IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 5 March 2024, accepted 18 March 2024, date of publication 22 March 2024, date of current version 28 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3380912

RESEARCH ARTICLE

Performance Improvement of Processor Through Configurable Approximate Arithmetic Units in Multicore Systems

SEYED ALI KASHANI GHARAVI AND SAEED SAFARI
School of Electrical and Computer Engineering, University of Tehran, Tehran 14395-515, Iran
Corresponding author: Seyed Ali Kashani Gharavi (a.kashani@ut.ac.ir)

ABSTRACT Multicore systems are utilized in a wide range of applications, from embedded systems to high-performance applications. Controlling power consumption while maximizing performance under the Thermal Design Power (TDP) becomes increasingly important when power density emerges as the key restriction for multicore systems. Dynamic voltage-frequency scaling (DVFS) approaches have been effective in dynamically power control and are commercially accessible. We propose a novel approach to improve the performance of multicore systems by utilizing configurable approximate Arithmetic units. The proposed system includes a machine learning-based framework for online power regulation and quality monitoring of application output. This framework dynamically adjusts the frequency and precision of the Arithmetic units to maximize performance while considering TDP constraints and the desired output quality. The experimental results demonstrate the effectiveness of the proposed approach. Using a floating point approximate Arithmetic Logic Unit (ALU) with three distinct configurations in each core, the multicore system can execute approximable applications up to 19% faster than a precise multicore system, while operating within the same TDP limit.

INDEX TERMS Approximate computing, reconfigurable approximate design, computer architecture, machine learning.

I. INTRODUCTION

Approximate computing is a new paradigm that trades the output quality/accuracy of error-resilient applications for performance, area, power consumption, and energy efficiency [1]. Examples of these applications include multimedia, image processing, video processing, machine learning, digital signal processing, computer vision, and big data analysis. The error resilience of these applications is because they may not require an answer specific to the golden output or maybe functionally perfect even if there are acceptable errors in the output. Other sources of error restoring force can be the noise of input data, iterative characteristics of other algorithms or error-containing outputs, and other levels of recognition by other users [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Mario Donato Marino.

Approximate computing performs a trade-off between accuracy and performance [3]. However, to take full advantage of the approximation possibilities, the hardware approximation system and software-controlled manner must work together [4]. Also, decisions such as the duration and degree of approximation can vary depending on external factors and the data stream of the application's input, which should be considered at runtime. This requires a approximation system that can be quickly switched on-demand in a controlled manner. We propose a quality assurance component in our framework that can also be invoked by the application itself or by supervisory software. This component ensures that the quality of the results is acceptable. Because the application's calculation quality needs may change dynamically during runtime, it is best to adjust the approximation mode (compensation for different degrees of quality effort) to fulfill the quality requirements while avoiding superfluous calculation

labor. The output quality, on the other hand, is largely dependent on the input. Even though it is not impossible, selecting an appropriate approximate mode that always meets the intended quality standards is difficult [5].

Multicore systems can take advantage of approximate computing to improve performance by relaxing the precision of results. Multicore processors are projected to transform Moore's Law into continuing performance growth by incorporating more and more cores [6]. However, in a general application, there are program segments that can be approximated and others which should not be approximated. Therefore, we propose cores that can execute approximately when it is needed. Our proposed core targets floating point operations for approximation because data flows are typically implemented using integer values, while data in a program that is sensitive to approximation is stored in floating point format. Therefore, our core can only perform approximation operations on floating point data.

In multicore systems, the ever-growing power consumption increases the burden of heat dissipation. In order to ensure the safety and reliability of multi- core systems, a TDP constraint is imposed which the system power consumption should not exceed. TDP constraint can be applied based on the preceding observations to ensure the efficiency of a multicore system. As a result, one of the key directions for power/performance optimization is to improve performance under TDP limits [7]. DVFS is being developed to save power at runtime, in addition to static techniques of raising the number of cores. On-chip calculations can execute more energy-efficiently thanks to the smart tweaking of core voltage and frequency (V/F) levels. Finding the optimal V/F level allocation, on the other hand, can be stated as an NP-hard integer linear programming issue [8]. Several methods have been developed to discover a near-optimal solution in polynomial time; however, they are only effective on small multicore systems, not on systems with many cores, and they do not address the problem of budget overshooting [9]. By utilizing approximate processors, we present a framework to improve the efficiency of multicore systems, which can significantly increase the efficiency of this approximate system under TDP and minimum quality constraints.

The primary contributions of our work can be summarized as follows:

- Enhanced Configurable Approximate Processor: In the proposed framework, we present the concept and architecture of a structurally revised approximate processor for more quality-configurable controllability with extended instruction set architecture (ISA).
- Flexible Framework for Improved Performance: For any individual application, the quality-configurable and general-purpose framework provides a great degree of flexibility. In fact, under a user-defined tolerable quality restriction, any error-resilient parallel program that operates on floating-point data can speed up by this framework.
- Leveraging Reinforcement Learning (RL) for Scalability: The proposed framework takes advantage of the RL approaches to improve the approximate computing paradigm using distributed algorithms to increase the system's scalability.
- Intelligent Selection of Approximation Configurations: The appropriate approximation configuration is selected based on the output quality prediction with the highest performance and lowest output quality degradation.

The remainder of this paper is organized as follows: Section II summarizes and reviews the related works. The approximation processor's architecture and the framework's mapping design are provided in Section III, while the framework itself and problem-solving approach is presented in Section IV. Section V contains experimental results as well as comparisons to precise CMP system. Finally, the paper's conclusion is offered in Section VI.

II. RELATED WORK

A. APPROXIMATE PROCESSOR

Many researchers have explored the use of hardware approximation techniques in processors. In particular, Esmaeilzadeh et al. propose a method called Disciplined Approximate Programming to improve the energy efficiency of processors. Their approach involves extending the ISA and microarchitecture to support approximate instructions and data storage. They use dual-voltage operation to save energy on approximate operations. They evaluate their approach using a variety of benchmarks and show that it can provide energy savings up to 43% [10]. In contrast, our work uses approximate hardware to implement approximation with the goal of increasing processor performance. Chandrasekharan et al. designed ProACt, a processor for high-performance on-demand approximate as a general-purpose processor for approximate floating-point operations. They provided an open-source progress framework consisting of a hardware processor and associated software tool chain [11]. In contrast to ProACt, our framework is designed for multicore processors and employs quality assurance mechanism. Felzmann et al. [12] enhanced energy efficiency using approximate computing to present Risk-5, a RISC-V architecture extension that utilized control mechanisms to arrange multiple coexisting approximation techniques and bridge the gap between software and hardware approximations. They present a method for approximating the RISC-V processor. Kanduri et al. showed that DVFS and power gating focus on the power actuation, cooperating on performance and energy. The power management scheme is evaluated using APPX, DVFS, and PG knobs hierarchically [13]. They employ software-based approximation techniques, while the multicore system lacks the ability to execute tasks in an approximate manner. In Peng et al., diverse approximate techniques and designs have been proposed and verified the value of relaxing the average output quality constraint utilizing a library of 8-bit and 16-bit energy-efficient approximate array multipliers with 20 altered settings, which are commonly

used in image and auditory processing applications. Peng et al. also proposed a novel neural network structure called AXNet to rage two NNs to a holistic end-to-end trainable NN resulted in 50.7% more requests and considerable cuts of training time than other current neural network-based approximate computing systems [14]. Rather than utilizing NNs accelerator, proposed framework involves integrating approximate modules directly into the processor, resulting in a versatile approach to approximation.

