JJEE

Volume 9 | Number 3 | September 2023 | Pages 285-300

Jordan Journal of Electrical Engineering

ISSN (print): 2409-9600, ISSN (online): 2409-9619

Homepage: jjee.ttu.edu.jo

Energy-Efficient Cache Partitioning Using Machine Learning for Embedded Systems

Samar Nour[1*], Shahira M. Habashy[2], Sameh A. Salem[3]

[1, 2, 3] Department of Computer and Systems Engineering, Faculty of Engineering, Helwan University, Cairo, Egypt
[3] Egyptian Computer Emergency Readiness Team, National Telecom Regulatory Authority, Cairo, Egypt
E-mail: samar_nour@h-eng.helwan.edu.eg

Received: December 01, 2022 *Revised: February 6, 2023* *Accepted: February 10, 2023*

***Abstract*—** Nowadays, embedded device applications have become partially correlated and can share platform resources. Cross-execution and sharing resources can cause memory access conflicts, especially in the Last Level Cache (LLC). LLC is a promising candidate for improving system performance on multicore embedded systems. It leads to a reduction in the number of high-latency main memory accesses. Currently, commercial devices can use cache partitioning. The software could better utilize the LLC and conserve energy by caching. This paper proposes a new energy-optimization model for embedded multicore systems based on a reconfigurable artificial neural network LLC architecture. The proposed model uses a machine-learning approach to express the reconfiguration of LLC, and can predict each task's next interval LLC partitioning factor at runtime. The obtained experimental results reveal that the proposed model - compared to other algorithms - improves energy consumption by 28%, and gives 33% reduction in the LLC miss rate.

***Keywords*—** Energy optimization; Multicore embedded systems; Last level cache; Cache partitioning; Machine learning.

1. INTRODUCTION

As the volume and complexity of multicore embedded systems grow, the need for increased performance while consuming less energy has become a requirement. As a result, multicore architectures are being used to design many embedded systems. Furthermore, the academic community in embedded systems has already recognized the need for new implementation models that can account for such nonlinearities between the CPU and memory. Some attempts have been made to isolate memory delay from computing clock cycles by establishing workload models with two different execution cycles, namely, computation and memory cycles. Because the core frequency can affect the computing cycles, it should be split into the execution period factor, providing a more accurate execution model. Memory cycles are only affected by the memory's operating frequency and bus topology. As a result of using such models, researchers would assume that the core and memory subsystems are entirely responsible for a task's execution time, and adding the task's features may be characterized by the amount of compute cycles and memory cycles.

Moreover, several multicore embedded systems use energy-saving methods that neglect the effect that caches have on system energy use. Cache has been previously established to meet performance loads by connecting the speed hole between the CPU and main memory.

* Corresponding author

Article's DOI: 10.5455/jjee.204-1669909560

The cache takes up a significant amount of space in the cores and consumes a significant amount of energy. Many generic strategies, such as selective-way energy conservation, have been suggested to offset the energy lost due to caching. Although, the advances in cache architectures to reduce energy usage, applying similar strategies to multicore systems has received little attention. Even with considerable advances in heterogeneous multicores and energy-efficient deadline-constrained scheduling, previous work suffers from two major drawbacks [1-4]. First, it disregarded the effect of the shared cache within task runtime. Recurrent memory requests from processes running on separate cores can cause unexpected execution time delays; for example, an executing task's relevant cache lines may be evicted from the shared cache by a task operating on another core. Due to the nature of memory requirements, schedulable tasks may miss deadlines. Second, it is assumed that tasks scale proportionally with core frequency. The memory latency encountered during task execution is not captured by this simple assumption. While increasing the core frequency does not influence the number of calculation cycles, it does increase memory latency cycles, which depend on the internal properties of the executing process. The memory system architecture includes caches and their levels, the consistency model, cache coherence support, and intra-chip interconnects. These describe how the cores communicate together and allow the system to be incredibly efficient, totally programmable, and parallel. Cache levels can be defined as the configuration of caches that governs the size, associativity, and number of levels of caches that the system requires. The workload determines the quantity of cache needed for a specific architecture. Also, the number of cache levels is determined by the interconnection between the main memory and each processing element (core).

Memory management frees up memory after the task process is completed. Any system that can multitask at any given time must have efficient memory allocation. Also, it is influenced by the hardware, operating system, and active workloads [5]. Cache partitioning (CP) can be employed in one of two ways. The first is a static scheme in which the CP given to a core remains constant during the task set execution [6]. While the second is a dynamic scheme in which the CP assignments can be altered at execution time. Several cache partition algorithms are created to schedule LLC to overcome the problem of performance delay induced by such a coarse-sized partition. Some partitioning algorithms use a performance collector to produce an LLC allocation scheme based on detailed performance statistics.

For example, the work aims to partition LLC in the workload based on each task's cache miss. They necessitate real-time monitoring of each task's cache misses and the implementation of a fine-grained cache division [7]. Machine learning (ML) has been widely employed in recent years in computer vision, natural language processing, automation, and other fields to make perfectly all-right judgments with minimal overhead [8]. Moreover, there is a complicated internal link between architectural clues (memory bandwidth, LLC capacity) and task performance [9]. Computer architectural suggestions are only required once as input to the ML models to infer the best partition for capturing relationships. In other words, the best split is determined in practice by sampling architectural clues one time. As a result, using ML in the LLC partition reduces the profiling overhead.

We can summarize the significant contributions of this paper as follows:

1) We propose a new energy optimization using cache partitioning in multicore embedded systems based on ML.

2) The proposed model provides the optimum LLC partition factor for individual tasks. It leads to minimizing the latency of main memory accesses.
3) The proposed model achieves the highest load balance to allocate tasks using a BIN packing optimizer.
4) We guarantee the effectiveness of the proposed approach in multicore embedded systems by assessing the optimum energy efficiency on different benchmarks to examine its suitability and reliability.

The organization of this paper is as follows: section 2 presents a summary of the related work. Section 3 demonstrates the methodology of the system model. Section 4 describes the proposed model and its algorithms. Section 5 explores a performance evaluation through the platform, experimental setup, results, and discussion. Section 6 discusses the conclusions.

