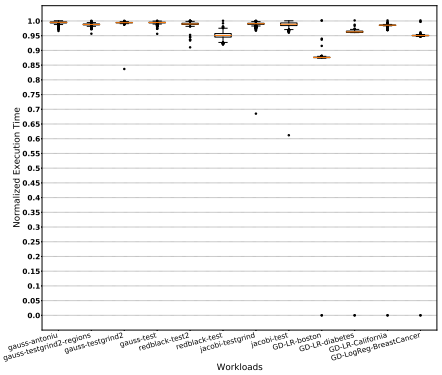


Fig. 13. Box plot of normalized execution time for 0% addition in SET.

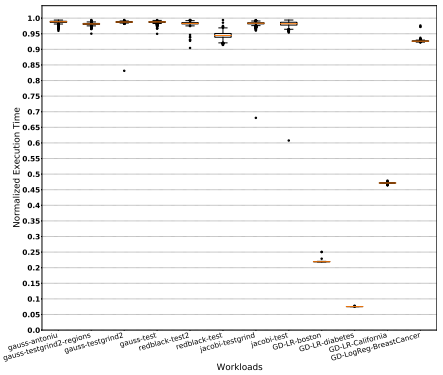


Fig. 14. Box plot of normalized execution time for 1% addition in SET.

target is reduced while the execution deadline remains unchanged. This allows the resource manager to stretch the reduced number of iterations over a more extended period. Hence, the reduced iteration count, while keeping the program's deadline the same, allows the time for individual iterations to increase in proportion to the generated slack. This makes it possible for the resource manager to reduce the resource allocation to save energy.

5.2 Explanations for Energy Savings

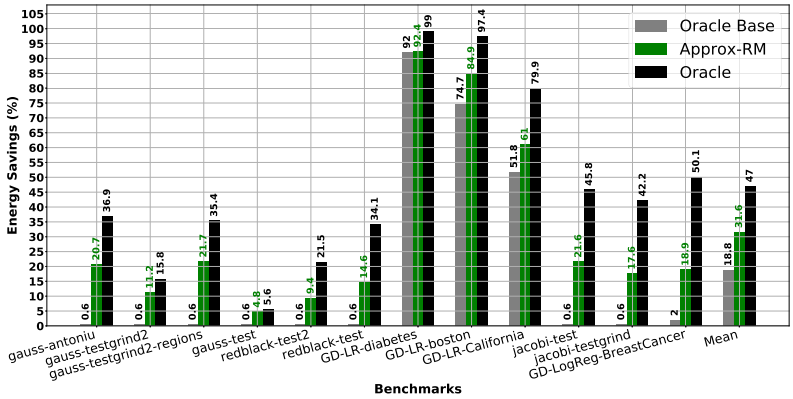
The amount of energy saved depends on the statistical distribution of the execution time per iteration. To elaborate on this, we use box plots to visualize the normal distribution of execution time per iteration compared to the deadline per iteration. Figure 13 and Figure 14 show the execution times per iteration normalized to the deadline per iteration for SET addition of 0% and 1%, respectively. The "box" in a box plot represents the middle 50% of samples ranging from the 25th to the 75th percentile, also referred to as the **interquartile range (IRQ)**. The orange line in the middle of the box represents the median, and the lower and upper whiskers signify the minimum and maximum values, respectively.

There are a few important things to observe here. First, analyzing Figure 13, we can see that there is a small amount of inherent slack. The distribution of normalized execution time for all the workloads is close to 1, or in other words, it is close to the deadline. Second, as the SET is relaxed, more slack is generated by observing Figure 14. This is manifested in an increase in energy savings from 14.7% to 31.6% for the SET increase from 0% to 1% for the Approx-RM scheme. In short, we can conclude that the relaxation in SET generates slack that, in turn, is exploited by Approx-RM to reduce resource allocation to save energy.