B. POWER MANAGEMENT

Several power management methods have been proposed for multicore systems, ranging from heuristic algorithms to formal methods [15]. For example, Devaraj et al. proposed a formal scheduler synthesis framework for safety-critical systems on multicores that guarantees adherence to a system-level peak power constraint while allowing optimal resource utilization [16]. Additionally, they proposed a supervisory control-based fault-tolerant scheduler synthesis scheme for real-time tasks on multicores that maximizes fault-tolerance and minimizes TDP [17].

But lately, reinforcement learning (RL) and machine learning methods have been applied to multicore systems, which have demonstrated favorable performance compared to alternative methods [18]. Isci et al. proposed MaxBIPS, which thoroughly searches for the optimum combination of V/F levels that maximizes the performance due to the power constraint. Nevertheless, while MaxBIPS offers a great quality solution, there is no extensible approach [19]. To address the scalability challenges of RL algorithms, Chen et al. propose a DVFS control technique based on Online Distributed Reinforcement Learning (OD-RL). This technique is designed to improve the performance of many-core systems under power restrictions [20]. Wang et al. present focus on improving the energy-efficiency of multicore systems. They propose an online DVFS control technique based on core-level Modular Reinforcement Learning (MRL) to adaptively pick optimum operating frequencies for each core to optimize the overall multicore system's energy efficiency [21]. Xiao et al. address the challenge of scalability of learning based power management through the utilization of a controller based on deep Q-network [22]. Deep Q-learning agent can be used to overcome the scalability limitations of Q-learning in systems with a large number of cores. Imitation learning has also been proposed for use in large-scale multicore systems. Kim et al. present the first architecture-independent Imitation Learning-based methodology for dynamic V/F island (DVFI) control in many-core systems, inspired by the recent success of imitation learning in many application fields and its major benefits over RL [23]. Martin et al. explore the use of imitation learning to minimize temperature in a heterogeneous clustered multi-core processor while meeting user-defined quality of service targets [24]. However, RL requires another step for learning the model and determining the behavior while learning the optimal behavior so that there is no model [25].

III. APPROXIMATE PROCESSOR AND PROBLEM FORMULATION

A. APPROXIMATE PROCESSOR

Our proposed multicore system is a homogeneous NOC-based system consisting of approximate processors. This section first describes the design of these approximate processors and then our framework's problem in this multicore system. The approximate processor's schematic is shown in FIGURE 1.

The Floating-Point Arithmetic Logic Unit (ALU) is designed accurately and approximately with different accuracy, as shown in FIGURE 1. Since in proposed design only one of the two ALUs is actively processing data at any given time, the other ALU can be turned off to save power. In this scheme this is done using a power gating mechanism, which is a technique for selectively turning off parts of a chip to reduce power consumption. Because this processor considers numerous possible approximate units, to operate the processor in approximate mode, at first the needed configuration must be selected, and then the approximated instructions must be executed approximately.

There are several instructions to ensure the quality required by the framework to be added to the processor, which we will described below. When a processor as a core in a multicore system is instructed by the framework to run in approximate mode, it must use its approximate computational units to execute approximate instructions. However, if the quality controller determines that the data is not safe for approximate processing, then all approximate instructions must be executed using precise computational units. As a result, these two criteria have been added to the ISA of the processor as two instructions (FIGURE 1). These two instructions set two flags in the processor named, Data Safe to be Approximate (DSA) and Approximate Mode (AM). The approximate unit is utilized if the result of bitwise AND of these two flags is 1, otherwise the precise computational unit is used. Since we have a configurable approximate ALU, we need a configuration signal called ''config'' in this schematic. This signal is used by our framework to select the appropriate configuration for the approximate ALU.

The processor's ISA has been extended to include support for approximate operations on floating point operands. This means that the processor now has a new opcode that can be used to perform approximate arithmetic operations on floating point numbers.

B. PROBLEM FORMULATION

In our problem modeling process, we begin by constructing a representation of the multicore system. Specifically, we examine a multicore system comprising N processors designed to perform approximate computations. These processors offer a range of M distinct operating voltages. Additionally, the system incorporates an approximate ALU with C different configurations. So, Each processor can be set to operate in one of C+1 modes, where the first C modes are approximate modes and the last mode is the precise mode.

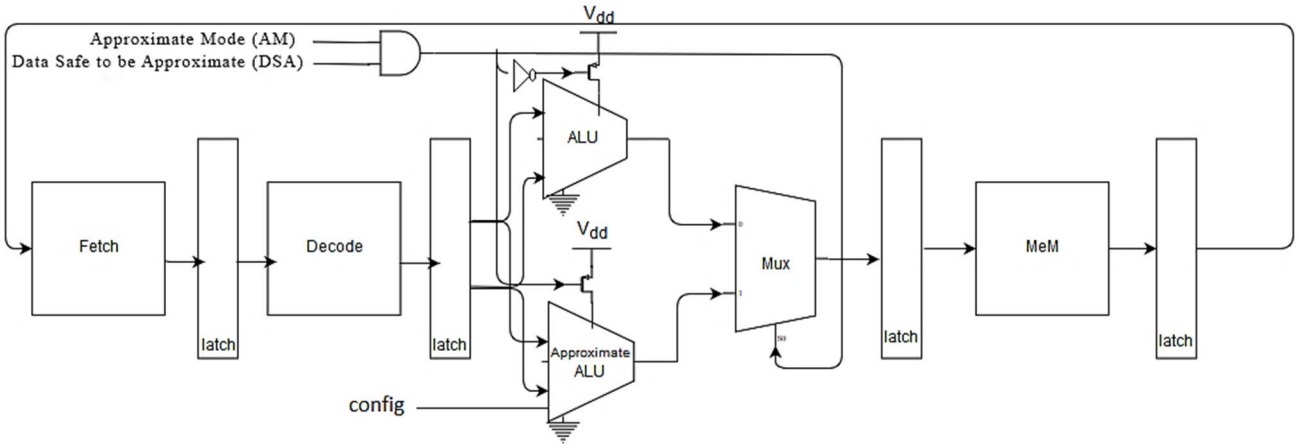

FIGURE 1. Approximate processor architecture.

We define the set of approximation levels as:

$$appx = \{appx_i \mid i = 1, \ldots, C\} \tag{1}$$

where $appx_i$ represents the i_{th} approximation level. Each approximation level corresponds to a specific degree of accuracy and speed. The approximate levels are sorted within this set based on their accuracy and speed.