2. RELATED WORK

Systems have been using several strategies: way shutdown, way management, cache splitting, and scalability [10-12]. Sheikh et al. suggested energy optimization approaches that improve the baseline scenario for each core and cache level [13]. They presented levels of cache and added a fascinating dimension to multicore embedded systems' energy minimization, which become heterogeneous multicore architectures that gain prominence in embedded multicore systems. In [14], authors proposed a comprehensive technique to reduce energy usage on clustered heterogeneous multicore systems while accounting for previous work's unsophisticated assumptions. They build a heuristic scheduling method called Task-Heterogeneity-Energy Aware Mapping (THEAM). More research was done to determine how Dynamic Voltage and Frequency Scaling (DVFS) and cache-partitioning strategies affect memory delay cycles that work independently. Nejat et al. presented a new online resource management algorithm (RMA) [15]. It utilizes a coordinated scheduler for DVFS and last-level cache partitioning to determine the most effective resource setting at each task period to decrease two essential processor energy components, primarily core energy per instruction and DRAM shares. To reduce run-time overhead, it should use the RMA as the heuristic technique that performs a polynomial-time configuration space search. In [16-18], the team obtained a vulnerability-aware energy optimization method for multicore embedded systems. They combined private L1 cache dynamic cache reconfiguration (DCR) with shared L2 cache partitioning (CP). Under the constraints of performance and vulnerability, the L2 CP is decreasing inter-core interference successfully, while L1 DCR can further reduce energy consumption. The suggested technique uses dynamic programming to efficiently explore the space for optimal L1 cache configurations for each workload and L2 cache partition factors for each core. This research in [19] described a machine-learning-based LLC capacity allocation technique for cross-applications that aimed to improve overall throughput and fairness. The Support Vector Machine (SVM) model is employed in the proposed method to group applications into three classes so that each task within the same class has equivalent performance change patterns. Furthermore, the Bayesian Optimizer is used to construct the ideal LLC resource allocation schema based on application classes. High write latencies in NVRAM-based main memory architectures result in performance and energy overheads in machine learning and graph applications [20]. This research improves the performance of systems by smartly designing software data

structures with reuse distances that change as a task is done. Last-level cache partitions are dynamically adjusted to maintain market data on the chip, reducing write access to off-chip evictions. In [21], they employed reinforcement learning (RL) to guide and expedite their design to build a cost-effective cache replacement policy. They used features that are easy to access in the LLC to educate an RL agent. In [22], they have developed a non-intrusive, artificial neural network (ANN)-based runtime energy optimizer for multicore embedded systems that can cope with the stringent time requirements of essential tasks in this research. DVFS and task migrations are two of the energy optimizer's features. It is activated by the ANN output, which seeks to forecast the impact of frequency scaling on a task's performance based on its performance trace. Researchers in [23] proposed that the application program's behavior is described using its IPC trend as a function of varying resources. They try to classify applications using a statistical approach called the k-means algorithm. The workloads are classified as herbivores, carnivores, omnivores, or amphibians using the k-means machine-learning algorithm used in the studies. The expected outcomes are consistent with the characterization. Nour et al. in [24] provided a practical and dependable hybrid approach for reducing energy and makespan in multicore systems. To achieve optimal Voltage/Frequency (Vmin/F) values, the suggested hybrid model improves and merges the greedy approach with dynamic programming. Then, based on the available workloads, the allocation mechanism assigns a suitable core or island for each task.

3. METHODOLOGY

This paper uses a machine-learning approach to optimize energy efficiency for multicore embedded systems. In contrast, ML determines the optimal partition factor for each task to the Last Level Cache (Shared Cache L2). The suggested ML model is automatically trained to predict objectives based on the data obtained from the input features. This model improved predictions based on the data gathered from the proposed platform and enhanced the decisions taken under crucial time constraints. Therefore, it is suitable and most appropriate for this model to use supervised learning approaches. It leads to a decrease in the error between the prediction model and the objective. The optimum goal is estimated using the normalizing features.

3.1. Multicore Architecture Model

Fig. 1 shows a typical multicore system with a shared on-chip L2 cache and one private L1 cache for each core. The shared L2 cache includes way-based partitioning. Also, the private L1 caches are separated into data and instruction caches (IL1 and DL1). As in [25], a shared L2 cache has way-based partitioning. Each L2 cache set (8-way associativity) is divided into four sections. Each section is assigned to a single core, as shown in Fig. 1. Each core will only access the cache sets allocated to it and implement the "least recently used" (LRU) policy for a replacement among its group of ways. The number of ways assigned to a core is called the cache partition factor. Fig. 1 shows that Core 0 has an L2 partition factor of 3. In this paper, static partitioning is used for the global L2 cache. In another sense, L2 partition factors are certified for each core. Also, it remains consistent during runtime. All workloads processed on that core have almost the same L2 partition factor.

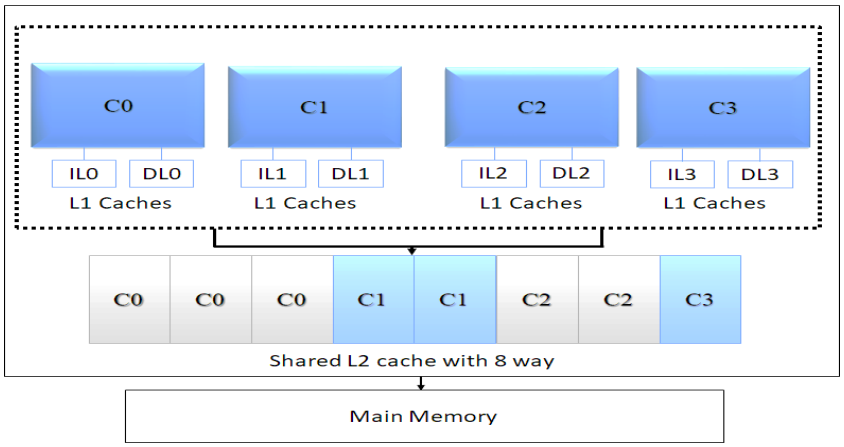


Fig. 1. Multicore architecture model including both the L1 private caches and the partition shared L2 cache.

