# COCKTAIL: Multi-Core Co-Optimization Framework With Proactive Reliability Management

Darong Huang, *Student Member, IEEE*, Ali Pahlevan, Marina Zapater, *Member, IEEE,*
David Atienza, *Fellow, IEEE*

***Abstract*—High performance computing (HPC) servers aim to meet an increase in the number and complexity of tasks and, consequently, to address the energy efficiency challenge. In addition to energy efficiency, it is essential to manage lifetime limitations of power-hungry components of servers (e.g., cores and cache), hence avoiding server failure before its lifetime period. Traditional approaches focus on either using hybrid caches to reduce the leakage power of traditional static random-access memory (SRAM) cache, and thus increase the energy efficiency, or the trade-off between the lifetime and performance of multi-core processors. However, these approaches fall short in terms of flexibility and applicability for HPC tasks in terms of multi-parametric optimization including quality-of-service (QoS), lifetime reliability, and energy efficiency. As a result, in this paper we propose COCKTAIL, a holistic strategy framework to jointly optimize the energy efficiency of multi-core server processors and tasks performance in the HPC context, while guaranteeing the lifetime reliability. First, we analyze the best cache technology among traditional SRAM and resistive random access memory (RRAM), within the context of hybrid cache architectures, to improve the energy efficiency and manage cache endurance limits with respect to tasks requirements. Second, we introduce a novel efficient proactive queue optimization policy to reorder HPC tasks for execution considering their end time and possible reliability effects on the use of the hybrid caches. Third, we present a dynamic model predictive control (MPC)-based reliability management method to maximize task performance, by controlling the frequency, temperature, and target lifetime of the server processor. Our results demonstrate that, while consuming similar energy, COCKTAIL provides up to 60% QoS improvement when compared to latest state-of-the-art energy optimization and reliability management techniques in the HPC context. Moreover, our strategy guarantees a design lifetime longer than 5 years for the whole HPC system.**

***Index Terms*—HPC servers, hybrid cache, reliability management, endurance-aware cache selection, proactive queue optimization, dynamic MPC.**

## I. Introduction

HIGH-performance multi-core processors have been massively deployed in recent years due to a rapid growth in the number and complexity of high performance computing (HPC) tasks. Consequently, there has been an increase in the number of HPC servers, ramping up the energy consumption [1]. In this context, an approximate power breakdown shows that multi-core processors contribute to 40% of the overall power of a HPC server [2]. Then, among processor components, the increase of the last level cache (LLC) size for better performance represents over 44% of the total power of a processor [3], [4]. A static random-access memory (SRAM)-based LLC has larger capacity and area [5] than L1 and L2 cache hierarchy. This causes the traditional SRAM LLC to suffer from a high leakage power, encompassing over 43% of the total leakage power, while the L1 and L2 caches only consist of 11% and 12% of the total leakage power [6], respectively. Therefore, the LLC and core components are becoming the key factors to consider for energy and lifetime reliability (processor aging) management in HPC infrastructures.

To reduce the overall energy consumption of multi-core processors, non-volatile memory (NVM) is a promising technology rather than traditional SRAM for LLC due to its near-zero leakage power [7]. Indeed, few different types of NVM technology are available, such as, resistive random access memory (RRAM), spin-transfer-torque magnetic random access memory (STT-MRAM), and phase-change random access memory (PCRAM). Although these NVM options combine the benefits of SRAM (i.e., speed) with more data storage density and less leakage power, they can only sustain few days of operation in intensive HPC tasks due to its limited number of temperature-dependent endurance [7]. Hence, a promising approach to address the shortcomings of both SRAM and NVM-based LLC is to exploit hybrid cache technologies (joint SRAM and NVM) at the architecture level [7], [8]. Unfortunately, state-of-the-art techniques mainly target NVM reliability improvements by considering different hardware design alternatives and miss the exploitation of software-level techniques to compensate at run-time the dynamic effects of different types of HPC workloads.

Recently, several studies have started to evaluate underlying reliability issues at the hardware level, such as, electromigration (EM), stress migration (SM), time-dependent dielectric breakdown (TTDB), thermal cycling (TC) and negative bias temperature instability (NBTI), in the context of workloads executed on multi-core processors [9], [10], and consequently propose new reliability-aware methods to control the operating temperature, which has a significant effect on overall reliability [11]. However, although the use of thermal management schemes [12] can indirectly help reliability, its use and the related studies do not attempt a joint optimization of system reliability driving the multi-core HPC architectures and considering tasks quality-of-service (QoS) (i.e., execution time), particularly in a dynamic scenario.

To attain the highest QoS and energy savings together with enhancing system reliability, there is a need to combine hybrid cache management with task-awareness management through dynamic voltage and frequency scaling (DVFS) for complete multi-core HPC platform. However, previous techniques tackle these objectives separately, either boosting

Darong Huang, Ali Pahlevan, Marina Zapater, and David Atienza are with the Embedded Systems Laboratory (ESL), EPFL, Switzerland. E-mail: {darong.huang, ali.pahlevan, marina.zapater, david.atienza}@epfl.ch.

Marina Zapater is also with the School of Engineering and Management Vaud (HEIG-VD), University of Applied Sciences Western Switzerland (HES-SO), Switzerland. E-mail: {marina.zapater}@heig-vd.ch.

task performance by appropriately increasing core frequency and temperature [13], or focusing on lifetime reliability by controlling the increased chip temperature [11]. Indeed, a major challenge in integrating reliability management with performance- and energy-aware problems is that the tasks' performance (i.e., QoS) is largely affected by the dynamic temperature and endurance restrictions of different components of a server processor (i.e., cores and cache). Thus, the overall optimization of the whole system on performance, reliability, and energy efficiency remains an open challenge as it requires a deep assessment of the previous techniques. More importantly, system-wide optimization requires dynamically changing the optimization goals during run-time to meet the system constraints and considering the trade-offs across these optimization objectives. These trade-offs represent opposite effects: when the performance of the HPC server is increased, the frequency level and, accordingly, the power consumption of the processor increase, thus leading to a high operating temperature and subsequently reducing the lifetime reliability of the system exponentially. Therefore, it is essential to introduce a co-optimization method to reach a balance between performance and reliability objectives. From the cache perspective, the same trade-off exists. A hybrid cache architecture consumes less energy than the traditional SRAM architectures; while RRAM suffers from limited endurance cycles (i.e., reliability issue). Therefore, we propose a holistic strategy, i.e., COCKTAIL, that accounts synergistically for all these aspects (i.e., performance, reliability, and energy efficiency) in the HPC context. More precisely, the specific contributions of this work are as follows:

- We introduce COCKTAIL, a multi-COre Co-optimization frameworK with proacTive reliAbILity management that maximizes task performance and server energy efficiency, while guaranteeing the expected design lifetime in highly dynamic environment.
- We propose an endurance-aware cache selection method for the hybrid cache architecture to fully exploit the benefits of different cache technologies (i.e., high endurance of SRAM and high energy efficiency of RRAM) and managing their lifetime limitations.
- We propose a dynamic model predictive control (MPC)-based reliability management to set cores DVFS level appropriately and maximize task performance, while utilizing the CPU lifetime deposits efficiently. We also present a proactive queue optimization method to improve overall tasks QoS with respect to their end time constraints.
- Our results demonstrate that COCKTAIL can reach similar results in terms of energy consumption when compared to other state-of-the-art reliability- and energy-aware methods, while ensuring a lifetime longer than 5 years for the power-hungry components of a server processor. Moreover, COCKTAIL improves QoS by up to 60% and reduces tasks congestion and overdue time by up to 80% when compared to conventional approaches.

The remainder of this paper is organized as follows. Sect. II reviews related work. In Sect. III, we provide an overview of the problem description and target scenario. Sect. IV describes the used system modeling for multi-core server processors. In Sect. V we introduce our proposed proactive reliability-aware method. Sect. VI and VII present the experimental setup and results, followed by the conclusion in Sect. VIII.

## II. Related Work and Background

We divide the previous research in two categories: 1) energy-efficient hybrid cache techniques and 2) system reliability management.

### A. Energy-Efficient Hybrid Cache Techniques

SRAM-based caches are the default standard for server design due to their performance. However, this technology is not suitable for future energy-efficient servers design because of its high leakage power consumption [8]. In this context, recent studies resort to NVM technologies, such as RRAM, STT-MRAM, and PCRAM. Nevertheless, these caches significantly suffer from lower endurance (i.e., shorter lifetime) [14].

In order to alleviate this problem and address energy-reliability trade-offs, hybrid cache architectures (i.e., SRAM+STT-RAM and SRAM+RRAM) are promising solutions, as studied in previous works [8], [15]. These solutions target NVM as the main LLC for most occasions to reduce cache power consumption, and use SRAM for performance compensation when necessary. In this case, they can benefit from the advantages of both techniques, namely performance and energy efficiency. Nonetheless, NVM technology suffers from the limited endurance cycles; thus, endurance- and reliability-aware management remain an open challenge for the hybrid caches.

Prior studies propose different reliability management schemes to avoid NVM failures during its design lifetime [8], [16]. Chen *et al.* [8] use a fixed access threshold in hybrid cache design to decide when to power on/off SRAM cache. When powering off the SRAM, they can save unnecessary leakage power consumption. However, this technique does not consider lifetime reliability management for NVM, leading to potential failure associated with setting an inappropriate threshold and using NVM cache frequently. Thus, Beiji *et al.* [16] introduce an endurance-aware RRAM memory management method. They firstly build an endurance model for RRAM memory to get runtime endurance information for each memory bank. Then, they reduce the access frequency to the banks with lower endurance. In this case, different memory banks can share a similar endurance profile and system lifetime can be guaranteed. Nevertheless, this approach does not consider a reliability management for hybrid cache design (e.g., SRAM+RRAM) such that its benefits become limited in terms of performance for HPC tasks.

### B. Server Reliability Management

Server reliability management techniques aim to find a balance between performance and reliability constraints. To control the reliability, temperature is the main factor greatly affecting system lifetime, and thermal management is an effective strategy to guarantee the lifetime period of a computing server [12], [17]. One universal approach to ensure CPU

design lifetime is to throttle CPU frequency to decrease temperature, keeping it below a certain threshold (e.g., 343.15 K = 70 °C). This limits performance, and can impact tasks' QoS.

In addition, a recent study demonstrates that high system temperature is not a direct reason for CPU failure [9]. It is actually induced by other different failure mechanisms, such as EM, SM, TDDB, TC, and NBTI. Srinivasan *et al.* [10] introduce a unified reliability model for microprocessors considering all these failure mechanisms. With this fine-grained model, we can quantify the system's reliability information, making it possible to better manage the CPU lifetime at runtime. In this direction, one approach [13] first deposits the cores lifetime during the period of low temperature and workload. Then, it consumes the stored lifetime deposit to boost performance in the period of heavy workload. Another direction is to increase the lifetime of CPU by limiting the system temperature and performance [11]. By using reliability model and lifetime information, the above methods can optimize performance or lifetime. However, these approaches fall short in presenting an efficient method to address the energy-performance trade-offs along with a temperature-dependent lifetime reliability control.