In this processor we also consider to have M different operating voltages defined as:

$$Voltage = \{V_i \mid i = 1, \ldots, M\} \tag{2}$$

From (2) and also the result discussed in Section V, it could be possible that different operating voltages and frequencies be used for the various configurations listed in equation (1), as well as for the precise ALU. Therefore, each core in this system can operate at any of (C+1)·M frequency levels. So we can define frequency levels as follow:

$$Freq = \{F_i \mid i = 1, \ldots, (C+1) * M\} \tag{3}$$

Combining the information derived from (1), (2), and (3), we are able to establish a comprehensive set of operating conditions for each core in our processor. This set, known as Operating Condition Table (OCT), encompasses the various modes, operating voltages, and frequency levels that each core can adopt.

The OCT can be represented as follows:

$$OCT = \left\{ (mode, V_i, F_j) \;\middle|\; \begin{array}{c} mode = \{app_1, \ldots, appx_c, exact\} \\ i = 1, \ldots, M \\ j = 1, \ldots, (C+1) * M \end{array} \right\} \tag{4}$$

The parallel application capable of being approximated that can run in this system can be modeled as a graph G = (T, E, A), where:

- $T = \{t_i \mid i = 1,2,\ldots, k\}$ is a set of k tasks which are graph nodes.
- $\{E_{ij} = e(i, j)\} \in$ E (DAG) is a set of graph edges, where $e_{ij} = 1$ if task t_i is dependent on task t_j, and 0 otherwise.
- A = {A_i | i = 1,2,…, k} is a set of 0s and 1s label corresponding to the tasks, where $A_i = 1$ if task t_i can be approximated, and 0 otherwise. A task t_i considered an approximate executable task if all of the floating-point operations used in t_i are approximate operations. Programmers should use approximate operations whenever possible.

The goal of the framework is to minimize the completion time of the application. In other word if we consider the completion time of each task$_i$ as finish$_i$, our goal is as follow:

$$minimize \quad \max_{1<i<k} finish_i \tag{5}$$

The framework is intend to achieve (5), subject to the following constraints:

- The maximum power consumption during execution does not exceed TDP.
- The output quality of the program is not less than Q_{min} that is the lowest acceptable quality.

IV. METHODOLOGY

RL algorithms are designed to identify the best solution to a sequential decision and optimization problem, and they have been proven to work in a range of situations. When the problem is complex, dynamic, or has a large search space. RL algorithms can learn from their experiences and improve their performance over time, making them well suited for optimization tasks. However, because they are prone to state-space explosion, they are prohibitively expensive to use in large-scale issues. The purpose of RL is to determine the best actions in various states so that the agent can maximize the long-term benefit by pursuing those best actions by adopting a policy π(s, a) which chooses an action under the conditions of a state s. By experiencing the states, RL determines how the agent modifies its policy. An agent aims to achieve π in a system interaction such that in each state, the system chooses the action with the higher long-term reward [26]. Because

1. It converges to the Bellman optimal solution in an online and incremental manner.

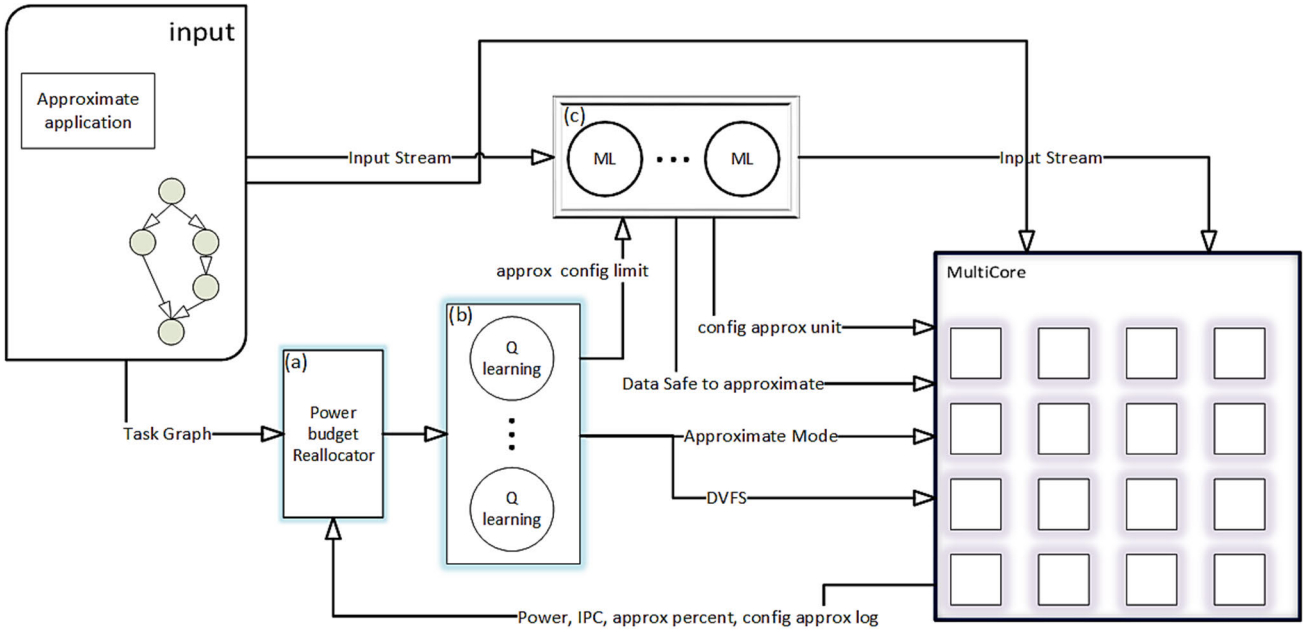

FIGURE 2. Power management framework for approximate multicore processor.

2. It does not require the model of the system, e.g., prior information.

Q-learning [27] is one of the most important achievements in RL. (6) give the updating rule of Q-learning:

$$Q_{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \propto * \left[r_t + \gamma \max_{a \epsilon A} Q(s_{t+1}, a) - Q(s_t, a_t) \right] \quad (6)$$

where α is the learning rate coefficient, γ is the discount factor coefficient between 0 and 1,*and* r_t is the received reward or penalty. In any learning-epoch t, the agent chooses an action related to the system state (s_t). The agent starts from state s and chooses action a. By observing the reward r and the next state s_{t+1}, $\left[r_t + \gamma \max_{a \epsilon A} Q(s_{t+1}, a) \right]$ gives the expected long-term reward of the state-action pair (s, a). Then Q-learning updates the Q value incrementally, which is suitable when the agent experiences the state-action pairs sequentially.

Herein, the states and actions are determined globally. We suppose that the system has N cores with K features per core. Each core i in the system has k features, denoted by f_{ik}. The state space of a single core is the set of all possible combinations of values for its features. The size of the state table is as follow:

$$|S_i| = \prod_{j=1}^{K} f_{ij} \quad (7)$$

Since all cores are identical, the total number of possible states in an N-core system is $|S|^N$. The action of all N cores will be a vector $A_{global} = (a_1, a_2, \ldots, a_N)$, where the a_i is the action of the i_{th} core. As a result, in the centralized RL technique, the total number of state-action pairs grows exponentially as the number of cores N grows. Decomposing the actions of separate cores, in which case each core makes its own independent decision under the global state, is one technique to increase the linear development of the number of state-action pairs. The number of states is decreased from S^N to SN in total. As a result, the RL-based approach to improving performance is combined with other algorithms and divided into two phases:

- Each core learns the optimum policy that maximizes performance under that budget, within its allocated budget fraction of the overall budget.
- Reallocate the power budget with heuristic algorithm, at a coarser level to ensure greater global usage of the overall budget.