3.2. Energy Model

The proposed energy model is described in [26]. As in Eq. (1), the cache energy usage combines static and dynamic energy hierarchy models proposed for shared L2 and private L1 (instruction and data caches). Static and dynamic energy is taken into account, as demonstrated in Eqs. (2) to (7):

$$E = Esta + Edyn \quad (1)$$

$$Edyn = (cache\ hits) * Ehit + (cache\ misses) * Emiss \quad (2)$$

$$Emiss = E(off\ chip\ access) + (miss\ cycles) * E(CPU\ stall) + E(cache\ fill) \quad (3)$$

$$Miss - Cycles = (cache\ misses) * (miss\ latency) + ((cache\ misses) * (line\ size/16)) * (memory - band - width) \quad (4)$$

$$Esta = (total\ cycle) * E(static\ per\ cucle) \quad (5)$$

$$E(static\ per\ cycle) = (cache\ size\ in\ Kbyte) * E(per\ kKbyte) \quad (6)$$

$$E(per - Kbyte) = (E(dyn\ of\ base\ cache) * .10)/\ base\ cache\ size\ in\ Kbyte \quad (7)$$

The proposed work is built on the Gem5 simulator [27]. The Gem5 simulator is used to figure out the dynamic energy, and the static power is considered 10% of the dynamic energy. The CPU consumed around 30% of the system's total energy. There are no links between instruction and data cache operations. L1 caches are studied locally with instruction, but the data caches are analyzed separately. In addition, the cache accesses hits and misses for each task set extracted using Gem5. Gem5 also investigates the shared cache L2. We got off-chip access energy from a standard low-power memory. Consequently, the fetch from the main memory took forty times as long as a retrieve from the level 2 cache. Also, memory bandwidth is measured [25].

3.3. Problem Formulation

The suggested model is a multicore embedded system with four rules that turn the exploration into a linear program as follows:

- The model consists of n cores. The set of all cores is symbolized by C: c1, c2, . . . , cn .
- Each core consists of private L1 (Instructions and Data) caches.
- The way-based partitioning factor is k-way associative with the shared cache L2, shared by all n cores.

- The m-independent tasks T: t1, t2 …, tm are performed on the proposed model. Each task in a given mix of jobs may conclude at a different time, but there is a standard soft deadline by which all tasks should be finished.

Our model aims to discover the best partition factor for the L2 cache so that the assigned tasks consume the least energy. Machine learning generates all possible L2 cache partitioning schemes W: w1, w2 …, wk and assigns wk ways to them. The minimization problem can be defined as follows:

$$Minimize \sum_{i=0}^{n}(Ei) \quad (8)$$

$$Maximize \sum_{i=0}^{n}(Timei) <= D \quad (9)$$

$$W = \sum_{i=0}^{n} Wi \epsilon [1, k] \quad (10)$$

Because such tasks do not exceed their deadlines (Timei), the total energy usage for all tasks (Ei) is reduced. Eqs. (8) and (9) ensure that all tasks are completed before the deadline (D). Eq. (10) confirms that the partitioning strategy is correct. Therefore, the overall partitioning factor is similar to shared L2 cache ways.

4. THE PROPOSED MODEL

4.1. Linear Regression using ANN

The shallow neural network approach allocates the sum of the input neurons, the few neurons at the hidden layer, and the output layer. The proposed model uses a supervised ANN to complete linear regression. Consequently, the ANN topology has three layers: an input layer, a hidden layer, and an output layer. The input neurons represent the received feature parameters (i.e., three features). While the output neurons represent the predicted partition factor. The hidden layer uses the sigmoid function to achieve the intended result effectively.

On the other hand, the output layer is used in the step of the unit function to become a linear output. Fig. 2 shows the suggested ANN topology. This design has the potential to deliver a high-impact performance. Initially, offline training of the ANN model is required. Then, we can modify the ANN hidden layer to establish an initial configuration for the hidden layer. We used pre-collected data. Therefore, we avoid cold start difficulties. It improves results using a simpler topology and a slight learning rate. In addition, many configurations of the ANN hidden layer are added. Therefore, the proposed model can quickly overfit the training data set [22, 28-30].

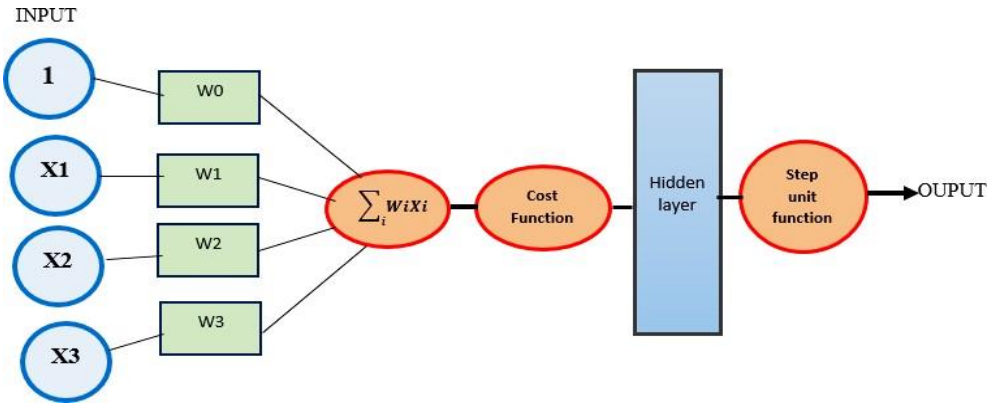


Fig. 2. The proposed ANN topology.

4.2. Predict Cache Partition Factor using ML

Fig. 3 shows an overview of the proposed optimization energy system model with ANN. Additionally, Algorithm 1 explains the implementation. In the beginning, some workloads were run on simulation, and the Gem5 in AIP was used for this. After that, we fill up the entries on the tables with simulated tasks in different configurations. Then, the models are trained several times using the collected data, and the model with the minimum error is selected. If the error is below or equal to a certain threshold, this model predicts the rest of the partition factor for LLC for the next workload. If not, more training data are collected by performing additional simulations and repeating the operation until the error threshold condition is met.