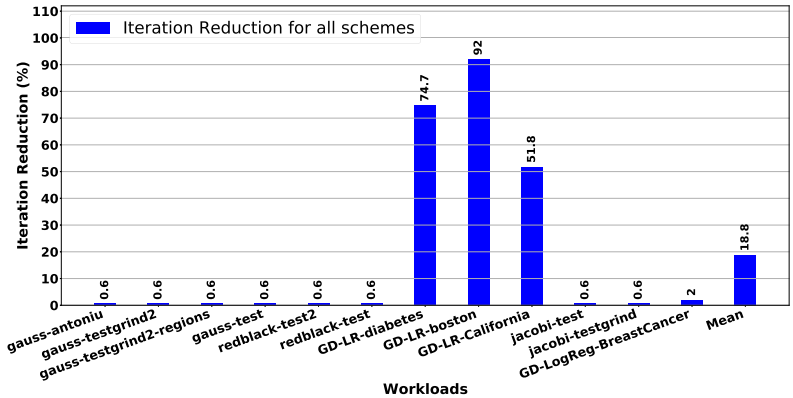
Finally, Approx-RM effectively harnesses the energy-saving potential as it closely tracks Oracle. For example, for the case of a 2% addition in SET, Approx-RM shows energy savings of 34.3% as compared to 48.9% for Oracle. It is important to note that Approx-RM only starts resource allocation after the duration prediction is within the confidence interval. As a result, Approx-RM lags behind Oracle, which starts resource allocation from the first iteration. This is because it has perfect knowledge of the application duration.

5.3 Energy Savings per Workload

Next, we show the energy savings and the reduction in the number of iterations for a 1% increase in the SET in Figure 15. Here, Figure 15(a) shows the energy savings, and Figure 15(b) shows the iteration count reduction. The horizontal axis in both Figure 15(a) and Figure 15(b) represents the



(a) Energy savings for various workloads with 1% addition in solution error target (SET)



(b) Reduction in iterations count for various workloads with 1% additions in solution error target (SET)

Fig. 15. Energy savings (a) and reduction in the number of iterations (b) for a 1% addition in solution error for various workloads.

workloads, while the vertical axis of Figure 15(a) represents the energy savings compared to RTI and that of Figure 15(b) shows percentage iteration reduction compared to the original iterations.

Energy savings for various workloads depend on available slack. The distribution can explain the energy savings for various workloads shown in the box plot in Figure 14. For example, *gauss-test* has most of the samples, i.e., upper to lower whisker, concentrated close to 1 or the deadline. Thus, it shows energy savings of 5.6% and 4.8% for Oracle and Approx-RM, respectively (see Figure 15). On the other hand, *GD-LR-California* has all the samples between 0.5 and 0.55 and thus shows energy savings of 51.8 %, 61%, and 79.9% for Oracle-Base, Approx-RM, and Oracle, respectively.

Looking at Figure 15(b), we can make the following observations. First, different workloads have different amounts of reduction in iteration count owing to their error curves. The solution error of some applications saturates more than others. For example, *gauss-testgrind2* and *redblack-test* only show 0.6% of reduction in iteration count. In contrast, *GD-LR-diabetes* and *GD-LR-California* show a large reduction, i.e., 74.7% and 51.8%, respectively.

Second, the iteration count reduction generates additional slack that can be observed by comparing the two box plots in Figure 14 and Figure 13 and scanning the position of the box that

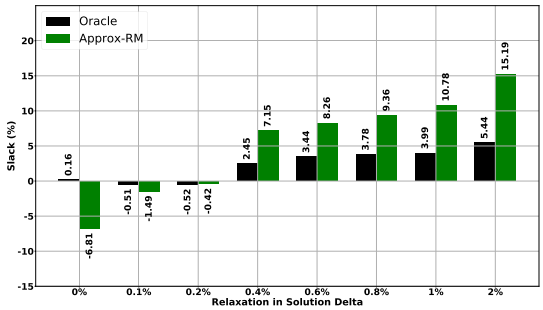


Fig. 16. Execution time slack at the completion of execution represented as a percentage with respect to the deadline.

corresponds to the IRQ or median 50% of samples. The IRQ for *GD-LR-California* approximately ranges from 0.96 to 0.97 (top of the box to bottom of the box) with a 0% reduction and ranges from 0.51 to 0.52 for a 1% reduction in SET. This significant change results from the 51% reduction in iteration count (see the bottom chart in Figure 15). Similarly, the IRQ for *GD-LogReg-BreastCancer* changes from 0.95–0.96 to 0.92–0.94 owing to a 30% reduction in iteration count. Here, it is important to note that the percentage reduction in iteration count is analogous to the slack generation. The execution time per iteration and deadline is different for both workloads.