This framework is designed to meet our goal for a flexible and scalable system and cover various applications. This framework, as shown in FIGURE 2, consists of three general parts. Sections (a) and (b) are executed at each time interval, but section (c) must be executed whenever input data has been changed. The programmer can embed section c as part of the program. In the following, we describe each of these three sections and their tasks.

A. POWER MANAGEMENT AT THE CORE LEVEL: REINFORCEMENT LEARNING ALGORITHM

Each core will use Q-learning to learn the best policy. The Q-learning algorithm takes as input the state of the core, the approximability of the task, and the available power budget. It then outputs an action that the core should take, such as setting the DVFS configurations and enabling/disabling approximate mode that signal to the AM signal of the core.

For each core, if the assigned task is approximable and the model located in box (c) determines that it is safe to

execute in approximate mode, it signals the DSA and sets the appropriate configuration from (1).

For each core, the frequency is assigned according to the voltage and the pairs we defined in (4).

After each epoch, the Q-learning algorithm receives a reward/penalty from the algorithm in box (a).

We defined the state space of each Q-learning machine as:

$$S = \{IPC, CP, CVF, CAEP, AS, MFC\} \tag{8}$$

where *IPC* is instruction per clock, *CP* is current power, *CVF* is current V/F, *CAEP* is current approximate execution percentage, *AS* is approximate speculation and *MFC* is most frequent configuration used in previous time interval. The state space can include various features related to the core, task behavior, and input data stream. To ensure that the state space remains acceptable for the DVFS controller, we carefully select parameters that provide sufficient information while keeping the state space size small. Also Traditionally, the features are discretized equally [28]. Instead, we discretize feature like IPC by its statistical distribution to minimize inter-interval oscillation due to poor discretization threshold. This means that the bins are sized according to the frequency of the IPC values.

We define the reward r as the core throughput, which stands for the preference of the action that leads to higher performance. Moreover, the budget-overshoot penalty is calculated as:

$$penalty \propto |Power\ Value - Power\ Budget| \tag{9}$$

So penalty is proportional to the value of budget overshoot, which follows the intuition that it is more desirable to eliminate a higher overshoot.

When approximate tasks are considered, one scenario is the oscillation between the processor's actual mode in each epoch. Because task execution times over bunch of input data are sometimes shorter than algorithm execution times, reducing the time interval between algorithm executions is not justified by more than a certain amount due to overhead. The system must be flexible and has the ability to change the mode to control data quality. In order to prevent the system from constantly fluctuating between the approximate configurations and precise states, we consider allowing each core to change modes up to four times. This is to avoid constantly fluctuating between the approximate and precise states, while still allowing for some changes to the system. Therefore, the time interval of the power controller should be adjusted so that the approximate tasks cannot be changed more than four times. If we do not consider this mechanism, the agent itself will learn not to use the approximate execution mode for very small tasks that frequently change state. Approximate speculation is one of the issues that Q-learning considers for approximate tasks. The approximate capacity of the system input is estimated for the future, which is calculated by part of its power allocation algorithm in box (a). The latest feature of the system, which is only used in approximate mode, is called the MFC. This feature specifies the maximum configuration used in the previous epoch.

B. POWER BUDGET REALLOCATION BETWEEN CORES

As distributed RL maximizes performance within a given budget at a finer grain, the power budget reallocation algorithm shown in box (a) of FIGURE 2 will act as a global coordinator to maximize performance and power usage at a coarser grain. The budget reallocation algorithm estimates the power consumption of all cores at their lowest V/F level at each power reallocation epoch and subtracts this number from the total budget calculating extra power. It is shown in lines 1-4 of Algorithm 1. Then, the algorithm creates a max-heap [29] from pairs (i, IPC$*\sqrt[\beta]{\text{number of dependent task}}$) and selects the core with the highest priority based on the IPC value and also task dependency as shown in line 5 of Algorithm 1. In this formula, the core number is represented by i and β is an experimental coefficient reducing the effect of task dependency. At each iteration, the budget required to raise the highest priority core to the highest V/F level is determined and assigned to the core. If there is still a budget left over after the first allocation, the algorithm will choose the core with the second-highest priority and reallocate it to allow for the highest V/F level selection. Our algorithm as shown in lines 6-17 of algorithm 1, repeats this process until no extra budget value is left. To estimate the power consumption of core i at a different V/F level $\mathrm{VF_i} = \mathrm{q}$, we use the V^2f Scale Law, which states that dynamic power is proportional to $\mathrm{V^2f}$ where V is voltage and f is frequency. Assuming core i is originally at $\mathrm{VF_i} = \mathrm{p}$, we estimate:

$$Pow\,(i.q) = Pow\,(i.p) \cdot \frac{Volt^2\,(q) \cdot freq\,(q)}{Volt^2\,(p) \cdot freq\,(p)} \cdot \beta\,(i) \tag{10}$$

where β(i) is the discount factor of core i accounting for the transition cost of V/F level, and Volt (p) and freq(p) are the voltage and frequency of V/F level p, respectively. Because subthreshold leakage power is usually stable for a given voltage, we subtract it from total power first, then do the $\mathrm{V^2f}$ scale on dynamic power, and then add the leakage power of the new voltage back to total power to improve power estimation accuracy. The subthreshold leakage power of different voltages is dignified during the idle machine period.

Another feature of this algorithm is to guess the approximate execution. Only if the approximate task is executed on core i, this estimation needs to speculate the V/F level. If the following epoch data is also estimated to be approximate, the maximum V/F value is set to the average frequency of the approximate configurations based on the log acquired from the approximate configuration in the previous period. Several speculation methods can be used here according to the interdependence of the data in some algorithms. However, we assume that the data behaves similarly to the prior epoch to consider the overall state of the problem. As a result, we expect random data to mislead the results of this section 50% of the time.

As depicted in Algorithm 1, the incorporation of a max-heap for core prioritization serves as the primary contributor to the computational complexity, yielding a complexity of *Nlog(N)*. This approach effectively mitigates the exponential growth of the state space by employing distributed Q-learning. Additionally, Algorithm in box (b), under worst-case scenarios, exhibits a complexity of O(N·log(N)). Consequently, these factors contribute to the scalability of our system, enabling seamless expansion of the processing core count.