To control the CPU temperature, different DVFS-based management methods have been proposed in HPC contexts [18]–[21]. Basireddy *et al.* [18] present a workload-aware runtime energy management technique for efficient DVFS control. Then, Singh *et al.* [19] propose an adaptive resource allocation approach to optimize both QoS and energy for HPC data centers. However, these methods do not consider thermal and reliability management for the whole system. On the other hand, other studies [20], [21] consider performance, thermal and reliability in state-of-the-art heterogeneous multi-processor architectures. Nonetheless, these works do not consider the use of hybrid cache architectures and advanced control methods to set the DVFS level appropriately in very dynamic scenarios considering the different and variable reliability effects on resistive and main SRAM technologies. Therefore, some works [17], [22] propose MPC-based methods that try to reach a fixed target temperature by setting DVFS appropriately. Nevertheless, they degrade the performance of tasks to reduce CPU temperature.

In this paper, we propose to jointly incorporate a reliability-aware MPC-based method and a hybrid cache management in a multi-core server processor to address the reliability-performance trade-offs by dynamically changing the target temperature limit with respect to tasks requirements. To the best of our knowledge, COCKTAIL is the first to address all these aspects together in a holistic framework.

## III. Problem Description

In this section we provide a description of the overall scenario, the system we optimize, and the main assumptions taken. Fig. 1 illustrates the proposed scenario and strategy, including inputs and outputs. The goal of the proposed strategy (i.e., COCKTAIL) is to co-optimize task QoS, energy efficiency, and lifetime reliability of a multi-core server processor using a hybrid cache architecture (SRAM+RRAM).

In our system, when a task arrives, it is first sent to the tasks queue (*queue*). Then, we choose the next task from the queue

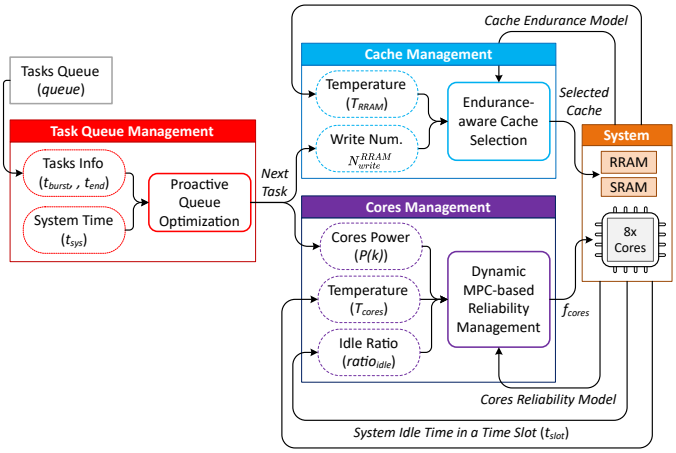



Fig. 1. Overall diagram of the proposed scenario and framework, i.e., COCKTAIL, including Proactive Queue Optimization, Endurance-aware Cache Selection, and Dynamic MPC-based Reliability Management blocks.

at the current system time ($t_{sys}$) for being executed on the processor (when the current task finished). At each time ($t_{sys}$), only one task runs on the server. In order to improve the tasks QoS, the Task Queue Management block takes all the tasks information (i.e., arrival time, amount of time required for being processed on server - burst time, and expected end time) and tries to re-order the tasks sequence in *queue* to reduce the number of overdue tasks (i.e., surpassing expected end time).

After determining the tasks order for execution from current system time ($t_{sys}$), we select the first task (head of queue) as the next task and adjust the system parameters (i.e., cache selection and cores frequency) based on its characteristics (e.g., number of accesses to LLC and required power consumption) to finish the task. Our goal is to achieve the best energy efficiency and performance, while maintaining the lifetime reliability of the whole HPC system.

In this work, we consider a hybrid cache architecture to combine the advantages of different cache technologies (SRAM and RRAM). SRAM provides higher speed for the writes but at the cost of higher leakage power consumption, while RRAM technology is more energy efficient but suffers from limited endurance (numbers of write cycles) [14]. Hence, for each task, we choose the cache (between SRAM and RRAM), at each time ($t_{sys}$) during the task execution time, to reduce the energy consumption, while guaranteeing the expected design lifetime of RRAM. Given task and system information (number of needed writes to RRAM-$N_{write}^{RRAM}$, RRAM cache temperature-$T_{RRAM}$, and endurance model) during the task execution time, we compute the lifetime deposit of RRAM cache to optimize the operation of caches at runtime.

From the cores perspective, for each task, we try to find the best DVFS for the cores, maximizing the task QoS (performance) while satisfying the cores lifetime constraints. We assume each core has its own frequency, thus, $f_{cores}$ contains information for all the cores of the processor. In addition, during the task execution time, we adapt the cores' frequency using a dynamic MPC-based method to dynamically control the temperature and achieve the best performance (maximum allowed power consumption for cores with respect to the lifetime deposit). For a time slot ($t_{slot}$), we estimate the system

idle time ratio (i.e., system idle time during a time slot divided by $t_{slot}$) in order to better use the lifetime deposit of the cores for that time slot (efficient lifetime deposit distribution among all the tasks). To propose a solution to both the cache selection and cores reliability-performance management, we need to accurately model the system.

## IV. System Characterization and Modeling

In this section we first detail the type of tasks tackled in this paper. Then, we describe all the models needed for energy, performance, temperature and reliability management of a multi-core server processor with a hybrid cache architecture.

### A. Task Characterization

For our target HPC tasks, we consider QoS in terms of execution time to measure the performance of each task. We compute the burst time of the task ($t_{burst}$), when the task runs on the server at maximum performance (highest frequency). However, $t_{burst}$ can be degraded to $\hat{t}_{burst}$ by lowering the CPU frequency to enhance energy efficiency and temperature-dependent reliability management of the server processor.

To compute $\hat{t}_{burst}$, we first measure and profile each task's total number of instructions ($inst_{total}$) and instructions per cycle ($IPC$) information per core using perf tool [23]. Then, we calculate the total number of instructions per second ($IPS$) executed by all the cores as: $IPS = IPC \cdot \sum_{i=1}^{N_{cores}} f_i$, where $N_{cores}$ is the number of cores and $f_i$ is the frequency of the $i^{th}$ core. In this case, each core can get its own frequency level (i.e., $f_i$). In our approach, we first assume that the execution of the task starts from $t_{start}$ and the number of remaining instructions ($inst_{rem}$) is $inst_{total}$. Then, at each time step (i.e., 1 second), we update the remaining instructions for the task as: $inst_{rem} = inst_{rem} - IPS$. Finally, at time $t_{end}$, all the needed instructions of the task are finished (i.e., $inst_{rem} \leq 0$), and then $\hat{t}_{burst} = t_{end} - t_{start}$. Moreover, if there are already some tasks in the queue, the task needs to wait ($t_{wait}$) until it is ready for execution. Therefore, total execution time of the task is $t_{exe} = \hat{t}_{burst} + t_{wait}$ and, accordingly, the QoS degradation is computed as $QoS = t_{exe}/t_{burst}$, achieving its highest value of '1', when there is neither waiting nor performance degradation.

In addition to execution time ($t_{exe}$) that should be minimized, end time constraint is another metric that also quantifies the performance of tasks. The system should guarantee that tasks finish before their expected end time limits (i.e., $t_{end}$). This provides a margin for our strategy to play with frequency scaling and task queue optimization, while meeting the expected end time thresholds of the tasks (i.e., $t_{exe} \leq t_{end}$). Otherwise, we consider the task that $t_{exe} > t_{end}$ as an overdue task.

### B. Hybrid LLC Architecture

Recently, most designers and manufacturers focus on SRAM. While SRAM has been designed to meet the performance goals and provide a wider working temperature range (i.e., long lifetime period), it suffers from high leakage power and low energy efficiency. On the contrary, NVM technologies, such as RRAM, offer low leakage power consumption (higher energy efficiency) with more density (higher data storage capacity than SRAM), but at the expense of at least four orders of magnitude less endurance capabilities. Therefore, in this paper, we consider a hybrid cache architecture (i.e., SRAM+RRAM). The detailed specifications and characteristics of both technologies have been introduced in Table II in Section VI-A2.

### C. Server Processor Power Characterization

As introduced in Section I, we consider in our model the two main contributors to the overall power consumption of the server processor: 1) CPU power (core power) containing 8-core ($N_{cores} = 8$) power consumption and 2) uncore power, which includes all components outside the core region, like LLC, memory controller, and IO subsystems.

*1) Core Power:* In our work, power profiling is performed on a homogeneous multi-core processor with identical cores. Therefore, each core maintains the same power value under a fixed frequency, when executing a task. For each task, core power consumption, including static and dynamic power, is measured as a function of frequency levels, ranging from 1.2 GHz to 3.5 GHz (16 levels). We obtain the power traces per task for each frequency level for an Intel Xeon E5 platform using running average power limit (RAPL) interface [24]. Then, we use a linear model (i.e., considering the same power usage for all the cores) to compute each core power consumption at each frequency. In this case, we characterize power traces for each task per core with respect to its individual frequency information. This is a common methodology we have adopted from similar previous studies [22], [25].

*2) Uncore Power:* For traditional SRAM cache, prior work [25] considers the worst-case power consumption as a constant value. Nevertheless, in this paper for hybrid caches, the leakage (as shown in Table II in Section VI-A2) and dynamic LLC power model for each technology was extracted by measuring a 16 MB capacity (the same size for both technologies), using NVSim simulator [26]. We also empirically measure the worst-case memory controller and IO subsystem power consumption overhead of the processor package (i.e., Intel Xeon v4 platform).

### D. Server Processor Thermal Model

For our target multi-core HPC platform, we extract the thermal model using the 3D-ICE simulator [27]. This thermal model provides temperature information for our reliability model, and it is integrated with dynamic MPC to enable cores reliability management.

The extracted thermal model mainly consists of three matrices: $G$, $C$, and $B$, indicating thermal resistance, thermal capacitance, and power injection matrix of the processor, respectively. Power injection matrix ($B$) contains the power distribution of the processor (the floorplan can be found in [28]). Therefore, the processor thermal model can be expressed as follows:

$$GT(t) + C\frac{dT(t)}{dt} = BP(t) \tag{1}$$

where $T(t)$ is the temperature distribution of the processor at time $t$, and $P(t)$ is the power consumption values of processor components obtained by the server power characterization, when running one task.