For *gauss-test* or *redblack-test*, the reduction in iteration count is very small, i.e., 0.6%, resulting in a small addition of slack. Again, the increase in slack is analogous to the percentage reduction in iteration count; consequently, the energy savings are different. Both *gauss-test* and *redblack-test* show a 0.6% reduction in iteration count but exhibit 4.8% and 14.6% energy savings for Approx-RM, respectively. This is due to different distributions of execution time per iteration and inherently available slack, as evident from the box plots.

5.4 Deadline Adherence and Slack Usage

In this section, we evaluate if Approx-RM can meet deadlines and how well the available slack is used for saving energy. Since slack is the difference between deadline and execution, we only show percentage slack in Figure 16, where the vertical axis represents the slack, and the horizontal axis represents the percentage relaxation of SET. *Oracle*, on average, meets the deadline for all SET settings as the percentage slack is positive. Conversely, *Approx-RM* meets the deadline for SET additions of 0.4% to 2% and misses the deadlines at a 0% to 0.2% addition. The reason for this is twofold. First, the prediction mechanism (i.e., duration and performance prediction) has some inaccuracy leading to the allocation of fewer resources than required. Second, the Approx-RM has some overheads resulting in an increase in total execution time. At lower settings of SET, there is not enough slack available due to little or no reduction in the iteration count to cover for these inaccuracies and overheads.

Moreover, Oracle can use most of the slack as it can perfectly predict application duration and execution behavior. In the case of *Approx-RM*, it uses a considerable amount of the available slack for energy savings. This is due to prediction errors in both workload duration and application behavior. Moreover, as mentioned earlier, the Approx-RM only starts resource allocation when duration prediction is mature, contributing to most of the accumulated slack. For example, for a 1% relaxation of SET, *Oracle* has only 4% of slack, but *Approx-RM* has 10%.

Two critical reasons limit the full usage of slack. The first reason is the quantization of the configuration space owing to discrete voltage-frequency pairs and core types. As a result, even with perfect prediction, *Oracle* may not fully exploit the slack as the following low-performance

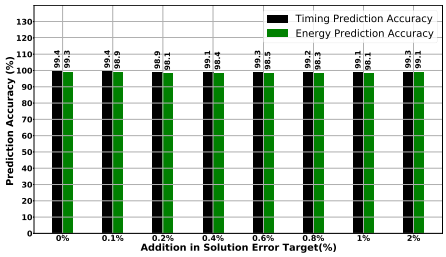


Fig. 17. Average accuracy across workloads for the execution time and energy predictions for various values of SET.

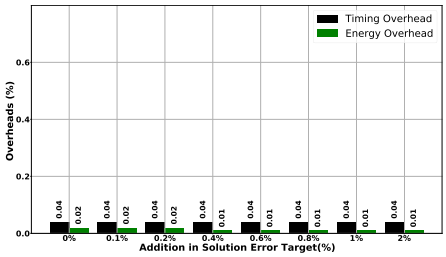


Fig. 18. Average overheads incurred across workloads by Approx-RM (in percent) for various values of SET.

configuration might violate the deadline. The second reason that only applies to *Approx-RM* is the delay in starting resource allocation and the prediction error in both workloads' duration and the the application's timing behavior. *Approx-RM* only starts resource allocation once duration prediction is within the confidence interval. The over-prediction of execution time forces *Approx-RM* to use a relatively high-performance configuration resulting in higher slack at the end of execution. While the duration prediction is not within the confidence interval, *Approx-RM* chooses the highest-performing configuration, i.e., big processor, 2 GHz, and four threads. Moreover, even after the start of resource allocation, the prediction errors in the application's duration, performance, and energy limit the full exploitation of slack. An analysis of the output schedules reveals that *Approx-RM* starts resource allocation after around 55% of the iterations compared to the total number of iterations. The difference in energy savings for *Oracle* and *Approx-RM* is reiterated in their ability to use slack. Regardless, *Approx-RM* saves considerable energy while providing a statistical guarantee to meet the deadline.