Algorithm 1 Budget Allocator

Input: IPC, DAG, Global Budget, N
Output: Budget i for a core i, 1<i<N
variable: residual Budget

1. for i=1,..,N do
2. $\text{Budget}_i \leftarrow$ PowerEstimator(i,V/F_i =lowest)
3. end for
4. residual Budget ← Global Budget - $\sum_{i=1}^{N} Budget_i$
5. Build a MAX-heap of core-feature based on IPC values ∗ power(DependentTasks(DAG, getCoreTask()), $1/\beta$)
6. while residual Budget > 0 do
7. pop the max-heap and get the tuple (i, core-feature i)
8. speculate task approximation to set the maximum V/F(config approx log)
9. Δ ← PowerEstimator(i,VF_i = highest) - PowerEstimator(i,V/F_i =lowest)
10. if Δ <= residual Budget then
11. $\text{Budget}_i \leftarrow \text{Budget}_i + \Delta$
12. residual Budget = residual Budget - Δ
13. else
14. $\text{Budget}_i = \text{Budget}_i$ + residual Budget
15. break
16. end if
17. end while

C. QUALITY MANAGEMENT

Here, we must specify minimum acceptable quality. The problem of whether the data output in a given approximate mode meets the minimum stated quality can be described as a classification problem, and various machine-learning methods can be employed to address it. As a result, a machine learning model is learned here in box (c) of FIGURE 2 for each application, which can declare at any point, based on the program's input data, whether the user's acceptable quality is predicted to observed if the task is completed approximately. Assuming that the declared mode is appx_i, the procedure for this section is that as long as $i > 1$, the appx_i ML must determine whether the data has an acceptable quality. If the quality condition is met, the appropriate appx_i is selected to configure approximate ALU. Otherwise, the value of i is decreased by one, and this process continues. Finally, if the quality condition is not met in any approximate case, the precise calculation mode is selected. In other words, this section is looking for a configuration that has a higher frequency and guarantees quality conditions within the declared quality range. Because some programs contain small tasks executed multiple times with different data throughout an epoch interval, this component must be implemented by lightweight machines. As a result, we choose to learn the simplest machine learning algorithm for each program and approximation configuration that produces high accuracy in predictions for use in this section.

V. EXPERIMENTAL SECTION

A. EXPERIMENTAL SETUP

The proposed framework is implemented in the Sniper simulator, a CMP simulator that supports the RISC_V processor [30]. The instructions mentioned in section III that are added to ISA are also added to the simulator. We also use our approximation instruction to code the approximate parts of the Axbench [31] benchmarks and also convert them into parallel programs.

Our proposed framework is also implemented by Python script in the simulator. These scripts are called at each time interval and set V/F and operating mode of the processors. To take into account the execution time of the quality management part, we embed it to the program codes as a task so that its execution time is included in the program execution time. To achieve optimal results in RL algorithms, it is crucial to justify certain parameter choices. For instance, the value assigned to the penalty parameter significantly influences the performance and budget overshoot in RL. To determine the most suitable values for these parameters, we conducted a comprehensive study across various benchmarks and core configurations. Through rigorous experimentation, we identified the values that yield the best outcomes, ensuring the effectiveness and efficiency of our approach.

To estimate the processor's operating frequency in various modes, we implemented the approximate processor by extending the RISC_V processor [32]. In this processor, we considered three different configurations for floating-point approximations. In Conf1, we used the LREA(32,8) adder [33], the Drum16 [34] multiplier, and the approximate 16-bit divider presented in [35]. In Conf2, an LREA(16,4), DRUM12 multiplier and 12-bit divider, and Conf3, LREA(16,4), DRUM8 multiplier, and an 8-bit divider were utilized. Within our framework, designers have the flexibility to choose any desired arithmetic unit. In our pursuit of enhancing the processor's frequency, we focused on identifying units that possess shorter critical paths during the synthesis stage. Additionally, we considered the computational accuracy of these units in relation to competing alternatives as another criterion for selection between the arithmetic units [36].

To obtain the operating frequency, the above processors are implemented and synthesized using 45nm Nangate library [37]. In each operating mode, it is assumed that modules that should be off are eliminated. As shown in Table 1, the operating frequencies of the processor are obtained in the approximate and precise modes.

TABLE 1. V/F of processor in each operating mode.

Voltage (v)	Precise mode Freq. (MHz)	Approximate conf1 Freq. (MHz)	Approximate conf2 Freq. (MHz)	Approximate conf3 Freq. (MHz)
0.95	590	700	730	770
1.1	710	850	900	970
1.25	1090	1350	1380	1420

TABLE 2. Benchmark setups and ml model for quality assurance.

Benchmark	Domain	Error metric	Error bound	ML model for QoS
Inversek2j	Robotics	Relative error	0.01	Logistic regression
Jmeint	3D gaming	Absolute error	0.01	Logistic regression
Sobel	Image Processing	Image diff	0.01	Logistic regression
FFT	Signal processing	Absolute error	0.05	Logistic regression
Kmeans	Machine learning	Image diff	0.01	Logistic regression
Jpeg	Image processing	Image diff	0.01	Logistic regression
Black-Scholes	Financial analysis	Relative error	0.01	Decision tree

B. RESULTS AND ANALYSIS

The efficiency parameters of the proposed framework are studied in this section. In order to establish a standardized and equitable experimental environment, we integrated the approximate processors into the Sniper simulator. To maintain consistency and establish a level playing field for all experiments, we conducted evaluations using an 8-core multicore processor. This selection ensures that the comparative analysis of different approaches is conducted under equal conditions, enabling meaningful and reliable conclusions to be drawn from the results. In addition, many classification algorithms have been tested for implementation in the quality recognition section on the input data for various applications. As previously stated, the purpose of this section is to guarantee that the output is of acceptable quality in the event of an approximate implementation. In Table 2, the best algorithm in terms of accuracy and simplicity is chosen for each benchmark. This table also shows the error bound considered for each benchmark and shows the metric to measure the program's output quality.

If approximate methods are deployed, our suggested system, according to the results of Table 1, can run error-resilient programs more quickly. It is designed to use this feature and improve the system's efficiency and is responsible for quality assurance according to the error bound and to ensure system constraints, which is TDP here. To evaluate this framework, we have compared it with the system that just has one approximate configuration. The next comparison is with the case where the approximate calculation dimension is omitted from the problem.

The framework and solution known as QQ in [20] are used by changing it and adjusting it to our non-approximate state's framework. As a result of these modifications, we refer to the framework as QQ if the approximation feature is removed from our framework, and we compare the output of the two modes. We also evaluate our proposed system to the situation that we have an approximation, but without configuration, which means there is only one situation in approximation. We consider conf1 as the ALU in this situation.

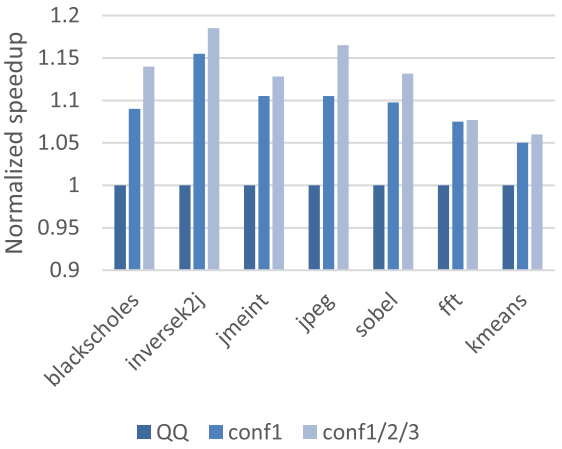

FIGURE 3. Comparison of normalized performance.