TABLE I
KEY PARAMETERS FOR RELIABILITY MODELING OF THE CORES

| Parameter | Description | Value |
|---|---|---|
| $E_a$ | Activation energy | 0.9 eV |
| $k_B$ | Boltzmann coefficient | 8.62 eV K$^{-1}$ |
| $T_{vapor}$ | Vapor deposition temperature | 500 K |
| $q$ | Coffin-Manson exponent | 2.35 |

As we measure the power values in a discrete time (time instance), we discretize the thermal model using Backward Euler method, as follows:

$$T(k) = AT(k-1) + B_d P(k) \tag{2}$$

where $T(k)$ and $P(k)$ are discretized version of $T(t)$ and $P(t)$, and $A$ and $B_d$ are extracted from $G$, $C$, and $B$ matrices, respectively. Indeed, $T(k)$ includes temperature information for the whole processor. We select the parts of interest using following equations:

$$T_{cores}(k) = L_{cores} T(k),\ T_{RRAM}(k) = L_{RRAM} T(k) \tag{3}$$

Using the matrix $L_{cores}$ and $L_{RRAM}$, we obtain the temperature distribution for cores ($T_{cores}$) and RRAM cache ($T_{RRAM}$), respectively.

### E. Reliability Modelling of The Server Processor

In this section, given the thermal model, we define the lifetime reliability models for two components: 1) Cores and 2) RRAM cache.

*1) Cores Reliability Model:* We take into account five main failure effects for cores based on previous works [10], [29], [30], as follows:

$$MTTF_{EM} \propto \frac{exp(\frac{E_a}{k_B T_{cores}})}{(V f \alpha)^{C_{EM}}} \tag{4}$$

$$MTTF_{SM} \propto |T_{vapor} - T_{cores}|^{-C_{SM}} exp(\frac{E_a}{k_B T_{cores}}) \tag{5}$$

$$MTTF_{TDDB} \propto (\frac{1}{V})^{a - bT_{cores}} exp(\frac{X + \frac{Y}{T_{cores}} + ZT_{cores}}{k_B T_{cores}}) \tag{6}$$

$$MTTF_{TC} \propto (\frac{1}{T_{cores} - T_{amb}})^{q} \tag{7}$$

$$MTTF_{NBTI} \propto [\ln(\frac{c_{n1}}{1 + 2e^{\frac{c_{n2}}{k_B T_{cores}}}}) - \ln(\frac{c_{n1}}{1 + 2e^{\frac{c_{n2}}{k_B T_{cores}}}} - c_{n3}) \cdot \frac{T_{cores}}{e^{\frac{-c_{n4}}{k_B T_{cores}}}}]^{\frac{1}{\beta}} \tag{8}$$

where $MTTF_{EM}$, $MTTF_{SM}$, $MTTF_{TDDB}$, $MTTF_{TC}$, and $MTTF_{NBTI}$ represent mean time to failures (MTTFs) under the effects of EM (transport of material), SM (caused by mechanical stress), TDDB (caused by gate oxide breakdown), TC (caused by core temperature variations), and NBTI (increasing the transistors threshold voltage), respectively. In Eqs. 4-8, $V$, $f$, and $\alpha$ indicate voltage, frequency, and activity factor of the cores, respectively. The other main parameters with their values are summarized in Table I, and the fitting parameters (i.e., $C_{EM}$, $a$, $X$, etc.) are extracted from the previous work [10].

Finally, we use the industry standard sum-of-failure-rates (SOFRs) model [10] to incorporate all the five reliability concerns, as follows:

$$MTTF_{cores} = (\sum \frac{1}{MTTF_{single}})^{-1} \tag{9}$$

where $MTTF_{single}$ represents each aforementioned lifetime reliability model (i.e., EM, SM, TDDB, TC, and NBTI).

*2) RRAM Reliability model:* Cache reliability model is defined based on the endurance. Each technology has its own endurance that can be measured in terms of the total number of write cycles. In our hybrid cache architecture, SRAM has an endurance of 1E16, while RRAM endurance is around 2E10 cycles at 300 K [14].

Previous work [16] showed that the maximum endurance of RRAM ($endu$) is dependent on its operating temperature (i.e., $T_{RRAM}$ computed by the thermal model), as follows:

$$endu(T_{RRAM}) \approx (c_{r1} T_{RRAM} e^{\frac{E_a}{kT_{RRAM}}})^{c_{r2}-1} \tag{10}$$

where $c_{r1}$ and $c_{r2}$ are material-dependent constants, which are 8.99E-3 and 4, respectively. Fig. 2 shows the relationship between the maximum endurance and temperature of RRAM cache based on Eq. 10.

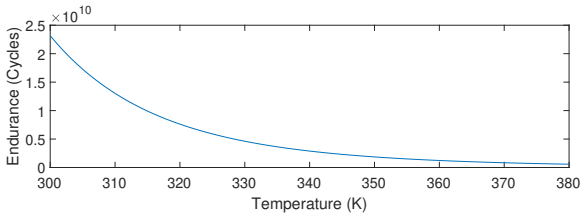



Fig. 2. Relationship between the maximum endurance and temperature of RRAM cache.

## V. COCKTAIL - MULTI-CORE CO-OPTIMIZATION FRAMEWORK WITH PROACTIVE RELIABILITY MANAGEMENT

In this section we introduce our proposed strategy, i.e., multi-COre Co-optimization frameworK with proacTive reliAbIlity management (COCKTAIL), which consists of three main blocks: A) proactive queue optimization, B) endurance-aware cache selection, and C) dynamic MPC-based reliability management.

### A. Proactive Queue Optimization

In order to improve the QoS of the executed HPC tasks, we propose a proactive queue optimization method, which is based on the shortest job first (SJF) approach [31], [32]. Different from the traditional first come first served (FCFS) approach [33], which strictly follows the order of tasks arrival time, SJF reorders the tasks queue by bringing the shortest job (job with the minimum burst time) to the head of queue for execution. However, SJF sends tasks with the longest execution times to the end of the queue, leading to a significant QoS degradation for these tasks. In addition, SJF does not take into account the waiting time of the tasks in the queue that can violate the expected end time limits (i.e., overdue tasks).

In order to alleviate drawbacks of SJF, our method considers both burst time and end time limit of tasks, as shown in

**Algorithm 1:** Proactive Queue Optimization

**Input :** Current system time ($t_{sys}$), Tasks queue (*queue*) and information: burst time ($t_{burst}$) and end time ($t_{end}$)
**Output:** Selecting next task ($task_{next}$) for execution

1 **if** *queue is empty* **then**
2  System is idle;
3 **else**
4  $task_{urgent} \leftarrow$ Find tasks with $t_{end} - t_{sys} < t_{burst}$;
5  **if** $task_{urgent}$ = *Null* **then**
6  $task_{shortest} \leftarrow$ Find the task with shortest $t_{burst}$;
  /* Proactive Action */
7  $t_{sys\_next} \leftarrow t_{sys} + task_{shortest}.t_{burst}$;
8  $task_{urgent} \leftarrow$ find task with $t_{end} - t_{sys\_next} < t_{burst}$ if $task_{shortest}$ is being selected;
9  **if** $task_{urgent}$ = *Null* **then**
10  $task_{next} \leftarrow task_{shortest}$;
11  **else**
12  $task_{next} \leftarrow task_{urgent}$;
13  **end**
14  **else**
15  $task_{next} \leftarrow task_{urgent}$;
16  **end**
17 **end**

Algorithm 1. The system is idle when the tasks' queue (*queue*) is empty (lines 1−3). Otherwise, we first find tasks ($task_{urgent}$), which breach their end time limits ($t_{end}$) with respect to the current system time ($t_{sys}$) and burst time ($t_{burst}$) (line 4). If there is no $task_{urgent}$ (lines 5−14), we perform a proactive action for selecting the next task ($task_{next}$), while avoiding the overdue tasks. For this purpose, we first select the shortest task ($task_{shortest}$) from the queue (line 6). We check the possible existence of urgent tasks if this shortest task is being executed (line 7−8). If there is any urgent task, we move it to the head of queue; otherwise, the selected shortest job is the best candidate for being executed (lines 9−13).

Fig. 3 shows the efficiency of our proposed method versus SJF to guarantee the expected end time thresholds. According to Algorithm 1, our proactive queue optimization method brings task 4 before 5 to execute, avoiding the overdue task. Although other methods [32], [34], [35] take into account the end time constraints of running tasks, they fall short in prioritizing the tasks for executions that breach their end time limit in the future due to making a short-sight decision. Besides the end time limit, overall QoS (i.e., execution time) is another metric that also indicates the tasks' performance. In this context, slack-based methods [36] guarantee the end time limit by executing the tasks with the lowest slack times. However, they neglect the overall QoS of the system because a shorter task may suffer from a longer waiting time. On the contrary, our proposed proactive queue optimization method aims to improve the overall QoS of the system while guaranteeing the end time limit of the tasks.

### B. Endurance-Aware Cache Selection

Based on Eq. 10 and Fig. 2, the endurance of RRAM cache exponentially decreases by increasing the temperature compared to its baseline operating temperature (i.e., 300 K = 26.85 °C). Hence, in order to manage the lifetime of RRAM,

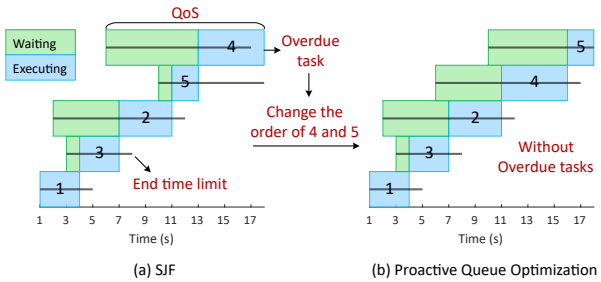



Fig. 3. SJF versus proactive queue optimization.

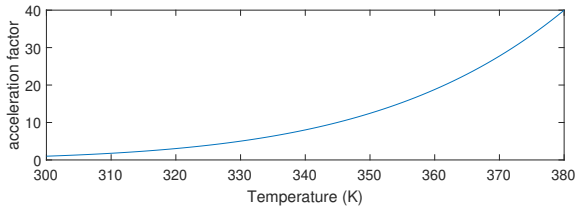



Fig. 4. Acceleration factor with temperature variations of RRAM.

we need to consider the effect of different temperatures on maximum endurance. Therefore, we introduce an acceleration factor, as follows:

$$f_{af}(T_{RRAM}) = endu(300\,\mathrm{K})/endu(T_{RRAM}) \tag{11}$$

Fig. 4 illustrates the acceleration factor of RRAM at different temperature with respect to its nominal value at 300 K ($endu_{total}$). We use this factor to compute the endurance consumption ($endu_{consumed}$) as the function of number of writes ($N_{write}^{RRAM}$) to the RRAM cache (i.e., $endu_{consumed} = f_{af}(T_{RRAM}) \cdot N_{write}^{RRAM}$). In this case, for temperatures higher than (300 K), the acceleration factor would be higher than '1', leading to a higher endurance consumption for a specific number of writes to the RRAM.