5.5 Execution Time and Energy Prediction Accuracy

In this section, we present the accuracy of the execution time and energy prediction mechanism as explained earlier in Section 3.4. Figure 17 depicts the prediction accuracy with various relaxations of SET. Here, the vertical axis represents the prediction accuracy (in percent), whereas the horizontal axis represents the relaxation of SET (in percent). The timing and energy prediction accuracy are both computed using Equation (18). Both predictions show very high accuracy, thus enabling *Approx-RM* to save energy. Both predictions exhibit at least 98% accuracy. Interestingly, the accuracy changes with the relaxation of SET. The reason is that the switching pattern changes with a specific SET target, inducing a different set of samples recorded in the history buffer. This, in turn, affects the prediction accuracy as the count and variety of samples from configurations contribute to the prediction accuracy. Another important insight is that the prediction inaccuracy inhibits slack from being used. A small error in prediction hinders *Approx-RM* from fully utilizing the slack compared to *Oracle*, which has a perfect estimation. Since slack usage affects energy savings, it is fair to say that the small prediction error in the timing and energy prediction mechanism contributes to the difference in energy savings between the *Approx-RM* and *Oracle*.

5.6 Overheads

Figure 18 shows timing and energy overheads. The vertical axis shows the overheads, whereas the horizontal axis shows the relaxation of SET. Timing and energy overheads are computed compared to the Race-to-Idle execution time and energy, respectively. These overheads include DVFS, core switch, Approx-RM, and duration prediction overheads.

In general, the timing and energy overheads are negligible, with maximum timing and energy overheads being 0.04% and 0.06%, respectively, for a 2% relaxation of SET. The reason is that even though Approx-RM is invoked after each iteration, not every component of it is used. For example, duration prediction is only invoked after "K" iterations (i.e., when the error history buffer contains completely new data). Resource Allocation is invoked for every iteration provided the duration prediction is mature. However, the overhead of the resource allocator is very low compared to the execution time of one iteration (i.e., Table 2 and Table 1). Furthermore, configuration changes do not happen in each iteration since after a configuration change, the system remains steady for many successive iterations.

Moreover, the timing and energy overheads of Approx-RM change slightly with the relaxation of SET. This is because more slack is available with an increase in SET, incurring a specific switching pattern between configurations for *Approx-RM*. In short, the overall timing and energy overheads of Approx-RM are approximately less than or equal to 0.1% in all cases.

6 RELATED WORK

Improving energy efficiency under a QoS constraint has been the subject of considerable attention because of higher energy and performance requirements. As a result, the literature abounds with resource management schemes to improve energy efficiency for single-threaded applications such as DVFS [13, 17, 32], thread placement [11, 30], and DVFS and thread placement [6]. For multithreaded or parallel applications, proposals such as [4, 10, 12] use various combinations of DVFS, processor count, and thread placement to save energy. However, these works do not address approximate applications. Sharif et al. [31] provide a framework for exploiting the approximate nature of ML inference by employing optimizations at development, installation, and runtime. However, they do not address approximate iterative applications.

Energy efficiency for AIA has gained considerable attention owing to the possible tradeoffs between accuracy, performance, and energy efficiency. To this end, Vassiliadis et al. [33] propose a framework where the user provides accurate and approximate implementations of the tasks and their significance to the accuracy of the result. Then, an offline profiling tool uses this information to train an application-specific model used at runtime to control the thread count, frequency, and ratio of accurate/approximate tasks to keep the energy below a target specified by the user. Similarly, Hoffmann [15] propose a runtime technique for providing energy guarantees while maximizing accuracy and relying on application-specific accuracy control mechanisms. First, an energy-efficient configuration satisfying the energy budget is found, and the accuracy is changed to meet the throughput target.