To evaluate the effectiveness of our system, we first analyze its performance improvement, adherence to TDP regulations, and output quality. We conduct a comparative study between the proposed framework and the Conf1 and QQ frameworks. By evaluating these factors, we aim to determine whether our system is superior in terms of its overall performance.

FIGURE 3 compares the system's normalized performance with the QQ technique. Using just Conf1 in this framework, we are able to increase the efficiency of the CMP system from 5% to 15%, while with the three mentioned configurations; we can increase the system's efficiency from 6% to 19%.

Different parameters affect the efficiency of this framework. The percentage of approximate code in the overall code is the first aspect to consider. This proportion changes according to the application. An application can also loop over an approximate kernel regularly for various data. As a result, the number of input data for a certain application can affect this percentage. We used a large enough amount of data in this comparison to get the most benefit from the estimated performance.

Another factor is the percentage of data approximated under the quality conditions listed in Table 2. Here we did not change the approximation ratio in the input data of the Axbench applications. Another important factor is the distribution of inputs that can execute approximate and accurate inputs and predicting the possibility of approximate or accurate execution. To reduce the effect of the inputs on the prediction accuracy. The more accurately this algorithm estimates, the better the policy Q-learning algorithm is, and the more efficient the framework is expected to be. Rewards and penalties are other factors that affect the performance of the system. Higher penalties lead the system policy to

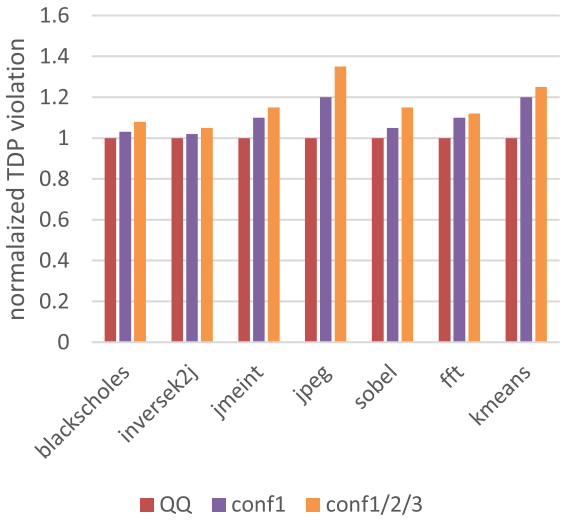

FIGURE 4. Comparison of power overshoots.

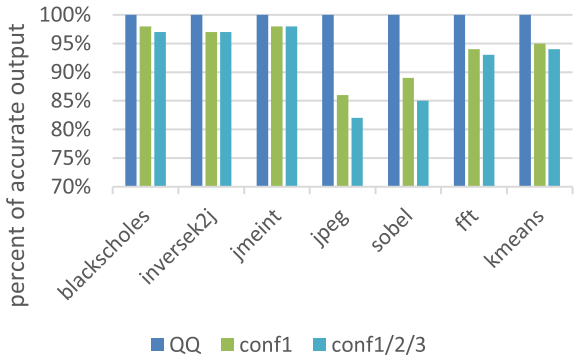

FIGURE 5. Comparison of accuracy degradation.

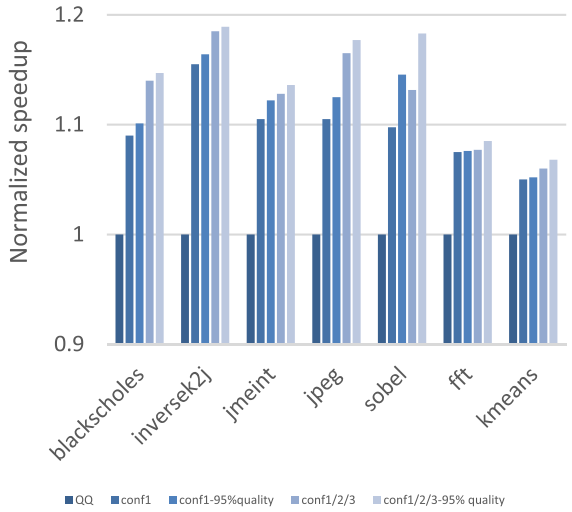

FIGURE 6. Comparison of speed up with 95% target output quality.

mitigate the TDP condition better. In contrast, a lower penalty will further increase the system's efficiency and, in return, will have less commitment than the TDP. The effect of modifying the value of those results is not substantial because we considered this penalty to be the same for both methods, and the results were normalized.

A comparison of TDP violations normalized to QQ is shown in FIGURE 4. This violation increases the burden of the cooling system and has a negative impact on the reliability of the system. In benchmarks, the blackscholes and inversk2j have the least increase in the amount of overshoot, so that in the case that we have only one configuration, this happened just 3% more than in the QQ mode, and in the case that we have three configurations, it has increased by 5% to 8%. On the other hand, in Kmeans, jpeg, and Sobel, we have the highest percentage increase in TDP violation compared to QQ mode. In these benchmarks, in Conf1 mode, overshoot has increased by about 20%. In the case of the three configurations, however, this increase has risen to 32%. This overshoot can be mitigated for these benchmarks by increasing the penalty in the Q-learning method. As another factor, the more correctly the approximate execution speculation method predicts, the lower the amount of violation.

The percentage of outputs for which the system maintained the quality criterion is shown in FIGURE 5. The QQ algorithm produces outputs with perfect precision; yet, because our system can execute instruction approximately, the quality criteria may not be met. Except for the jpeg and sobel benchmarks, the system has convinced us that the outputs are acceptable in conf1 mode (about 94%) and the three configuration modes (around 93%). The input data has been down-sampled in these two benchmarks to make the quality management unit as lightweight as it can. In truth, there is a trade-off between improving performance and maintaining output quality in some applications. In other words, the more confident we are in quality assurance, the more complicated the machine will become, resulting in overhead when the application is run.

In the next evaluation, we allowed the system to have up to a 5% decrease in output quality. We compared our framework to the Conf1 and QQ architectures in terms of performance improvement, adherence to TDP regulations, and output quality.

FIGURE 6 shows the increase in system efficiency in exchange for accepting 5% lower output quality compared to Table 2. The results of this practical test show that in the case of the three configurations in all benchmarks, the system can benefit more from its approximation ability to increase efficiency than in Conf1. Of course, if we consider the percentage of increase in efficiency, in some benchmarks such as kmeans, Conf1 has experienced more growth. It can also be seen that some benchmarks such as Sobel have benefited the most in this situation. As we can see, by accepting a 5% more drop in quality, we can increase the efficiency of the system by an average of 1.25% to 1.4%. In general, it can be claimed that the efficiency of the system can be increased by accepting the loss of quality. Of course, as we can see in FIGURE 7

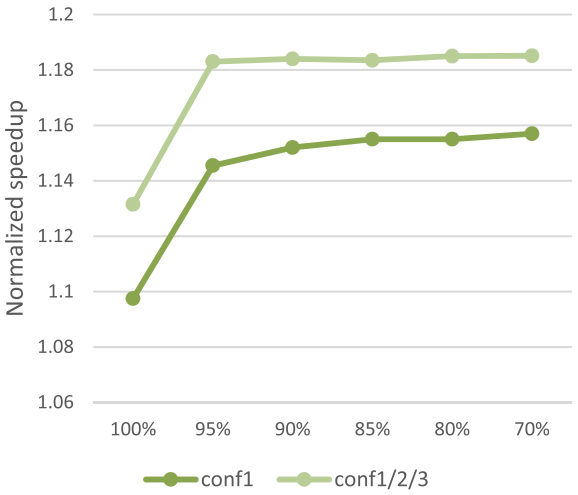

FIGURE 7. Sobel speed up in comparison with quality degradation.