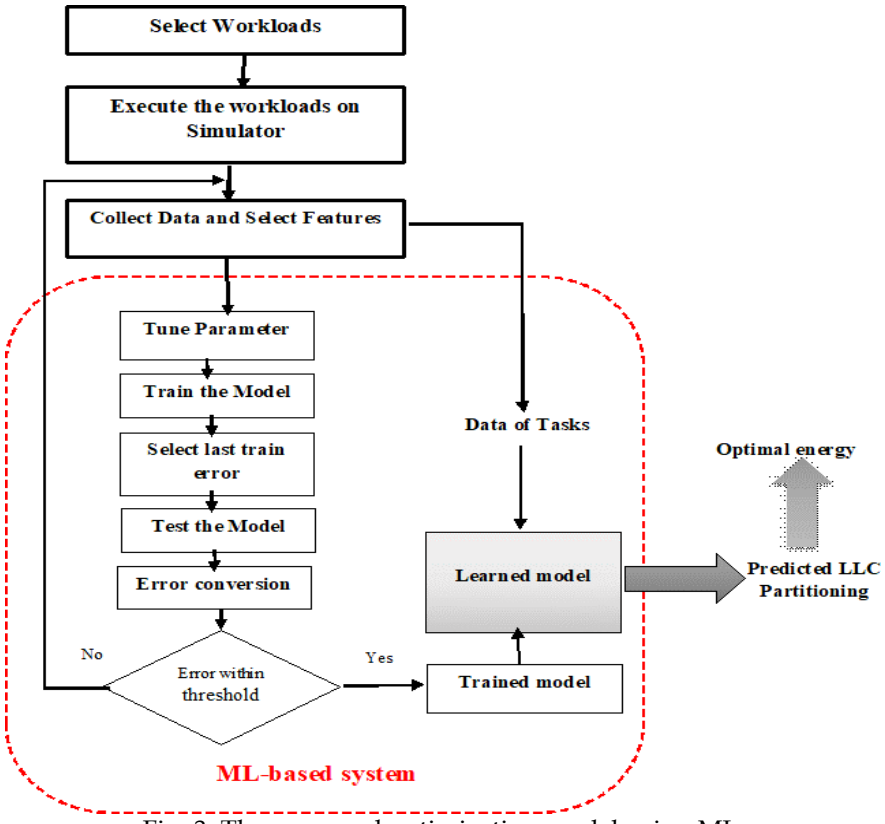


Fig. 3. The proposed optimization model using ML.

4.2.1. Data Collection and Feature Processing

Feature Selection: The most crucial component of a machine learning challenge is choosing appropriate features. The predicting partition factor and LLC cache entries with different cache configurations are selected as the features. Let X = <X0, X1, X2, X3> be the feature vector where:

X0→1 (Bias).

X1→ refers to the type of task if memory or CPU task, where memory task X1=1, CPU task X1=0.

X2 →refers to the utilization of task, where the range of X2 is $0 < X2 < 1$.

X3→ refers to the miss rate of L1 Cache, where the range of X3 is $0 < X3 < 1$.

Increasing the number of features makes the learning model more sensitive to over-fitting. When there are more features, the over-fitting issue can be resolved by using more training data sets. In the proposed scenario, another option is to reduce the training data.

Trained Model: The data are prepared with the filled table entries X and their LLC cache partition factor. The proposed model is divided into training, cross-validation, and testing. The training group is used to train the model. A cross-validation group is utilized for hyper-parameter tunings such as learning rate or weight adaptation. The test group is used to determine the model with the slightest inaccuracy. The experiment uses a split of 80% for training, 10% for cross-validation, and 10% for testing. Finally, after training models with multiple hyper-parameters, the model with the lowest Mean Square Error (MSE) on the test set is chosen, as illustrated in Algorithm 1.

Algorithm 1. Predict cache partition factor based machine learning

(Trained Model).
Input Layer: Queue of tasks (T 1 .....Ti)
1: X0=1 (Bias)
2: X1(Memory Task or CPU Task) where Memory Task=1 and CPU Task=0.
3: X2 (Utilization of Task) where 0 < X2 < 1
4: X3 (Miss rate of L1 Cache) where 0 < X3 < 1
Output Layer: Cpi = Predict Optimum Cache Partition Factor per task.
5: Input layer data and the actual value of observation were generated from simulation.
6: for ti = t1 to Ti do
7: Wi= Random Weights Parameters (W0, W1, W2, W3)
8: Artificial Neural Network (ANN)
9: while the actual output ≃ Prediced ouput do
10: Calculate SOP= Σ3 (WXi)
11: Calculate Cost Function= the Sigmoid Function
12: One Hidden Layer
13: Linear Regression
14: Calculate Step unit Function
15: Calculate Error =MSE
16: if (Error > threshold) then
17: Select Learning Rate η
18: 0 < η < 1
19: Calculate Weights Adaptation
20: else
21: Cpi = simulated output
22: break
23: end if
24: end while
25: end for

4.3. Allocation Tasks using Bin Packing

The bin-packing problem is an NP-hard problem in which various tasks must be assigned to the smallest number of accessible bins (cache partition). The task allocation between cores is done using one of the bin-packing strategies that have resulted in the development of numerous heuristics in this field. After applying the energy-efficient selector

approach described in Algorithm 1, each task's LLC partition factor (Cpi) is determined, and the job is assigned to a single core. As a result of Algorithm 1, the sum of the partition factor to all the tasks may have the same total cache partition for multicore systems total cache partition (Tp) or greater than or less than it. So, bin-packing techniques are used to resolve this problem.

Additionally, this paper proposes a new best-fit (BF) partitioning technique for multicore embedded systems. The BF gets the optimal allocation to all tasks to save more energy. Algorithm 2 is divided into three phases; the first phase splits duties into three groups. On the other hand, the second phase tries to get the optimally allocated jobs for each core. Finally, the third phase runs all tasks according to the Cpi and BF allocated.

Algorithm 2. Allocation tasks using bin packing.

```
Input: Output from Algorithm 1 (Predict Optimum Cache Partition Factor per task)(Cpi)
Tp=Total Cache Partition for Multi-core Systems Model SF= Split Factor
G1, G2 and G3 (task groups) Output: Best fit for all tasks per Cores.
SF=Tp/2
First Phase:
for cpi = cp1 to Cpi do
    if (cpi > SF ) then
        assign cpi to G1
    else if (cpi < SF) then
        assign cpi to G2
    else
        assign cpi to G3
    end if
end for
Second Phase:
    Sorting descending to three groups
for (i = 1 to T G1) do
    for (j = 1 to T G2) do
        if (Cpi + Cpj <= Tp) then
            map Ti and Tj to empty cores
        else if (cpi <= Tp) then
            map Ti to empty cores
        else
            map Tj to empty cores
        end if
        Update Tp
        Tp=Tp-((Cpi + Cpj)
    end for
end for
for (k = 1 to T G3) do
    if (Cpk + Cpk+1 <= Tp) then
        map Tk and Tk+1 to empty cores
    else
        map Tk to empty cores
    end if
    Update Tp
    Tp=Tp-(Cpk + Cpk+1 )
end for
Third Phase:
The tasks are running according to their partition factor Cpi per task and allocated best
fit (Bin Packing)
```