Zhang et al. [35, 36] propose two techniques for iterative applications [36] and **artificial neural networks (ANNs)** [35] that provide quality guarantees while minimizing energy. First, both methods build a model using offline characterization to determine the criticality and energy-saving potential of the available approximation means. This model is then used at runtime to reduce energy while ensuring the quality of the final solution. Second, [36] targets approximate iterative applications, as we do in this work, and uses approximate adders. In contrast, the second solution [35] targets artificial neural networks employing memory skipping, precision scaling, and approximate multipliers to reduce energy.

Farrell and Hoffmann [14] propose a method that ensures timing guarantees and energy efficiency. The technique leverages application-specific accuracy affecting parameters to accomplish the approximation. It is based on allocating resources for average-case latency requirements to save energy while meeting the timing requirement by identifying the worst-case application regions and approximating them to reduce the computational requirements.

Dayapule et al. [9] employ offline analysis to obtain a profile of maximum performance (i.e., job arrival rate) versus power efficiency (i.e., jobs per second per watt) for accurate and approximate versions of jobs (or tasks) while satisfying the QoS constraint on tail latency. This offline profile is then used to select a configuration of core types and accurate or approximate job selection to fulfill the tail latency. The quality guarantees are implicitly provided by limiting the number of approximate jobs used in succession.

Kulkarni et al. [21] provide quality and throughput guarantees by employing offline characterization of the design space to identify energy-saving opportunities. Their technique uses dynamic recompilation to configure application-specific approximation modes, core count, and memory allocations to save energy.

You et al. [34] propose a framework targeting GPUs for improving energy efficiency or latency under the quality constraint by varying power and batch sizes. They rely on a regression model to identify the tradeoff between the power and execution time.

The techniques mentioned above provide either timing or energy guarantees but not both. In contrast, *Approx-RM* is the first work that provides a runtime management scheme for energy reduction under QoS constraints while providing statistical guarantees for performance and accuracy for approximate iterative applications.

7 CONCLUSION

This article investigates the prospect of saving energy by exploiting the quality-energy tradeoff in approximate iterative applications. Approximate iterative applications are run on heterogeneous multi-processor platforms under QoS constraints on the timing and solution quality. A key insight that our proposed scheme *Approx-RM* builds upon is the diminishing improvement in the solution quality of approximate iterative applications in the later stages of the execution. A small yet controlled reduction of solution quality requirements can significantly reduce the number of iterations, thus producing execution time slack that can be translated into energy savings. In this article, we have shown that solution quality, along with traditional means to control resource usage such as DVFS, processor type, and processor count, can be used to allocate sufficient resources to gain higher energy efficiency. *Approx-RM* achieves this by first predicting the application duration by recording and applying curve-fitting on the solution error function and then allocating an appropriate amount of resources. *Approx-RM* employs an execution time and energy prediction mechanism with high accuracy (≈98%) to predict the behavior of the HMP configuration space and then allocate an appropriate amount of resources. *Approx-RM* achieves 31.6% energy reduction, on average, compared to Race-to-Idle at only a 1% reduction in solution quality. Moreover, *Approx-RM* meets the timing deadline and only incurs an overhead of at most 0.1%.

REFERENCES

[1] J. Abella, C. Hernandez, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-askasua, J. Perez, E. Mezzetti, and T. Vardanega. 2015. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES'15)*. 1–10. https://doi.org/10.1109/SIES.2015.7185039

[2] Susanne Albers and Antonios Antoniadis. 2014. Race to idle: New algorithms for speed scaling with a sleep state. *ACM Trans. Algorithms* 10, 2, Article 9 (Feb. 2014), 31 pages. https://doi.org/10.1145/2556953

[3] M. Waqar Azhar. 2021. Workloads for Approx-RM. https://github.com/waqarazhar/Approx-RM-Workloads.