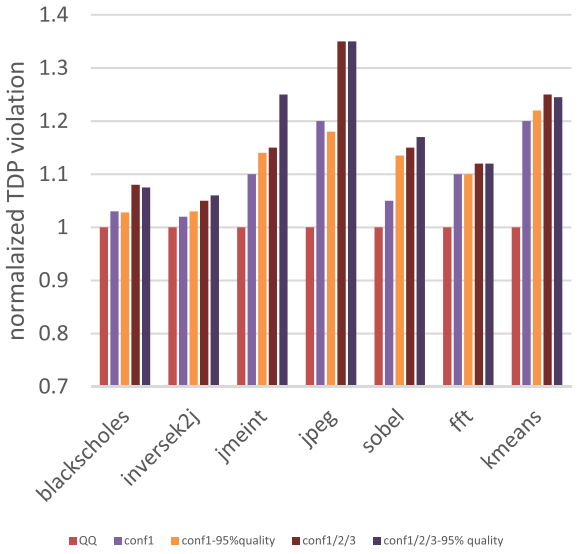

FIGURE 8. Comparison of power overshoot with 95% target output quality.

about the Sobel benchmark, other system limitations prevent the increase in efficiency by decreasing the quality.

In FIGURE 8, we can see that this increase in efficiency, taking into account the quality loss, did not mean more violations of the TDP limit. Although TDP violations have increased in some benchmarks such as jmeint, no significant difference can be observed in most benchmarks. The same conditions are true for the accuracy of the outputs. As we can see in FIGURE 9, this decrease in quality finally up to 3% was able to affect the number of outputs that did not meet the desired quality, which even had a positive effect in some benchmarks and configurations.

VI. CONCLUSION AND FUTURE WORK

We propose the utilization of approximate processors in multicore systems, demonstrating that they can effectively operate in approximate modes at higher frequencies. To address

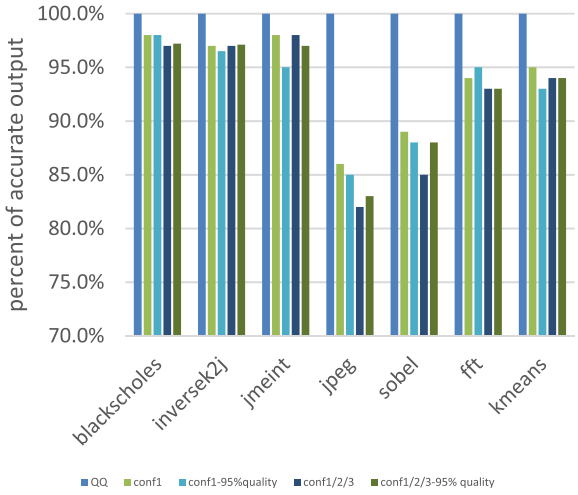

FIGURE 9. Comparison of accuracy degradation with 95% target output quality.

the challenge of TDP limitations and ensure output quality in approximate mode, we introduce an online framework that leverages machine learning techniques. This framework consists of two components: one dedicated to assuring output quality, and the other employing the amplification RL algorithm to determine core frequencies and operating modes based on power budgets. Our experimental results showcase the potential of this approach, revealing that by employing three distinct approximate configurations, we achieve a notable 19% boost in multicore system efficiency while maintaining a high reliability of 83% compared to an acceptable quality threshold. However, it is important to note that there is a trade-off, as TDP violations increase by up to 32%.

As for future work, we plan to focus on further enhancing the power controller of the multiprocessor system. Additionally, we aim to explore the integration of approximate accelerators alongside the multicore system. These future endeavors will contribute to advancing the performance and efficiency of multicore systems through the application of approximate computing methodologies.

REFERENCES

[1] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Des. Test. IEEE Des. Test. Comput.*, vol. 33, no. 1, pp. 8–22, Feb. 2016.

[2] K. S. Mohamed, "Approximate computing: Towards ultra-low-power systems design," in *Neuromorphic Computing and Beyond: Parallel, Approximation, Near Memory, and Quantum*, 2020, pp. 147–165. [Online]. Available: https://scholar.googleusercontent.com/scholar.bib?q=info:AudPmWNsQfwJ:scholar.google.com/&output=citation&scisdr=ClEC6nSkEMyy8gCXbs4:AFWwaeYAAAAAZf6Sds46N806zAP0HbH0iczABvk&scisig=AFWwaeYAAAAAZf6SdmhaWBxXFOP8q0DwFaUVGjg&scisf=4&ct=citation&cd=-1&hl=en and https://link.springer.com/chapter/10.1007/978-3-030-37224-8_5

[3] V. Gupta, T. Li, and P. Gupta, "LAC: Learned approximate computing," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 1169–1172.

[4] V. Leon, M. Abdullah Hanif, G. Armeniakos, X. Jiao, M. Shafique, K. Pekmestzi, and D. Soudris, "Approximate computing survey, Part I: Terminology and software & hardware approximation techniques," 2023, *arXiv:2307.11124*.

[5] F. Baharvand and S. G. Miremadi, "LEXACT: Low energy N-modular redundancy using approximate computing for real-time multicore processors," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 431–441, Apr. 2020.

[6] J. Shalf, "The future of computing beyond Moores Law," *Philos. Trans. R, Soc. A*, vol. 378, no. 2166, 2020, Art. no. 20190061. [Online]. Available: https://scholar.googleusercontent.com/scholar.bib?q=info:sDfA_BfmqY8J:scholar.google.com/&output=citation&scisdr=ClEC6nSkEMyy8gCRBo0:AFWwaeYAAAAAZf6UHo1bdnD9XDY0jvfhI2uxVC0&scisig=AFWwaeYAAAAAZf6UHt0ndAdA_V9c7CyQN9gdJ68&scisf=4&ct=citation&cd=-1&hl=en

[7] A. K. Singh, S. Dey, K. McDonald-Maier, K. R. Basireddy, G. V. Merrett, and B. M. Al-Hashimi, "Dynamic energy and thermal management of multi-core mobile platforms: A survey," *IEEE Des. Test. IEEE Des. Test. Comput.*, vol. 37, no. 5, pp. 25–33, Oct. 2020.

[8] Q. Fettes, M. Clark, R. Bunescu, A. Karanth, and A. Louri, "Dynamic voltage and frequency scaling in NoCs with supervised and reinforcement learning techniques," *IEEE Trans. Comput.*, vol. 68, no. 3, pp. 375–389, Mar. 2019.