The first phase splits the tasks into three groups, where group 1 is called G1, group 2 is called G2, and group 3 is called G3. The "split factor" (SF) determines how to assign tasks to a specific group. The value of SF is equal to the total cache partition for the system divided into two. The first phase starts when checking if the Cpi of each task individually is greater than SF, at which point this task is assigned to G1. If the Cpi is smaller than SF, the task is delegated to G2. However, if the Cpi is the same as SF, this task is delegated to G3 (as shown in lines 3 to 13). When the first phase is finished filling out the three groups, the second phase will start. The first step of the second phase is sorting descending order into three groups (as in line 15). The next stage of this phase is taking the first element of G1. After that, this element is added to each component in G2 to achieve the condition. The condition is that the sum of the cache partition factor Cpi in G1 and the cache partition factor Cpj in G2 is less than or equal to Tp (as in lines 17 to 19). The phase will map each Cpi (Ti) and Cpj (Tj) to empty cores if the condition is actual. If it is not obtained, the stage will check that the Cpi is less than or equal to Tp; if accurate, map Ti to empty cores; otherwise, map Tj to empty cores (as in lines 20 to 24). Subsequently, the phase updates Tp with a new value (as in lines 25 to 26). The stage will repeat all the above steps for all elements in G1 and G2. The final step of the second phase maps every two elements in G3 to empty cores if the summing of the partition factor Cpk of the first element in this group and the Cpk+1 of the next part is less than or equal to Tp. The phase will map each of Cpk and Cpk+1 to empty cores; otherwise, it will map Cpk to empty cores (as in lines 28 to 34). Subsequently, the phase updates Tp with a new value (as in lines 35 to 36). The third phase starts after the second phase, which is the end phase in Algorithm 2. In this phase, the tasks run according to their partition factor Cpi and are allocated according to "best fit" bin packing (as in line 38).

5. PERFORMANCE EVALUATION

The proposed heuristic model is tested on a different set of experiments for validation. Many random experiments with real-time workloads have been done. This section presents the platform, how to set up the investigation, the results, and the discussion. For comparison, the proposed model is compared with three competitive algorithms: CRUML [17], OMLFEA [22], and THEAM [14], using the same set of experiments.

5.1. Platform

The performance of the proposed model is assessed on a four-core Cortex-A53 power platform. In addition, it has a 32 kB instruction cache and a 32 kB data cache as a private L1 cache (IL1, DL1). The single cluster can share the shared L2 cache that supports 512 kB. As shown in Table 1, an 8-way associative with 32-byte lines can operate in the same dynamic voltage frequency scaling (DVFS) domain. In our experiments, it used 25 benchmarks from three separate benchmark sets, Parallel Suites Benchmark: NAS V3.3.1 (NPB) [31], SPEC suite CPU2006 [32], and PARSEC suite v3.0 [33]. Also, online parallel benchmark suites are available. The power data is tracked for each trial, such as energy, full use, and benchmark data about operations per second. Each experiment was performed ten times, and the average value was calculated.

Table 1. Architecture model parameters.

Parameter type	Parameter	Parameter value/ description
Cores Parameters	Number of cores	4 (One Cluster)
	Core Frequency	2.4 GHz
	Instruction set architecture	X86
Memory Parameters	L1 Cache	Private and separate instruction and data cache, each 32 kB in size
	L2 Cache	Reconfigurable, shared and 512 kB in size
	Cache coherence	MESI two-level directory based cache coherence protocol
	Memory size	4 GB DDR

5.2. Experimental Setup

The Gem5 simulator [27] is considered a promising candidate for the design tools used by computer architecture researchers. Researchers can use this simulation infrastructure to simulate modern computer hardware on a cycle-by-cycle level. Also, they can run large tasks for different architectures, such as X86, Arm, and RISC-V, on top of Linux operating systems. The proposed work used the Gem5 simulator to simulate the multicore system, as shown in Fig. 1. The simulator has been enhanced to provide way-based partitioning of the shared L2 cache. The statistics and machine learning Python code are used to train the proposed machine learning model. The energy consumption of the task described in Table 1 was evaluated on the same platform using a Linux power governor to provide a baseline for assessing the suggested energy optimizer (using Raspberry Pi OS with Kernel version 5.4). Conservative power-governor strategies and CPUFreq OnDemand were chosen in our model.

5.3. Results

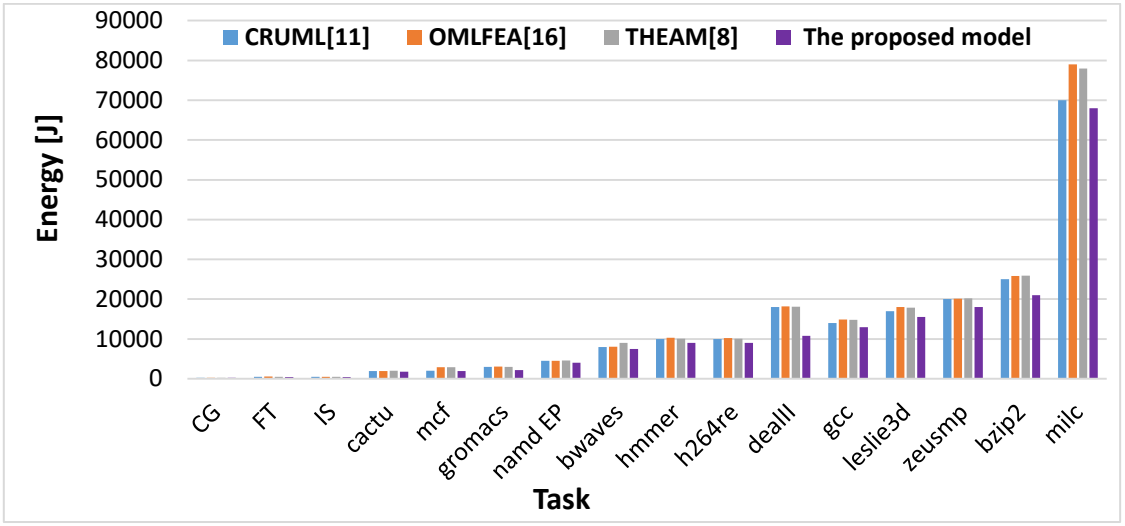
The experimental parameters employed are listed in the preceding section. As a result, for each core form, we create an energy model. Also, we measure the energy efficiency of each core and the total amount of energy used at the same frequencies by dividing the benchmarks of multicores within all LLC partitioning configurations.