[4] M. Waqar Azhar, Miquel Pericàs, and Per Stenström. 2019. SaC: Exploiting execution-time slack to save energy in heterogeneous multicore systems. In *Proceedings of the 48th International Conference on Parallel Processing (ICPP'19)*. ACM, Article 26, 12 pages. https://doi.org/10.1145/3337821.3337865

[5] M. Waqar Azhar, Miquel Pericàs, and Per Stenström. 2022. Task-RM: A resource manager for energy reduction in task-parallel applications under quality of service constraints. *ACM Trans. Archit. Code Optim.* 19, 1, Article 11 (Jan. 2022), 26 pages. https://doi.org/10.1145/3494537

[6] M. Waqar Azhar, Per Stenström, and Vassilis Papaefstathiou. 2017. SLOOP: QoS-supervised loop execution to reduce energy on heterogeneous architectures. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 4 (2017), 1–25.

[7] Barcelona Super Computing Center. 2021. BSC application repository. Retrieved from https://pm.bsc.es/projects/bar

[8] Hongsuk Chung. 2013. Heterogeneous Multi-Processing Solution of Exynos 5 Octa with ARM big.LITTLETM Technology. https://www.semanticscholar.org/paper/Heterogeneous-MultiProcessing-Solution-of-Exynos-5-Chung/0

[9] Sai Santosh Dayapule, Fan Yao, and Guru Venkataramani. 2019. PowerStar: Improving power efficiency in heterogenous processors for bursty workloads with approximate computing. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 175–182. DOI: https://doi.org/10.1109/CloudCom.2019.00035

[10] Daniele De Sensi, Massimo Torquati, and Marco Danelutto. 2016. A reconfiguration algorithm for power-aware parallel applications. *ACM Trans. Archit. Code Optim.* 13, 4, Article 43 (Dec. 2016), 25 pages. https://doi.org/10.1145/3004054

[11] Christina Delimitrou and Christos Kozyrakis. 2013. QoS-aware scheduling in heterogeneous datacenters with paragon. *ACM Trans. Comput. Syst.* 31, 4, Article 12 (Dec. 2013), 34 pages. https://doi.org/10.1145/2556583

[12] B. Donyanavard, T. Mück, S. Sarma, and N. Dutt. 2016. SPARTA: Runtime task allocation for energy efficient heterogeneous manycores. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'16)*. 1–10.

[13] Stijn Eyerman and Lieven Eeckhout. 2011. Fine-grained DVFS using on-chip regulators. *ACM Trans. Archit. Code Optim.* 8, 1, Article 1 (Feb. 2011), 24 pages. https://doi.org/10.1145/1952998.1952999

[14] Anne Farrell and Henry Hoffmann. 2016. MEANTIME: Achieving both minimal energy and timeliness with approximate computing. In *2016 USENIX Annual Technical Conference (USENIX ATC'16)*. USENIX Association, Denver, CO, 421–435. https://www.usenix.org/conference/atc16/technical-sessions/presentation/farrell.

[15] Henry Hoffmann. 2015. JouleGuard: Energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP'15)*. Association for Computing Machinery, New York, NY, 198–214. https://doi.org/10.1145/2815400.2815403

[16] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic knobs for responsive power-aware computing. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XVI)*. Association for Computing Machinery, New York, NY, 199–212. https://doi.org/10.1145/1950365.1950390

[17] C. J. Hughes, J. Srinivasan, and S. V. Adve. 2001. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proceedings 34th ACM/IEEE International Symposium on Microarchitecture (MICRO-34)*. 250–261. https://doi.org/10.1109/MICRO.2001.991123

[18] Connor Imes and Henry Hoffmann. 2015. Minimizing energy under performance constraints on embedded platforms: Resource allocation heuristics for homogeneous and single-ISA heterogeneous multi-cores. *SIGBED Rev.* 11, 4 (Jan. 2015), 49–54. https://doi.org/10.1145/2724942.2724950

[19] Stefanos Kaxiras and Margaret Martonosi. 2008. Computer architecture techniques for power-efficiency. *Synthesis Lectures on Computer Architecture* 3, 1 (2008), 1–207.