Using the nominal endurance value ($endu_{total}$) and endurance consumption ($endu_{consumed}$), we propose a cache selection method based on the reliability banking technology management mechanism [13]. In reliability banking technology mechanism, the main goal is to use $endu_{total}$ during a system design lifetime, which is selected to be 5 years in our case. Hence, we define an endurance consumption rate ($endu_{limit} = endu_{total}/(design\ lifetime = 5\ years)$) that describes the average consumption rate of RRAM during five years. This consumption rate limit ($endu_{limit}$) guarantees the operation of RRAM for this specific period.

Then, in our proposed method, as shown in Algorithm 2, we try to select a cache between RRAM and SRAM for the system, while ensuring the RRAM cache endurance limit. We have an endurance deposit "account" ($endu_{deposit}$) for the RRAM cache and it will receive "salary" of $endu_{limit}$ in every management time. If $endu_{deposit}$ is larger than 0 (i.e., positive saving), RRAM can maintain reliability larger than its design lifetime. Consequently, we select the RRAM cache option for energy efficiency, while respecting its lifetime limit (lines 1−2). In addition, RRAM cache experiences an endurance consumption ("expenditure") of $endu_{consumed}$ (line 3). Otherwise, in case of $endu_{deposit} < 0$, we select SRAM cache

**Algorithm 2:** Endurance-Aware Cache Selection

**Input :** Acceleration factor ($f_{af}$), RRAM temperature ($T_{RRAM}$), and task write number ($N_{write}^{RRAM}$)
**Output:** Cache selection between RRAM and SRAM

```
1 if endu_deposit > 0 then
2 |   Select RRAM cache;
3 |   endu_consumed ← f_af(T_RRAM) · N_write^RRAM;
4 else
5 |   Select SRAM cache;
6 |   endu_consumed ← 0;
7 end
8 endu_deposit ← endu_deposit + (endu_limit − endu_consumed);
```

(lines 4−5). In this case, there is no endurance consumption for RRAM cache when SRAM cache is selected (line 6). After selecting the cache, the net endurance saving in each management time is computed as $endu_{limit} - endu_{consumed}$. Finally, we update the $endu_{deposit}$ with respect to the net endurance saving, and save it for subsequent tasks (line 8).

The power overhead of our proposed cache selection method is negligible from the software perspective due to the use of a low-overhead rule-based policy. From the hardware perspective, the overall energy consumption reduction can be achieved by introducing the hybrid cache architectures (i.e., hardware design and managing its operation) instead of current cache management implementations (e.g., data migration) [7], [8].

### C. Dynamic MPC-Based Reliability Management

In this section we propose a dynamic MPC-based reliability management method to improve the task performance. Thus, we split this method in two parts: 1) cores reliability management and 2) dynamic MPC for frequency scaling.

*1) Cores Reliability Management:* Similarly to endurance-aware cache selection method, we use a reliability banking technology method for cores. We consider a core lifetime deposit "account" ($lifetime_{deposit}$) and a "salary" as nominal lifetime consumption rate ($lifetime_{limit}$), in each management time. We compute the net lifetime saving as $lifetime_{limit} - lifetime_{consumed}$, where $lifetime_{consumed}$ (i.e., lifetime consumption) is obtained by cores temperature-dependent reliability model presented in Section IV). Our proposed reliability management method aims to give the opportunity for increasing task performance (i.e., reducing execution time), while keeping the cores $lifetime_{deposit}$ always above 0.

In the following subsection, we propose a dynamic MPC-based method to control the temperature and reliability deposit of cores by setting an appropriate frequency level, while maximizing task performance.

*2) Dynamic MPC:* Traditional MPC controllers [17], [22] consider the maximum allowed temperature for the processor (383.15 K = 110 °C), when the lifetime deposit is still positive. However, setting this temperature threshold uses the deposit lifetime for a limited number of tasks. Thus, for the rest of tasks in a certain time, the performance would be sacrificed due to the reliability management (control deposit by temperature).

Fig. 5 shows the lifetime deposit and consumption rates of the cores with respect to the different operating temperature, obtained by the reliability model and reliability banking mechanism. Based on the analysis, temperature 343.15 K (i.e., 70 °C) is the nominal temperature that converges to the nominal lifetime limit ($lifetime_{limit}$). Moreover, although the lifetime reliability is exponentially consumed by increasing the cores temperature, we cannot get an exponential performance improvement.

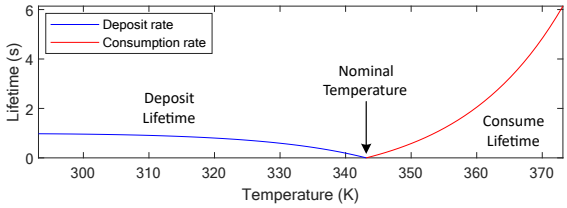



Fig. 5. Lifetime deposit and consumption rate under different temperature of cores.

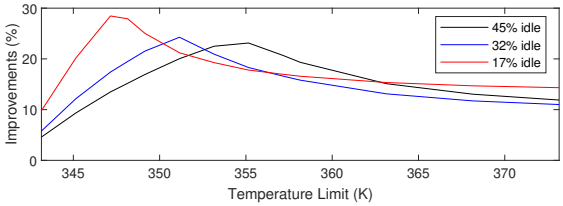



Fig. 6. Performance improvement with different temperature limits for the cores.

We find that the most efficient way is to set the temperature limit dynamically according to the system status (i.e., active and idle ratio). Fig. 6 shows the performance improvement of the system (capability of increasing cores frequency) for three different system idle ratios under different temperature limits. We assume that the system consumes the lifetime deposit during the active times for maximizing the performance, while it increases the deposit during the idle times (lowest possible temperature due to the lowest frequency). As the idle time ratio increases from 17% to 45%, the best temperature limit increases from 348.15 K to 353.15 K (i.e., 75 °C to 80 °C) to gain higher performance (higher frequency). This is because the lifetime deposit is consumed by a shorter activity period (lower load). Thus, we can increase the consumption speed of lifetime deposit by increasing the temperature of the cores.

Based on the aforementioned analysis, our method computes the lifetime consumption rate limit dynamically with respect to the active times of the system, as follows:

$$lifetime_{limit}^{dyn} = \frac{lifetime_{deposit}}{(1 - ratio_{idle}) \cdot t_{total}} \tag{12}$$

where $lifetime_{deposit}$, $t_{total}$, and $ratio_{idle}$ are current lifetime deposit, total time, and idle time ratio of the system under a reliability management time, respectively. Eq. 12 assumes that $lifetime_{limit}^{dyn}$ uses all the $lifetime_{deposit}$ for the next time slot, while achieving the best performance. In order to model realistic scenarios, as $ratio_{idle}$ is unknown for the next time slot, we use a last-value predictor to estimate this parameter. The last-value predictor assumes that the idle ratio of the next time slot is exactly the same as current slot. However, due to the prediction error, especially when the predicted value is higher than the real one, we consider a correction factor

**Algorithm 3:** Dynamic MPC-Based Reliability Management Method

**Input :** Current cores power ($P(k)$), cores temperature ($T_{cores}$), and system idle ratio ($ratio_{idle}$)
**Output:** Cores frequency ($f_{cores}$)

$lifetime_{consumed} \leftarrow$ Lifetime consumption computed by the reliability model and $T_{cores}$;
$lifetime_{deposit} \leftarrow lifetime_{deposit} + (lifetime_{limit} - lifetime_{consumed})$;
/* Dynamic MPC */
$T_{cores}^{target} \leftarrow$ Compute dynamic target temperature according to $ratio_{idle}$ and $lifetime_{deposit}$;
$\Delta P \leftarrow$ Cores power adjustment obtained by MPC;
$P(k+1) \leftarrow P(k) + \Delta P$;
$f_{cores} \leftarrow$ Find the best frequency level w.r.t. the power characterization and $P(k+1)$;

($\gamma < 1$) for $lifetime_{limit}^{dyn}$ (i.e., $\gamma \cdot lifetime_{limit}^{dyn}$).

Given the $lifetime_{limit}^{dyn}$, reliability, and lifetime deposit and consumption rate models (Fig. 5), we first find the target cores temperature ($T_{cores}^{target}$) for the next control phase. Then, we build a MPC controller [17], [22] to find the power budget for the cores and, consequently, cores frequency based on the current cores temperature ($T_{cores}$) and $T_{cores}^{target}$.

For our MPC controller, as presented in Algorithm 3, we first define the target temperature distribution over $N_p$ steps into the future (prediction horizon $N_p$) as:

$$\mathbb{T}_{target} = [T_{cores}^{target}, ..., T_{cores}^{target}] \tag{13}$$

Furthermore, the predicted cores temperature ($T_p$) in MPC controller is defined as:

$$T_p = V\hat{T}(k) + \Phi\Delta P \tag{14}$$

where matrices $V$ and $\phi$ are extracted from the processor thermal model, presented in Eq. 2. $\hat{T}(k)$ is represented as $[\Delta T(k); T(k)]$, where $\Delta T(k) = T(k) - T(k-1)$. $\Delta P$ (power adjustment) is the control knob for the future $N_c$ steps (control horizon) and computed by minimizing the cost function as:

$$F = (\mathbb{T}_{target} - \hat{T}(k))^T (\mathbb{T}_{target} - \hat{T}(k)) \tag{15}$$

As a result of minimizing cost function $F$, we get:

$$\Delta P = (\Phi^T\Phi)^{-1}\Phi^T(\mathbb{T}_{target} - V\hat{T}(k)) \tag{16}$$

The last step for MPC controller is to update the power consumed by the cores for the next time, i.e., $P(k+1) = P(k) + \Delta P$ (only the first control horizon of $\Delta P$ is used in receding-horizon mechanism), that tracks the target temperature distribution. Finally, $P(k+1)$ is sent to the processor DVFS controller for finding the best frequency level ($f_{cores}$) that meets the updated power.

### D. The COCKTAIL

In order to effectively solve the performance-reliability, and energy efficiency trade-offs for the whole system, COCKTAIL combines all the proposed methods. First, the endurance-aware cache selection method maintains the reliability of the hybrid cache, while contributing to the energy consumption reduction of the system due to the efficient management of the RRAM cache. Second, the dynamic MPC-based reliability management method utilizes the additional power budget provided by the hybrid cache and lifetime deposit of the cores to improve the tasks' performance, while guaranteeing the cores' temperature-dependent reliability constraint.