5.3.1. Energy Efficiency Results

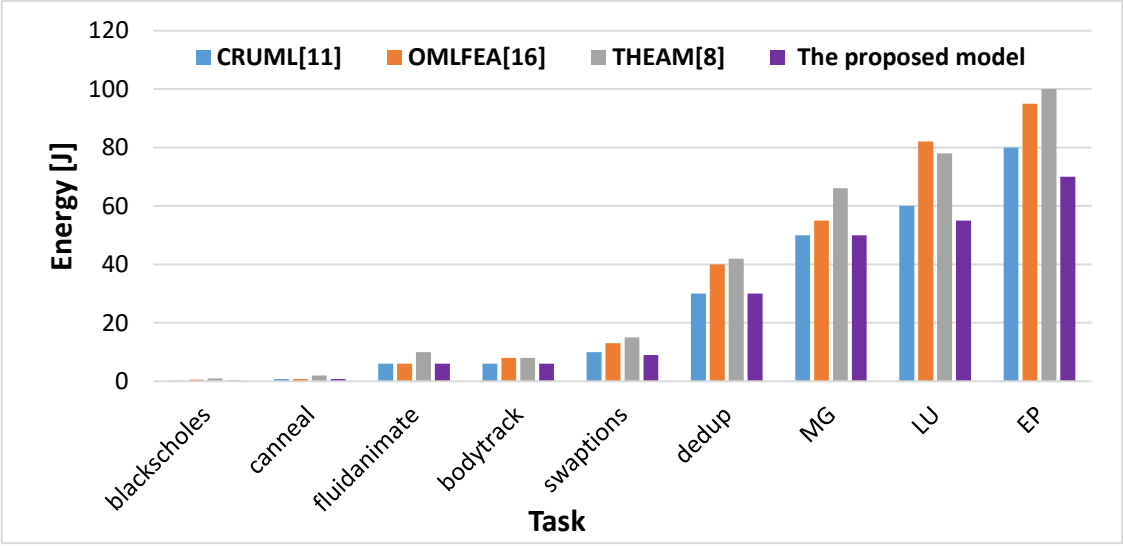
Fig. 4 shows the energy efficiency of the cores for our proposed model across three other competitive models, CRUML [17], OMLFEA [22], and THEAM [14]. We used the cache partitioning technique in all those models while running each benchmark. As demonstrated, when the task has a variable rate that it takes to finish, we can expect different levels of efficiency for different rules. As shown in Fig. 4, the proposed model has improved energy consumption compared with other models, and the efficiency variations range from approximately 18% to 34%. As a result, cache partitioning reduces energy consumption, especially for tasks that require a long time to execute, such as sequential tasks like milc and mcf, as shown in Fig. 4(a).

Against parallel operations, energy consumption decreases or does not grow when the system's utilization increases. Also, when it has a long execution time with cache partitionings such as MG and dedup, as shown in Fig. 4(b).

In short, the proposed model can increase system use in the case of one application rather than others. So, across all benchmarks, we use the proposed architecture with cache partitioning to reduce energy consumption. Fig. 4 shows that our proposed method effectively reduces energy use for most usage benchmarks.



(a)



(b)

Fig. 4. Energy consumption: a) sequential tasks; b) parallel tasks.

5.3.2. L2 Cache Miss Rate

The properties of the benchmarks that we utilize are depicted in Fig. 5. The number of L2 cache misses per 1000 core cycles for various L2 cache sizes is shown in the graph. Fig. 5 demonstrates the rate of L2 misses using the suggested methodology and other comparable models, where the partitioning cache improves by 41% in some workloads. The results showed a value gap between the presented model and the different techniques of 18% to 32%. The proposed model has the highest success rate. It is also worth noting that the CRUML [17] technique does not work better because it ignores the L1 miss rate.

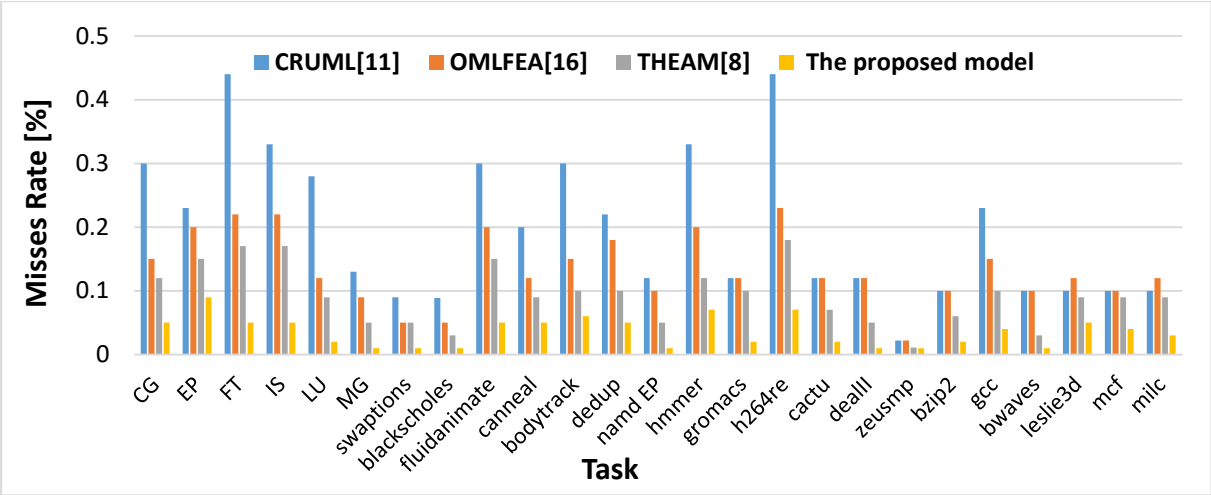


Fig. 5. L2 cache misses.

5.4. Discussion

This section introduces the performance enhancements of the proposed model. The proposed model was compared with CCRUML [17], OMLFEA [22], and THEAM [14] in various benchmarks. On average, the proposed model achieves a 22% to 32% reduction in energy consumption compared with the competitive methods. LLC and the cache partition factor are considered the most critical parts of the proposed model. The proposed LLC configuration can save energy without affecting the accuracy of task timing or outperforming the deadline strategy. The evaluated tasks are given different formats for the LLC. These configurations provide a multicore embedded system's adaptive capability. As a result, the proposed model obtains the optimal LLC configurations for each task.