[9] S. Zhao, X. Dai, and I. Bate, "DAG scheduling and analysis on multi-core systems by modelling parallelism and dependency," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4019–4038, Dec. 2022.

[10] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *ACM SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 301–312, Apr. 2012.

[11] A. Chandrasekharan, D. Große, and R. Drechsler, "ProACt: A processor for high performance on-demand approximate computing," in *Proc. Great Lakes Symp. VLSI*, May 2017, pp. 463–466.

[12] I. Felzmann, J. F. Filho, and L. Wanner, "Risk-5: Controlled approximations for RISC-V," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4052–4063, Nov. 2020.

[13] A. Kanduri, M.-H. Haghbayan, A. M. Rahmani, P. Liljeberg, A. Jantsch, H. Tenhunen, and N. Dutt, "Accuracy-aware power management for many-core systems running error-resilient applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2749–2762, Oct. 2017.

[14] Z. Peng, X. Chen, C. Xu, N. Jing, X. Liang, C. Lu, and L. Jiang, "AXNet: ApproXimate computing using an end-to-end trainable neural network," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.

[15] B. Kocot, P. Czarnul, and J. Proficz, "Energy-aware scheduling for high-performance computing systems: A survey," *Energies*, vol. 16, no. 2, p. 890, Jan. 2023.

[16] R. Devaraj, A. Sarkar, and S. Biswas, "Supervisory control approach and its symbolic computation for power-aware RT scheduling," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 787–799, Feb. 2019.

[17] R. Devaraj and A. Sarkar, "Resource-optimal fault-tolerant scheduler design for task graphs using supervisory control," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7325–7337, Nov. 2021.

[18] N. Wu and Y. Xie, "A survey of machine learning for computer architecture and systems," *ACM Comput. Surveys*, vol. 55, no. 3, pp. 1–39, Feb. 2022.

[19] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2006, pp. 347–358.

[20] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 1521–1526.

[21] Z. Wang, Z. Tian, J. Xu, R. K. V. Maeda, H. Li, P. Yang, Z. Wang, L. H. K. Duong, Z. Wang, and X. Chen, "Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system," in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2017, pp. 684–689.

[22] X. Li, L. Chen, S. Chen, F. Jiang, C. Li, and J. Xu, "Power management for chiplet-based multicore systems using deep reinforcement learning," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2022, pp. 164–169.

[23] R. G. Kim, W. Choi, Z. Chen, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Imitation learning for dynamic VFI control in large-scale manycore systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 9, pp. 2458–2471, Sep. 2017.

[24] M. Rapp, H. Khdr, N. Krohmer, and J. Henkel, "NPU-accelerated imitation learning for thermal optimization of QoS-constrained heterogeneous multi-cores," *ACM Trans. Design Autom. Electron. Syst.*, vol. 29, no. 1, pp. 1–23, Jan. 2024.

[25] L. Chen, X. Li, F. Jiang, C. Li, and J. Xu, "Smart knowledge transfer-based runtime power management," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2023, pp. 1–6. [Online]. Available: https://scholar.googleusercontent.com/scholar.bib?q=info:CiXTdO6jOFYJ:scholar.google.com/&output=citation&scisdr=ClEC6nSkEMyy8gCWoIo:AFWwaeYAAAAAZf6TuIq0RCozVLs4cOY-Bv0fQRU&scisig=AFWwaeYAAAAAZf6TuLDoDGuDVS3hAevkd9d2u3Q&scisf=4&ct=citation&cd=-1&hl=en

[26] E. F. Morales and H. J. Escalante, "A brief introduction to supervised, unsupervised, and reinforcement learning," in *Biosignal Processing and Classification Using Computational Learning and Intelligence*. Elsevier, 2022, pp. 111–129. [Online]. Available: https://scholar.googleusercontent.com/scholar.bib?q=info:DqNpXsLO6jYJ:scholar.google.com/&output=citation&scisdr=ClEC6nSkEMyy8gCRvtU:AFWwaeYAAAAAZf6UptXzCDXcUHz78UpoWEvWXtU&scisig=AFWwaeYAAAAAZf6Upp8MUjU-8_NjJhnhhs1q79A&scisf=4&ct=citation&cd=-1&hl=en

[27] M.-A. Chadi and H. Mousannif, "Understanding reinforcement learning algorithms: The progress from basic Q-learning to proximal policy optimization," 2023, *arXiv:2304.00026*.

[28] L. Peng, W. Qing, and G. Yujia, "Study on comparison of discretization methods," in *Proc. Int. Conf. Artif. Intell. Comput. Intell.*, vol. 4, Nov. 2009, pp. 380–384.

[29] M. D. Atkinson, J.-R. Sack, N. Santoro, and T. Strothotte, "Min-max heaps and generalized priority queues," *Commun. ACM*, vol. 29, no. 10, pp. 996–1000, Oct. 1986.

[30] N. B. Mallya, C. Gonzalez-Alvarez, and T. E. Carlson, "Flexible timing simulation of RISC-V processors with sniper," *Simulation*, vol. 4, no. 1, 2018.

[31] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "AxBench: A multiplatform benchmark suite for approximate computing," *IEEE Des. Test. IEEE Des. Test. Comput.*, vol. 34, no. 2, pp. 60–68, Apr. 2017.

[32] A. K. Asanovic, R. Avizienis, J. Bachmeyer, and C. F. Batten, "The RISC-V instruction set manual v2.2," in *Proc. IEEE Int. Conf. Ind. Technol.*, Jun. 2012, pp. 1–32.

[33] R. Zhou and W. Qian, "A general sign bit error correction scheme for approximate adders," in *Proc. Int. Great Lakes Symp. VLSI (GLSVLSI)*, May 2016, pp. 221–226.

[34] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A dynamic range unbiased multiplier for approximate applications," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2015, pp. 418–425.

[35] S. Hashemi, R. I. Bahar, and S. Reda, "A low-power dynamic divider for approximate applications," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[36] S. Reda and M. Shafique, *Approximate Circuits*. Cham, Switzerland: Springer, 2019.

[37] NanGate, Inc., "NanGate 45nm Open Cell Library," 2008. [Online]. Available: http://www.nangate.com/?page id=2325 and https://mflowgen.readthedocs.io/en/latest/stdlib-freepdk45.html

SEYED ALI KASHANI GHARAVI received the B.S. degree in computer engineering from the Amirkabir University of Technology (AUT) and the M.S. degree from the University of Tehran, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering. He is a member of the Computer Architecture Laboratory, University of Tehran.

SAEED SAFARI received the Ph.D. degree in computer architecture from the Computer Engineering Department, Sharif University of Technology, Tehran, Iran, in 2005. Since then, he has been a Faculty Member with the Electrical and Computer Engineering Department, University of Tehran, Tehran. From May 2009 to September 2010, he collaborated with TeleRobotics and Applications (TERA) Laboratory, IIT, Genoa, Italy, working on different aspects of low-power parallel implementation of machine vision applications. His research interests include artificial intelligence, high performance computing, computer architecture, and computer arithmetic.