## VI. Experimental Setup, Scenarios and Comparison Methods

### A. Experimental Setup

*1) Server configuration:* We use a Supermicro SuperBlade server that consists of 8-core ($N_{cores} = 8$) Intel Xeon E5-2667 v4 CPU with 16 frequency levels varying from 1.2 GHz to 3.5 GHz, and 256 GB of memory. The memory subsystem comprises L1 instruction and data cache both of 32 KB and a dedicated L2 of 256 KB per core (floorplan in [28]).

*2) Hybrid LLC Characteristics:* We consider a hybrid cache architecture (i.e., SRAM+RRAM) with the capacity of 16 MB for each technology and without increasing the LLC area in the floorplan. The characteristics of both technologies have been compared and shown in Table II, obtained by NVSim simulator [26] for a 16 MB cache size. In the simulator, we tuned the parameters of the general purpose model in accordance with previous work [37].

In this work, we assume that the SRAM cache does not have an endurance restriction that is higher with respect to the other components of the HPC system. Indeed, the SRAM cache with an endurance of 10E16 can sustain a lifetime period of longer than 15 years in the worst case (i.e., for memory-write-intensive workloads), which is drastically larger than the design lifetime of an HPC server (i.e., 5 years). However, under the same setting, RRAM cache with an endurance of 2E10 is sustained for a period of 1 year, when its operating temperature is 300 K = 27 °C; and the lifetime of RRAM decreases to 1 month once its operating temperature increases to 350 K = 77 °C.

*3) Task Description and Simulation Framework:* To simulate realistic scenarios, we use the SPEC CPU 2017 benchmark suite [38], which consists of HPC tasks (applications). We collect the tasks power and performance statistics (i.e., power, burst time, and LLC read/write accesses) running on the target server under different frequency levels using RAPL [24] and perf [23] tools, respectively. Fig. 7 shows how the read/write ratio varies for each task, enabling the benefits of dynamic cache selection. Also, we used the obtained power, performance, thermal, and reliability models (Section IV) to co-optimize the operation of the whole HPC system by exploiting an in-house high-level simulation tool written in MATLAB, where we coded all the algorithms used in this paper.

TABLE II
Characteristics of 16 MB SRAM and RRAM cache

| | SRAM | RRAM |
|---|---|---|
| Area (mm) | 11.64 | 1.37 |
| Read Latency (ns) | 5.82 | 2.71 |
| Write Latency (ns) | 3.00 | 20.93 |
| Leakage Power (W) | 5.15 | 0.83 |
| Endurance (Cycles) | 10E16 | 10E6-10E12 |

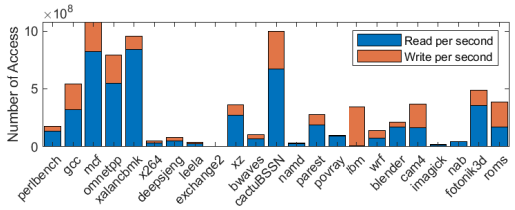



Fig. 7. Different applications (tasks) read and write data to the LLC.

Finally, we fix the control time step for our proposed management methods to 1 second. We perform our experiments for a simulation time (scenario time) of $3 \times 10^7$ seconds (i.e., 347 days), splitting it into 300 time slots (i.e., $t_{slot}$ is $10^5$ seconds) for computing the arrival tasks and system idle time ratio ($ratio_{idle}$). These time values have been chosen to avoid the possible random effects of different workloads and methods. The execution of this $3 \times 10^7$ seconds of simulated time was carried out in 80 minutes on a system equipped with a 6-core Intel i7-8700 CPU@3.2GHz and 16GB of memory, for the whole scenario time of $3 \times 10^7$ seconds.

### B. Scenarios

In order to create different system idle times, we use a Poisson distribution defined by the parameter $\lambda$, as previously done in our related works in the literature [1], [39]. As a result, the number of tasks that arrive in the system at each 1 second during a time slot can be tuned by $\lambda$. Then, we use a uniform distribution to assign the specific benchmarks to the arrived tasks. More specifically, a random number from 1 to 23 (23 SPEC benchmarks) is generated by the uniform distribution to assign one benchmark to one task. In this work, we consider two different scenarios for our experiments.

*1) Scenario I - Static $\lambda$:* In this scenario, $\lambda$ is fixed for all the simulation time slots. We evaluate the effectiveness of our proposed strategy under four different $\lambda$ values (i.e., $4E-4$, $5E-4$, $6E-4$, and $7E-4$). According to Poisson distribution, $\lambda$ controls the average number of arrival tasks in each time slot, i.e., the higher the $\lambda$, the higher the number of arrival tasks is. Therefore, choosing the different values for $\lambda$ can affect the idle time in each time slot. Furthermore, the idle time is also dependent on the benchmark suite and the execution time of the applications on the server. Based on our experiments with SPEC CPU 2017 benchmarks, when $\lambda = 4E-4$, the system idle time ratio is around 45% with an average of 40 tasks per time slot, while for $\lambda = 7E-4$, the idle time ratio is around 4% with an average of 70 tasks. Table III summarizes this information for the different $\lambda$ values.

*2) Scenario II - Dynamic $\lambda$:* In a general scenario, the server load varies during a full day, i.e., high-load during the day (large $\lambda$), and lower during the night (small $\lambda$). Therefore, we investigate the efficiency of our method for a dynamic scenario, where $\lambda$ changes every time slot (i.e., $t_{slot}$).

TABLE III
AVERAGE IDLE TIME AND NUMBER OF TASKS FOR DIFFERENT $\lambda$ PER TIME SLOT

| $\lambda$ | Average idle time per time slot | Average number of tasks per time slot |
|---|---|---|
| $4E-4$ | 45% | 40 |
| $5E-4$ | 32% | 50 |
| $6E-4$ | 17% | 60 |
| $7E-4$ | 4% | 70 |

### C. Comparison Methods and Metrics

In this work we compare COCKTAIL against seven different state-of-the-art reliability and energy optimization methods for HPC systems. All approaches consist of a task queue optimization and cores (in addition to cache) reliability management policy to cover all the possibilities and combinations, as shown in Table IV.

*1) Reactive Method (Reactive) [11]:* This reliability management method sets a hard lifetime threshold based on the nominal lifetime of the cores (i.e., $lifetime_{limit}$). Once current lifetime consumption ($lifetime_{consumed}$) goes beyond the threshold, it reduces cores power consumption to stay in the safe area. Otherwise, it increases power until reaching the threshold limit. This approach ignores the hybrid cache reliability management, and uses only SRAM cache technology for performance goals. Moreover, since this method does not provide any task queue management, we consider the FCFS method for executing the tasks.

*2) Reactive Method with Hybrid Cache (Hybrid Reactive) [8]:* This approach proposes an energy-efficient technique for hybrid cache architectures (i.e., SRAM+NVM). This method sets a threshold for the number of write accesses to the cache. If the number of writes exceeds the threshold, SRAM is used to reduce stress of NVM. Otherwise, SRAM is powered off to save the energy. To adapt this method to our work, we define the threshold based on the RRAM's endurance limit (i.e., $endu_{limit}$) and its design lifetime. However, this approach does not consider cores reliability management. Therefore, we combine the hybrid cache management with reactive method to control the energy, performance, and reliability of the whole HPC system.

*3) FCFS+Static MPC:* In order to show the advantage of advanced control policies, we implemented a static MPC-based method [17], [22] for cores reliability management. In this method, we consider a fixed target cores temperature (i.e., $T_{cores}^{target} = 373.15\,\mathrm{K} = 100\,^{\circ}\mathrm{C}$). MPC tries to reach this target temperature if the cores lifetime deposit is available. Also, we combine this static MPC-based method with our proposed endurance-aware cache selection technique (EAC) and FCFS queue management for executing the tasks.

*4) FCFS+Dynamic MPC:* To evaluate the impact of cores reliability management, we consider the same FCFS and cache selection methods in FCFS+Static MPC jointly with our proposed dynamic MPC-based reliability management policy.

*5) Proactive Queue Optimization with Static MPC (PQ+Static MPC):* For a further comparison with FCFS+Static MPC, we replace the FCFS management by our proposed proactive queue optimization to investigate its impact on task performance.

*6) Proactive Queue Optimization with Hybrid Cache and Dynamic MPC (PQ+Hyb+MPC):* This method indicates the effectiveness of our jointly proposed queue optimization and dynamic core management techniques with the same hybrid

TABLE IV
COMPARISON METHODS WITH THEIR POLICIES

| Methods | Queue | | Cache | | | Core | | |
|---|---|---|---|---|---|---|---|---|
| | FCFS | PQ | SRAM | Hybrid | EAC | Reactive (Rea) | Static MPC | Dynamic MPC |
| Reactive | ✓ | | ✓ | | | ✓ | | |
| Hybrid Reactive | ✓ | | | ✓ | | ✓ | | |
| FCFS+Static MPC | ✓ | | | | ✓ | | ✓ | |
| FCFS+Dyn. MPC | ✓ | | | | ✓ | | | ✓ |
| PQ+Static MPC | | ✓ | | | ✓ | | ✓ | |
| PQ+Hyb+MPC | | ✓ | | ✓ | | | | ✓ |
| PQ+EAC+Rea | | ✓ | | | ✓ | ✓ | | |
| COCKTAIL | | ✓ | | | ✓ | | | ✓ |

cache management technique compared to Hybrid Reactive approach. Moreover, this method illustrates the efficiency of our proposed cache selection policy solely in comparison with COCKTAIL.

*7) Proactive Queue Optimization and our Endurance-Aware Cache Selection (EAC) combined with Reactive core management (PQ+EAC+Rea):* This method shows the benefits of our dynamic MPC-based core management technique solely when compared to COCKTAIL.

We compare these methods in terms of energy consumption, average and worst-case QoS, number of overdue tasks, and overall overdue time, while guaranteeing the lifetime reliability of the system. We define average and worst-case QoS in terms of the average and worst-case execution time of the tasks, running on the server over the whole simulation time. Moreover, overdue tasks are those tasks that violate their expected end time limits during the experiment time.

## VII. EXPERIMENTAL RESULTS

### A. Scenario I - Static $\lambda$

Fig. 8 to 15 show the cores temperature, lifetime deposit and consumption, cache selection decision, and total power consumption, with $\lambda = 7E-4$, for different methods. We start by considering a limited simulation time and only one $\lambda$ case to better show the behavior of different methods. We highlighted the most relevant comparison points by drawing rectangles in figures and adding numbers in a circle that indicate to which figures this trace should be compared to. For instance, in Fig. 9, there is a red rectangle for endurance deposit with a number 8 inside the circle to show the main differences of Hybrid Reactive method from Fig. 8 (i.e., Reactive Method). Also, as the power traces of all the methods are different, we did not highlight them in the figures.