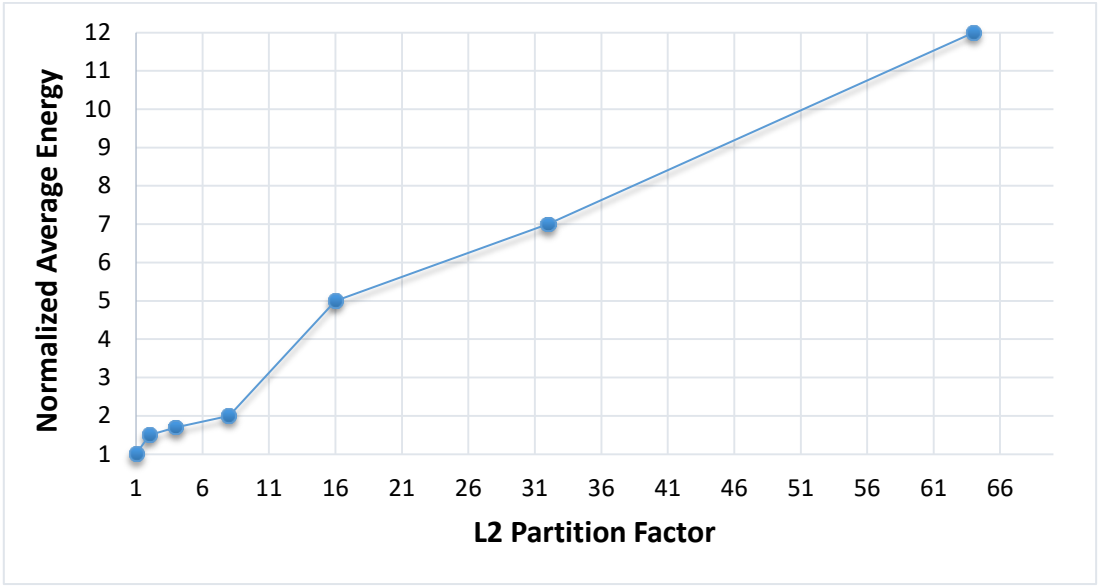
The proposed ANN models provide the generic tasks' starting data during offline training. The suggested model is expected to adapt ANN through learning in the first cache partition factor reductions. Consequently, the proposed model successfully predicts the tested tasks and can achieve the best cache partition factor configuration without compromising fundamental task timing accuracy.

Table 2 summarizes the overall performance measures for the proposed ANN models against the other three competitive models across several benchmarks. The results show that the suggested model outperforms all performance measures compared with other competitors. In addition, the proposed method achieves the best LLC configuration levels for handling different workloads on different systems. Moreover, the proposed model optimizes the number of cache partition factors and allows the cores to be more deterministic by using BIN packing. As shown in Table 2, the proposed model achieves the optimal goal through the minimum error during training.

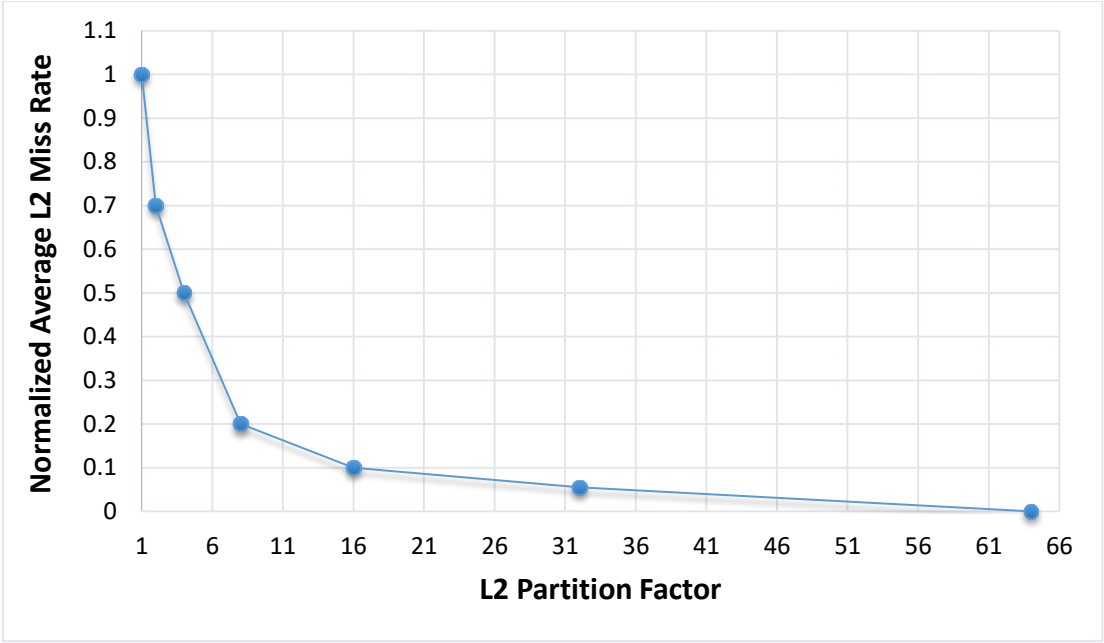
Table 2. Overall performance evaluation.

Algorithms	Average Energy [J] ± Std. dev. [%]	Average L2 Misses Rate ± Std. dev. [%]	ML Error [%]
CRUML [17]	1348 ± 13.91	2.696 ± 0.027	0.30
OMLFEA [22]	1338 ± 14.81	2.776 ± 0.028	0.48
THEAM [14]	1607 ± 16.35	3.123± 0.032	.52
The Proposed Model	1128 ± 11.51	2.256 ± 0.023	0.20

Additionally, Fig. 6 explains the effect of the LLC partition factor on both the energy and L2 miss rates. It guarantees that increasing the LLC partition factor increases the power, as shown in Fig. 6(a); however; it reduces the L2 miss rate, as shown in Fig. 6(b).



(a)



(b)

Fig. 6. LLC partition factor: a) normalized average energy; b) normalized average L2 miss rate.

6. CONCLUSIONS

This paper introduced a holistic model to minimize energy consumption on multicore embedded systems. The proposed model suggested a new ML-based LLC resource-partitioning strategy. As demonstrated, the proposed model involved learning the multicore embedded system. In addition, it can choose the best partition LLC for each task, minimizing memory latency and using the least amount of energy.

Furthermore, the introduced model used a pin-packing optimizer to produce the best task allocation schema based on the LLC partitioning factor. For validation, the proposed engine was tested in different applications to examine its suitability and reliability. Results showed average and maximum energy savings of 22% to 32% individually for each core. In addition, it gave 18% to 34% for system-level energy consumption. At the same time, it could achieve an average 33% reduction in the L2 miss rate.

REFERENCES

[1] D. Liu, J. Spasic, P. Wang, T. Stefanov, "Energy-efficient scheduling of real-time tasks on heterogeneous multicores using task splitting," *IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 149– 158, 2016.