[20] David H. K. Kim, Connor Imes, and Henry Hoffmann. 2015. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. *Proceedings of the 3rd IEEE International Conference on Cyber-physical Systems, Networks, and Applications (CPSNA'15)*. 78–85. https://doi.org/10.1109/CPSNA.2015.23

[21] Neeraj Kulkarni, Feng Qi, and Christina Delimitrou. 2019. Pliant: Leveraging approximation to improve datacenter resource efficiency. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*. IEEE, 159–171.

[22] J. Li and J. F. Martinez. 2006. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *The 12th International Symposium on High-Performance Computer Architecture, 2006.* 77–87. https://doi.org/10.1109/HPCA.2006.1598114

[23] I. Lin, B. Jeff, and I. Rickard. 2016. ARM platform for performance and power efficiency — Hardware and software perspectives. In *2016 International Symposium on VLSI Design, Automation and Test (VLSI-DAT'16)*. 1–5. https://doi.org/10.1109/VLSI-DAT.2016.7482541

[24] Nikola Markovic. 2015. Hardware thread scheduling algorithms for single-ISA asymmetric CMPs. *TDX (Tesis Doctorals en Xarxa)*. https://upcommons.upc.edu/handle/2117/96039.

[25] Larry McVoy and Carl Staelin. 1996. Lmbench: Portable tools for performance analysis. In *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference (ATEC'96)*. USENIX Association, 23.

[26] Mehrzad Nejat, Madhavan Manivannan, Miquel Pericàs, and Per Stenström. 2020. Coordinated management of DVFS and cache partitioning under QoS constraints to save energy in multi-core systems. *J. Parallel and Distrib. Comput.* 144 (2020), 246–259. https://doi.org/10.1016/j.jpdc.2020.05.006

[27] Mehrzad Nejat, Miquel Pericas, and Per Stenstrom. 2019. QoS-driven coordinated management of resources to save energy in multi-core systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS'19)*. 303–313. https://doi.org/10.1109/IPDPS.2019.00040

[28] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang. 2013. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 5 (May 2013), 695–708. https://doi.org/10.1109/TCAD.2012.2235126

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[30] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. Mossé, J. Mars, and L. Tang. 2015. Octopus-Man: QoS-driven task management for heterogeneous multicores in warehouse-scale computers. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. 246–258. https://doi.org/10.1109/HPCA.2015.7056037

[31] Hashim Sharif, Yifan Zhao, Maria Kotsifakou, Akash Kothari, Ben Schreiber, Elizabeth Wang, Yasmin Sarita, Nathan Zhao, Keyur Joshi, Vikram S. Adve, Sasa Misailovic, and Sarita Adve. 2021. ApproxTuner: A compiler and runtime system for adaptive approximations. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'21)*. Association for Computing Machinery, New York, NY, 262–277. https://doi.org/10.1145/3437801.3446108

[32] Jinho Suh, Chieh-Ting Huang, and Michel Dubois. 2015. Dynamic MIPS rate stabilization for complex processors. *ACM Trans. Archit. Code Optim.* 12, 1, Article 4 (April 2015), 25 pages. https://doi.org/10.1145/2714575

[33] Vassilis Vassiliadis, Charalampos Chalios, Konstantinos Parasyris, Christos D. Antonopoulos, Spyros Lalis, Nikolaos Bellas, Hans Vandierendonck, and Dimitrios S. Nikolopoulos. 2016. Exploiting significance of computations for energy-constrained approximate computing. *International Journal of Parallel Programming* 44, 5 (2016), 1078–1098.

[34] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. 2023. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)*. 119–139.

[35] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. 2015. ApproxANN: An approximate computing framework for artificial neural network. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE'15)*. IEEE, 701–706.

[36] Qian Zhang, Feng Yuan, Rong Ye, and Qiang Xu. 2014. ApproxIt: An approximate computing framework for iterative methods. In *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*. Association for Computing Machinery, New York, NY, 1–6. https://doi.org/10.1145/2593069.2593092

Received 5 October 2022; revised 17 April 2023; accepted 15 June 2023