As shown in Fig. 8, Reactive method keeps cores temperature below a limit (343.15 K = 70 °C). This is due to a fixed lifetime threshold for cores to guarantee the lifetime longer than 5 years. The highlighted area in the temperature subplot represents each task running on the server. This method only uses SRAM cache technology to meet the performance goals at the expense of higher leakage power consumption. On the other hand, Hybrid Reactive method (Fig. 9) provides better energy efficiency than Reactive because of selecting RRAM cache for some tasks, while guaranteeing the RRAM endurance constraint. The main difference between Hybrid Reactive and Reactive methods is endurance deposit subplot in Fig. 8 and 9. Reactive method always uses SRAM cache (red background color), while Hybrid Reactive method represents the usage of energy efficient RRAM (green background color). FCFS+Static MPC method (Fig. 10) uses the core lifetime deposit to maximize the overall task QoS (performance) with higher core power consumption because of the increased core frequency. However, the total energy consumption of the system is controlled by selecting the RRAM more frequently than Hybrid Reactive, leading to similar energy consumption, while still maintaining an endurance deposit larger than 0.

FCFS+Dynamic MPC, as shown in Fig. 11, determines a dynamic target temperature with respect to the lifetime deposit and system idle time ratio, which is $T_{cores}^{target} = 352.15\,\text{K}$ (i.e., 79 °C) in this scenario. This gives the opportunity to better distribute the lifetime deposit among all the tasks in a time slot. Therefore, FCFS+Dynamic MPC improves the overall QoS (for all the executed tasks) compared to FCFS+Static MPC that uses the deposit for a limited number of tasks. The main difference between FCFS+Static MPC and FCFS+Dynamic MPC can be seen from subplots of temperature and lifetime deposit in Fig. 10 and 11. PQ+Static MPC method reduces the number of overdue tasks compared to other approaches due to using proactive queue optimization method. For instance, as shown in Fig. 12, it executes the task 9 before 8 to meet its expected end time limit.

Compared to PQ+Hyb+MPC (Fig. 13), COCKTAIL benefits from the proposed endurance-aware cache selection policy (EAC), providing higher energy efficiency (larger green area). This is because PQ+Hyb+MPC uses a conservative endurance consumption limit. To evaluate the effect of cores reliability management, COCKTAIL attains better QoS than PQ+EAC+Rea (Fig. 14) thanks to our dynamic MPC-based core management policy. As a result, COCKTAIL (Fig. 15) integrates the advantages of all these methods to increase the efficiency of HPC systems. To be more precise, Fig. 16, 17, and 18 solely show the specific effects of each proposed policy, including queue optimization, cache selection, core reliability management, on cache endurance deposit, core temperature, and core lifetime deposit compared to other state of the arts.

Table V shows the results for Reactive method under different $\lambda$ values. With the increase of $\lambda$ (less idle time), energy consumption, average QoS, total overdue tasks and time all increase due to the higher number of executed tasks in a time slot. For $\lambda = 7E-4$, tasks occupy the server (almost zero idle time) over the whole time slot, leading to higher number of overdue tasks in the presence of reliability management. As a result, the average QoS is degraded by 4x when compared to $\lambda = 6E-4$. For better comparison, we evaluate the efficiency of different methods (improvements) for

TABLE V
RESULTS FOR REACTIVE METHOD UNDER STATIC $\lambda$ USING ALL THE BENCHMARKS

| $\lambda$ | 4E-4 | 5E-4 | 6E-4 | 7E-4 |
|---|---|---|---|---|
| Energy (J) | 1.2E9 | 1.4E9 | 1.5E9 | 1.7E9 |
| Average QoS | 2.5 | 3.3 | 6.5 | 31.8 |
| Worst-case QoS | 44.0 | 36.4 | 90.7 | 249.0 |
| Overdue tasks (#) | 1453 | 2830 | 7035 | 16320 |
| Overdue time (s) | 2.8E6 | 6.6E6 | 3.9E7 | 4.7E8 |

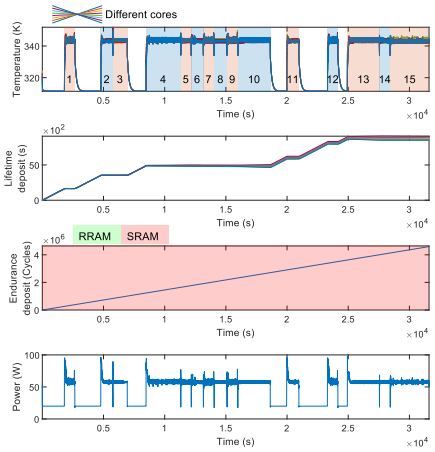



Fig. 8. Reactive method.

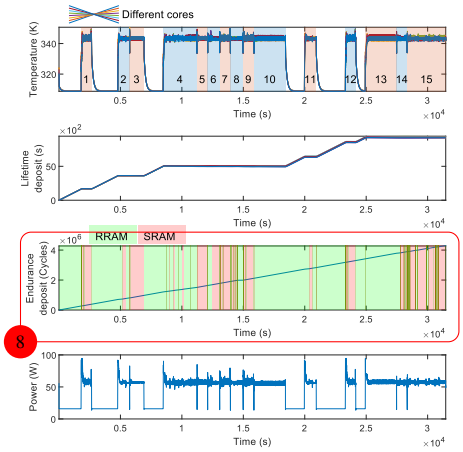



Fig. 9. Hybrid Reactive method.

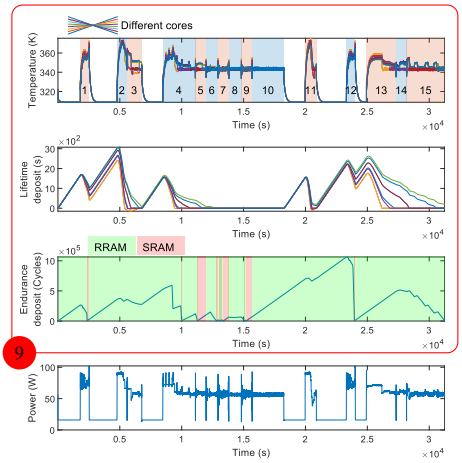



Fig. 10. FCFS+Static MPC.

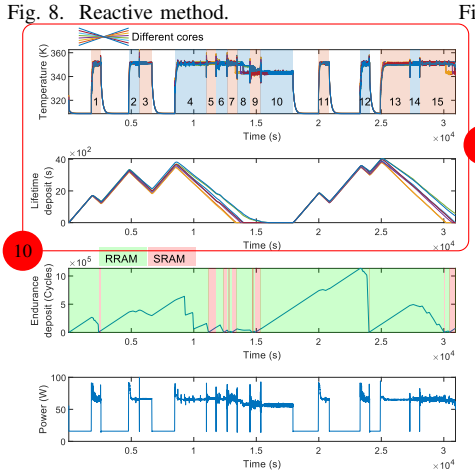



Fig. 11. FCFS+Dynamic MPC method.

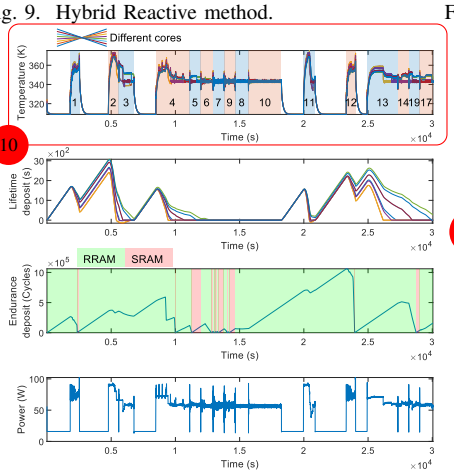



Fig. 12. PQ+Static MPC method.

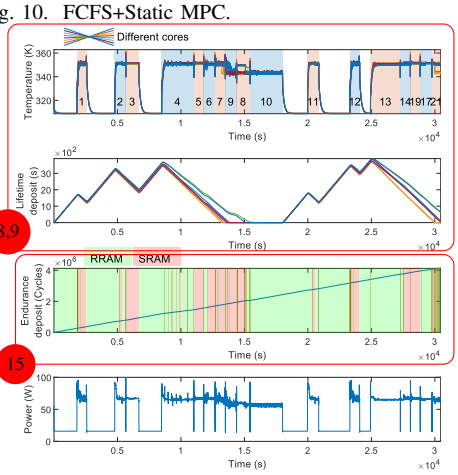



Fig. 13. PQ+Hyb+MPC method.

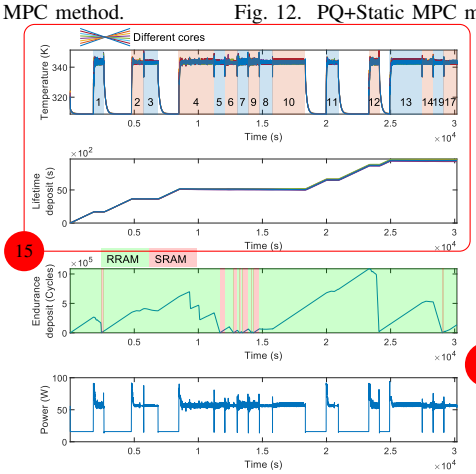



Fig. 14. PQ+EAC+Rea method.

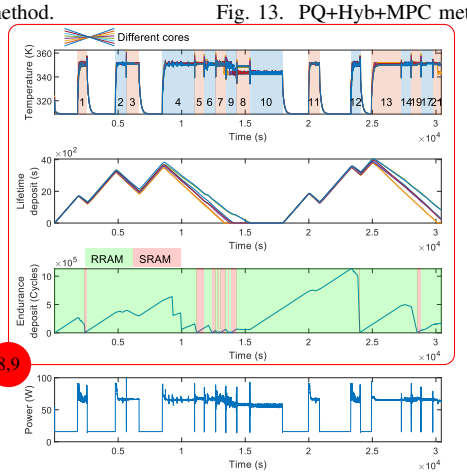



Fig. 15. COCKTAIL.

different metrics compared to Reactive method, as a baseline.