[2] M. Thammawichai, E. Kerrigan, "Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors," *Real-Time Systems*, vol. 54, no. 1, pp. 132–165, 2018.

[3] A. Suyyagh, Z. Zilic, "Energy and task-aware partitioning on singleisa clustered heterogeneous processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 2, pp. 306–317, 2019.

[4] S. Pagani, A. Pathania, M. Shafique, J. Chen, J. Henkel, "Energy efficiency for clustered heterogeneous multicores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1315–1330, 2016.

[5] S. Patil, S. Scholten, M. Tao, H. Al-Asaad, "Survey of memory, timing, and power management verification methods for multi-core pro- cessors," *in 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference*, pp. 0110–0119, 2019.

[6] S. Nour, S. Salem, S. Habashy, "Energy optimization by using last level cache partitioning for multicore platforms," *in 2021 16th International Conference on Computer Engineering and Systems*, pp. 1-6, 2021.

[7] H. Lee, S. Cho, B. Childers, "Cloudcache: expanding and shrinking private caches," *in 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 219–230, 2011.

[8] W. Wu, N. Chen, C. Zhou, M. Li, X. Shen, W. Zhuang, X. Li, "Dynamic ran slicing for service-oriented vehicular networks via constrained learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2076-2089, 2020.

[9] L. Liu, "Qos-aware machine learning-based multiple resources scheduling for microservices in cloud environment," *arXiv preprint arXiv:1911.13208*, 2019.

[10] A. Malik, B. Moyer, D. Cermak, "A low power unified cache architecture providing power and performance flexibility," *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pp. 241–243, 2000.

[11] M. Modarressi, S. Hessabi, M. Goudarzi, "A reconfigurable cache architecture for object-oriented embedded systems," *in 2006 Canadian Conference on Electrical and Computer Engineering*, pp. 959–962, 2006.

[12] C. Zhang, F. Vahid, W. Najjar, "A highly configurable cache for low energy embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 2, pp. 363–387, 2005.

[13] S. Sheikh, M. Pasha, "Energy-efficient real-time scheduling on multicores: a novel approach to model cache contention," *ACM Transactions on Embedded Computing Systems*, vol. 19, no. 4, pp. 1–25, 2020.

[14] S. Sheikh, M. Pasha, "Energy-efficient cache-aware scheduling on heterogeneous multicore systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 206-21, 2021.

[15] M. Nejat, M. Manivannan, M. Peric'as, P. Stenstrom, "Coordinated management of dvfs and cache partitioning under qos constraints to save energy in multi-core systems," *Journal of Parallel and Distributed Computing*, vol. 144, pp. 246–259, 2020.

[16] Y. Huang, P. Mishra, "Vulnerability-aware energy optimization using reconfigurable caches in multicore systems," *in 2017 IEEE International Conference on Computer Design*, pp. 241–248, 2017.

[17] A. Ahmed, Y. Huang, P. Mishra, "Cache reconfiguration using machine learning for vulnerability-aware energy optimization," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 2, pp. 1–24, 2019.

[18] S. Charles, A. Ahmed, U. Ogras, P. Mishra, "Efficient cache recon- figuration using machine learning in noc-based many-core cmps," *ACM Transactions on Design Automation of Electronic Systems*, vol. 24, no. 6, pp. 1–23, 2019.

[19] J. Qiu, Z. Hua, L. Liu, M. Cao, D. Chen, "Machine-learning-based cache partition method in cloud environment," *Peer-to-Peer Networking and Applications*, pp. 1–14, 2021.

[20] M. Ahmad, H. Dogan, F. Checconi, X. Que, D. Buono, O. Khan, "Software-hardware managed last-level cache allocation scheme for largescale nvram-based multicores executing parallel data analytics applications," *in 2018 IEEE International Parallel and Distributed Processing Symposium*, pp. 316–325, 2018.

[21] S. Sethumurugan, J. Yin, J. Sartori, "Designing a cost-effective cache replacement policy using machine learning," *in 2021 IEEE International Symposium on High-Performance Computer Architecture*, pp. 291–303, 2021.

[22] J. Conradihoffmann, A. Frohlich, "Online machine learning for energy-aware multicore real-time embedded systems," *IEEE Transactions on Computers*, vol. 71, no. 2, pp. 493-505, 2021.

[23] P. Jain, S. Surve, "Resource centric characterization and classification of applications using kmeans for multicores," *in 2019 International Conference on Information Networking*, pp. 25–30, 2019.

[24] S. Nour, S. Salem, S. Habashy, "A hybrid model for reliability aware and energy-efficiency in multicore systems," *Computers, Materials and Continua*, vol. 70, no. 3, pp. 4447–4466, 2022.

[25] A. Gordon-Ross, P. Viana, F. Vahid, W. Najjar, E. Barros, "A oneshot configurable-cache tuner for improved energy and performance," *in 2007 Design, Automation and Test in Europe Conference and Exhibition*, pp. 1–6, 2007.

[26] M. Alsafrjalani, A. Gordon-Ross, "Scheduling and tuning for low energy in heterogeneous and configurable multicore systems," *Computers*, vol. 7, no. 2, pp. 25, 2018.

[27] A. Semakin, "Simulation of a multi-core computer system in the gem5 simulator," *in AIP Conference Proceedings,* vol. 2318, no. 1, pp. 090006, 2021.

[28] A. Saeed, D. Mueller-Gritschneder, F. Rehm, A. Hamann, D. Ziegenbein, U. Schlichtmann, A. Gerstlauer, "Learning based memory interference prediction for co-running applications on multi-cores," *in 2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD*, pp. 1–6, 2021.

[29] A. Sezavar, H. Farsi, S. Mohamadzadeh, "Multi-depth deep similarity learning for person re-identification," *Jordan Journal of Electrical Engineering*, vol. 8, no. 3, pp. 279-287, 2022.

[30] Z. Dorrani, H. Farsi, S. Mohamadzadeh, "Deep learning in vehicle detection using ResUNet-a architecture," *Jordan Journal of Electrical Engineering*, vol. 8, no. 2, pp. 165-178, 2022.

[31] NASA Advanced Supercomputing Division, Nas Parallel Benchmarks Suite, v3.3.1. <https://www.nas.nasa.gov/software/npb.html.>

[32] J. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[33] C. Bienia, S. Kumar, J. Singh, K. Li, "The parsec benchmark suite: characterization and architectural implications," *in Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp. 72–81, 2008.