According to the results in Table VI, Hybrid Reactive method achieves up to 6% energy savings than Reactive method. In addition, Hybrid Reactive provides up to 33% and 39% average QoS and overdue time improvements, respectively. This is because with leakage power savings from RRAM cache, the processor has room for increasing the cores power. Hence, it results in further performance improvement under the same temperature limit, especially in tasks congestion situation (higher $\lambda$). However, Hybrid Reactive is conservative in reliability management, always keeping the lifetime deposit positive. FCFS+Static MPC uses the endurance-aware cache selection method that obtains higher energy savings for cache, and it also uses MPC to boost performance. The cache energy savings with higher temperature threshold for the cores given by MPC (maximum allowed cores temperature 373.15 K = 100 °C, which is higher than nominal temperature for lifetime management) brings better performance than Hybrid

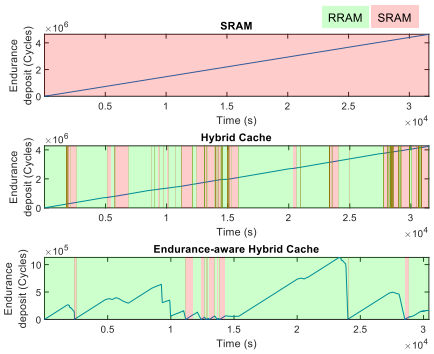



Fig. 16. Endurance deposit comparison of different cache management methods.

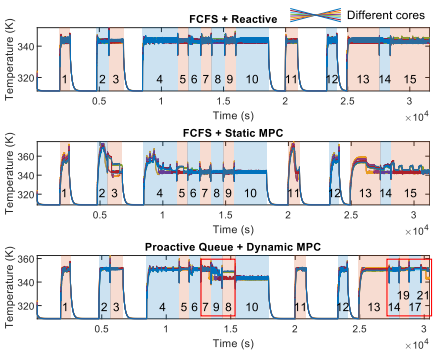



Fig. 17. Temperature comparison of different scheduling and core management methods.

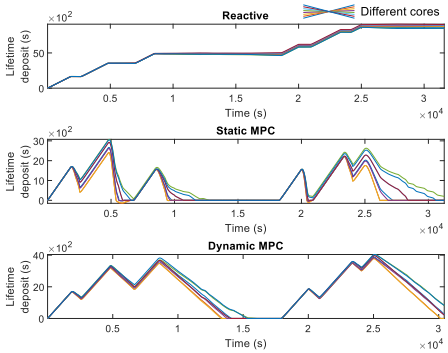



Fig. 18. Lifetime deposit comparison of different core management methods.

TABLE VI
HYBRID REACTIVE, FCFS+STATIC MPC, PQ+STATIC MPC, FCFS+DYNAMIC MPC, PQ+HYB+MPC, PQ+EAC+REA, AND COCKTAIL IMPROVEMENTS (%) COMPARED TO BASELINE (REACTIVE) UNDER STATIC $\lambda$ USING ALL THE BENCHMARKS

| | Hybrid Reactive | | | | FCFS+Static MPC | | | | PQ+Static MPC | | | | FCFS+Dynamic MPC | | | | PQ+Hyb+MPC | | | | PQ+EAC+Rea | | | | COCKTAIL | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ $(xE-4)$ | **4** | **5** | **6** | **7** | **4** | **5** | **6** | **7** | **4** | **5** | **6** | **7** | **4** | **5** | **6** | **7** | **4** | **5** | **6** | **7** | **4** | **5** | **6** | **7** | **4** | **5** | **6** | **7** |
| Energy (%) | 6 | 5 | 3 | 2 | -1 | 0 | 2 | 3 | -1 | 1 | 2 | 3 | -2 | -1 | 1 | 2 | -3 | -2 | 0 | 1 | 7 | 6 | 4 | 3 | -2 | -1 | 1 | 2 |
| Average QoS (%) | 4 | 5 | 9 | 33 | 12 | 11 | 14 | 43 | 23 | 27 | 31 | 47 | 24 | 24 | 28 | 48 | 30 | 34 | 39 | 41 | 17 | 24 | 30 | 47 | 31 | 36 | 42 | 52 |
| Worst-case QoS (%) | 2 | 4 | 7 | 18 | 4 | 6 | 10 | 24 | 59 | 26 | 10 | 24 | 12 | 9 | 27 | 29 | 66 | 23 | 20 | 23 | 59 | 26 | 10 | 25 | 66 | 28 | 23 | 30 |
| Overdue tasks (#) (%) | 4 | 8 | 7 | 9 | 16 | 14 | 13 | 14 | 49 | 50 | 35 | 21 | 43 | 37 | 27 | 18 | 63 | 63 | 46 | 20 | 46 | 49 | 34 | 21 | 65 | 65 | 50 | 26 |
| Overdue time (s) (%) | 10 | 13 | 16 | 39 | 22 | 22 | 24 | 51 | 73 | 77 | 51 | 54 | 59 | 52 | 47 | 56 | 85 | 84 | 64 | 48 | 71 | 76 | 51 | 55 | 86 | 86 | 69 | 60 |

Reactive, while reaching almost similar results in terms of total energy consumption than Reactive method.

Differently from FCFS+Static MPC, PQ+Static MPC exploits the proposed proactive queue optimization method to further improve QoS and reduce the number of overdue tasks with the same energy consumption. In order to investigate the efficiency of our dynamic MPC method compared to static MPC, our results demonstrate that FCFS+Dynamic MPC provides less overdue time without the usage of proactive queue optimization method compared to FCFS+Static MPC. This is due to finding the best temperature threshold with respect to the system idle time ratio and distributing the lifetime deposit among the all executed tasks. In addition, our results show the higher efficiency for dynamic MPC compared to Reactive and our endurance-aware cache selection method (EAC) to Hybrid in terms of QoS and overdue tasks by comparing COCKTAIL with PQ+EAC+Rea and PQ+Hyb+MPC, respectively. Finally, COCKTAIL combines the benefits of proactive queue optimization and endurance-aware cache selection, together with dynamic MPC-based reliability management to co-optimize the system efficiency, while ensuring the system lifetime longer than 5 years, as shown in Table VI.

In summary, COCKTAIL obtains significant improvements on average QoS (up to 52%), worst-case QoS (up to 66%), overdue tasks (up to 65%), and overdue time (up to 86%) compared to the Reactive method, while consuming almost the same energy (i.e., up to 2% improvements) over all the time slots. In addition, our proposed method (i.e., COCKTAIL) outperforms the other methods, providing better trade-offs in all performance metrics. This is because COCKTAIL uses energy saved by the hybrid cache and lifetime deposit to efficiently boost the tasks' performance (i.e., better average and improving the worst-case QoS). Thus, it achieves better overall energy efficiency (i.e., performance/power) compared to other state-of-the-art techniques. Moreover, it has the capability of reordering tasks to reduce the number of overdue tasks and final penalty time in executed tasks.

### B. Scenario II - Dynamic $\lambda$

In this scenario, we generate different $\lambda$ values for each time slot during the whole simulation time. Similar to Scenario I, we first show the results of Reactive method (baseline) for different metrics in Table VII. Then, we compare the different methods with respect to the baseline, presented in Table VIII.

Based on the obtained results, the Hybrid Reactive method achieves the highest energy savings, but at the cost of the lowest QoS. Then,PQ+Static MPC, FCFS+Dynamic MPC, PQ+Hyb+MPC, and PQ+EAC+Rea provide better performance than Hybrid Reactive and FCFS+Static MPC, thus highlighting the benefits of proactive queue optimization, endurance-aware cache selection (EAC), and dynamic reliability management, individually or by any combination of two methods, on the QoS. Finally, COCKTAIL outperforms all the methods by reaching the best overall performance.

### C. Corner Cases Analysis for Memory-Intensive and Non-Memory-Intensive Scenarios

In order to evaluate the memory-bounded tasks' behavior and their overheads on the LLC, we build two new experiments for memory-intensive and non-memory-intensive tasks. Based

TABLE VII
RESULTS FOR REACTIVE METHOD UNDER DYNAMIC $\lambda$ USING ALL THE BENCHMARKS

| Energy | Average QoS | Worst-case QoS | Overdue tasks (#) | Overdue time (s) |
|---|---|---|---|---|
| 1.4E9 | 9.6 | 224.2 | 6344 | 1.1E8 |

TABLE VIII
IMPROVEMENTS OF DIFFERENT METHODS IN COMPARISON TO THE BASELINE (REACTIVE) UNDER DYNAMIC $\lambda$ USING ALL THE BENCHMARKS

| | Energy (%) | Avg QoS (%) | WC QoS (%) | Overdue tasks (#) (%) | Overdue time (s) (%) |
|---|---|---|---|---|---|
| Hybrid Reactive | 4 | 18 | 23 | 6 | 27 |
| FCFS+Static MPC | 1 | 28 | 35 | 12 | 40 |
| PQ+Static MPC | 1 | 35 | 35 | 27 | 46 |
| FCFS+Dyn. MPC | 0 | 54 | 64 | 32 | 75 |
| PQ+Hyb+MPC | -1 | 56 | 62 | 43 | 75 |
| PQ+EAC+Rea | 5 | 35 | 36 | 27 | 47 |
| COCKTAIL | 0 | 60 | 64 | 47 | 80 |

on the different benchmarks' read and write accesses to the LLC (as shown in Fig. 7), we select four non-memory-intensive tasks (i.e., exchange2, nab, namd, and povray) and four memory-intensive tasks (i.e., lbm, cactusBSSN, mcf, and omnetpp) for each experiment under the dynamic scenario (i.e., Section VI-B2), respectively. Table IX shows the results for the baseline method (i.e., Reactive) for both memory-intensive and non-memory-intensive scenarios. Then, Table X and XI show the improvements of the different methods compared to the baseline.

According to Table X, the hybrid cache architecture achieves higher energy saving for non-memory-intensive tasks, when compared to the mixture of all tasks' scenario (i.e., Table VIII). The reason is that the energy-efficient RRAM cache is selected more often by the system when a low load (write access) is imposed on the LLC by the tasks. Therefore, the PQ+EAC+Rea method gives the best energy reduction (i.e., 6%), and the COCKTAIL outperforms all the methods by reaching the best overall performance (higher performance with similar energy consumption) due to using the additional power budget obtained by the cache management to increase the cores' frequency.

For memory-intensive tasks, Hybrid Reactive method achieves no performance improvement compared to the baseline, as shown in Table XI. This is due to the usage of

TABLE IX
RESULTS FOR REACTIVE METHOD UNDER DYNAMIC $\lambda$ FOR MEMORY- AND NON-MEMORY-INTENSIVE SCENARIOS

| Memory Intensive | Energy | Average QoS | Worst-case QoS | Overdue tasks (#) | Overdue time (s) |
|---|---|---|---|---|---|
| No | 1.4E9 | 5.9 | 130.5 | 5537 | 2.8E7 |
| Yes | 1.4E9 | 3.9 | 53.6 | 4471 | 2.2E7 |

TABLE X
IMPROVEMENTS OF DIFFERENT METHODS IN COMPARISON TO THE BASELINE (REACTIVE METHOD) UNDER DYNAMIC $\lambda$ FOR THE NON-MEMORY-INTENSIVE SCENARIO

| | Energy (%) | Avg QoS (%) | WC QoS (%) | Overdue tasks (#) (%) | Overdue time (s) (%) |
|---|---|---|---|---|---|
| Hybrid Reactive | 5 | 17 | 27 | 14 | 34 |
| FCFS+Static MPC | 3 | 22 | 32 | 20 | 41 |
| PQ+Static MPC | 3 | 27 | 35 | 25 | 54 |
| FCFS+Dyn. MPC | 1 | 37 | 33 | 39 | 61 |
| PQ+Hyb+MPC | 1 | 38 | 27 | 44 | 66 |
| PQ+EAC+Rea | 6 | 26 | 35 | 24 | 54 |
| COCKTAIL | 1 | 41 | 35 | 47 | 71 |

TABLE XI
IMPROVEMENTS OF DIFFERENT METHODS IN COMPARISON TO THE BASELINE (REACTIVE METHOD) UNDER DYNAMIC $\lambda$ FOR THE MEMORY-INTENSIVE SCENARIO

| | Energy (%) | Avg QoS (%) | WC QoS (%) | Overdue tasks (#) (%) | Overdue time (s) (%) |
|---|---|---|---|---|---|
| Hybrid Reactive | 3 | 0 | 1 | 0 | 1 |
| FCFS+Static MPC | -2 | 7 | 4 | 7 | 11 |
| PQ+Static MPC | -2 | 10 | 4 | 11 | 27 |
| FCFS+Dyn. MPC | -2 | 28 | 31 | 30 | 58 |
| PQ+Hyb+MPC | -3 | 30 | 29 | 34 | 68 |
| PQ+EAC+Rea | 4 | 8 | 5 | 7 | 27 |
| COCKTAIL | -2 | 31 | 31 | 37 | 70 |

SRAM cache, when a high load (write access) is imposed on the LLC. In this case, there is no room (additional power budget) for the cores to increase their frequency. Thus, energy savings only occur during the idle times of the system. Despite the lack of advantage in using the hybrid cache for the memory-intensive tasks, COCKTAIL still outperforms all methods by reaching the best overall performance thanks to the proposed proactive queue optimization and dynamic MPC-based reliability management methods.

### *D. Latency Overhead Analysis for The Different Cache Architectures*

According to Table II, the write latency in the SRAM cache is 7x lower than RRAM, while the read latency in RRAM is 2x lower than SRAM. Therefore, RRAM cache is faster than SRAM when read accesses mainly go to the LLC. In addition, most of the benchmarks have more read accesses than write accesses to the LLC based on the detailed performed profiling (Fig. 7). Hence, this situation gives us an opportunity to compensate the write latency overhead of RRAM during the read access times.

In the proposed endurance-aware cache selection method, the endurance deposit of RRAM quickly decreases when a lot of write accesses imposed on the RRAM. Therefore, the operating cache changes to SRAM for reliability and performance concerns. For the memory-write-intensive benchmarks, in the worst case, the hybrid cache architecture uses the SRAM cache and behaves the same as the SRAM-only architecture. That is, the hybrid cache architecture outperforms the traditional SRAM architecture for the tasks with the higher number of read accesses to the LLC.

In this work, we analyzed the latency overhead of the hybrid and SRAM-only architectures according to the latency presented in Table II and the time periods of using different cache technologies obtained by our proposed cache-selection method in a dynamic scenario (i.e., Section VII-B). The results show that the hybrid cache architecture achieves 9% less latency overhead than the SRAM architecture. This is because, first, the number of read accesses of most workloads to the LLC is higher than their write accesses. Second, during the write-intensive access periods, the operating cache is switched to SRAM for performance and reliability concerns, thus avoiding the large write latency of the RRAM.

For the corner cases analysis on memory-intensive and non-memory-intensive workloads' scenarios (i.e., Section VII-C),

our proposed strategy (i.e., COCKTAIL) achieves 8% and 25% cache latency reduction (i.e., better task execution time) for memory-intensive and non-intensive scenarios, respectively, compared to the SRAM-only architecture.

## VIII. Conclusion

In this paper we have proposed COCKTAIL, a holistic strategy framework to jointly optimize the energy efficiency of multi-core server processors and tasks performance in the HPC context, while guaranteeing system lifetime reliability. In COCKTAIL we have integrated for the first time in literature, a novel proactive queue optimization and endurance-aware cache selection method, together with dynamic MPC-based reliability management policy for server processors with hybrid cache architectures (integrated SRAM and RRAM technologies). We have compared our framework with the state-of-the-art temperature-dependent reliability and energy optimization techniques for HPC systems, and the experimental results have shown that COCKTAIL provides up to 60% QoS improvement. Moreover, our strategy guarantees the design lifetime for the whole system, when running very diverse sets of HPC tasks.

## Acknowledgment

This work has been partially supported by the EC H2020 RECIPE FET-HPC project (No. 801137), the ERC Consolidator Grant COMPUSAPIEN (No. 725657), and the EC H2020 DeepHealth Project (GA No. 825111).

## References

[1] M. Zapater *et al.*, "Leakage-aware cooling management for improving server energy efficiency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2764–2777, Oct 2015.

[2] D. Meisner *et al.*, "PowerNap: eliminating server idle power," *SIGARCH Comput. Archit. News*, vol. 37, no. 1, pp. 205–216, 2009.

[3] S. Chakraborty and H. K. Kapoor, "Analysing the role of last level caches in controlling chip temperature," *IEEE Transactions on Sustainable Computing*, vol. 3, no. 4, pp. 289–305, Oct 2018.

[4] A. Pahlevan *et al.*, "Towards near-threshold server processors," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 7–12.

[5] H. K. Ahn *et al.*, "Evaluation of stt-mram l3 cache in 7nm finfet process," in *International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE, 2018, pp. 1–4.

[6] Y. Gu, "Investigating read/write aggregation to exploit power reduction opportunities using dual supply voltages," Master's thesis, Washington University, St. Louis, Missouri, 2017.

[7] S. Mittal and J. S. Vetter, "AYUSH: Extending lifetime of SRAM-NVM way-based hybrid caches using wear-leveling," in *IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Oct 2015, pp. 112–121.

[8] Y. Chen *et al.*, "Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design," in *DATE*, March 2012, pp. 45–50.

[9] JEDEC, "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122-A*, 2002.

[10] J. Srinivasan *et al.*, "The case for lifetime reliability-aware microprocessors," in *Proceedings. 31st Annual International Symposium on Computer Architecture*, June 2004, pp. 276–287.

[11] M. G. Moghaddam *et al.*, "Investigation of DVFS based dynamic reliability management for chip multiprocessors," in *International Conference on High Performance Computing Simulation (HPCS)*, July 2015, pp. 563–568.

[12] A. Naveh *et al.*, "Power and thermal management in the Intel Core Duo processor," *Intel Technology Journal*, vol. 10, no. 2, 2006.

[13] Z. Lu *et al.*, "Improved thermal management with reliability banking," *IEEE Micro*, vol. 25, no. 6, pp. 40–49, 2005.

[14] S. Yu and P.-Y. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 43–56, 2016.

[15] T. Chien *et al.*, "Write-energy-saving ReRAM-based nonvolatile SRAM with redundant bit-write-aware controller for last-level caches," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, July 2017, pp. 1–6.

[16] M. V. Beigi and G. Memik, "THOR: Thermal-aware optimizations for extending ReRAM lifetime," in *IEEE International Parallel and Distributed Processing Symposium*, 2018, pp. 670–679.

[17] F. Zanini *et al.*, "Multicore thermal management with model predictive control," in *DATE*, Aug 2009, pp. 711–714.

[18] K. R. Basireddy *et al.*, "Workload-aware runtime energy management for HPC systems," in *HPCS*. IEEE, 2018, pp. 292–299.

[19] A. K. Singh *et al.*, "Value and energy aware adaptive resource allocation of soft real-time jobs on many-core HPC data centers," in *International Symposium on Real-Time Distributed Computing*. IEEE, 2016, pp. 190–197.

[20] S. Sharifi *et al.*, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs," in *Asia and South Pacific Design Automation Conference*. IEEE, 2010, pp. 873–878.

[21] S. Isuwa *et al.*, "TEEM: Online thermal-and energy-efficiency management on CPU-GPU MPSoCs," in *DATE*. IEEE, 2019, pp. 438–443.

[22] H. Wang *et al.*, "Hierarchical dynamic thermal management method for high-performance many-core microprocessors," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 1, Aug 2016.

[23] "perf: Linux profiling with performance counters," https://perf.wiki.kernel.org/index.php/Main_Page.

[24] "Running average power limit – rapl," https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl.

[25] A. Iranfar *et al.*, "Enhancing two-phase cooling efficiency through thermal-aware workload mapping for power-hungry servers," in *DATE*, 2019, pp. 66–71.

[26] X. Dong *et al.*, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, July 2012.

[27] A. Sridhar *et al.*, "3D-ICE: A compact thermal model for early-stage design of liquid-cooled ICs," *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 2576–2589, Oct 2014.

[28] https://en.wikichip.org/wiki/intel/microarchitectures/broadwell.

[29] C. Liu *et al.*, "Service reliability in an HC: Considering from the perspective of scheduling with load-dependent machine reliability," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 476–495, 2019.

[30] X. Gao *et al.*, "Investigating security vulnerabilities in a hot data center with reduced cooling redundancy," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[31] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating systems: Three easy pieces*. Arpaci-Dusseau Books LLC, 2018.

[32] A. H. El Bakely and H. A. Hefny, "Using shortest job first scheduling in greencloud computing," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, pp. 348–354, 2015.

[33] Y. Etsion and D. Tsafrir, "A short survey of commercial cluster batch schedulers," School of Computer Science and Engineering, The Hebrew University of Jerusalem, Tech. Rep., 2005.

[34] G. Le *et al.*, "Dynamic resource provisioning and scheduling with deadline constraint in elastic cloud," in *2013 International Conference on Service Sciences (ICSS)*. IEEE, 2013, pp. 113–117.

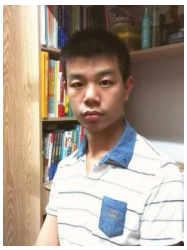
[35] J. Ru and J. Keung, "An empirical investigation on the simulation of priority and shortest-job-first scheduling for cloud-based software systems," in *Australian Software Engineering Conference*. IEEE, 2013, pp. 78–87.

[36] M. Hwang *et al.*, "Least slack time rate first: New scheduling algorithm for multi-processor environment," in *International Conference on Complex, Intelligent and Software Intensive Systems*. IEEE, 2010, pp. 806–811.

[37] C. Jan *et al.*, "A 14 nm SoC platform technology featuring 2nd generation tri-gate transistors, 70 nm gate pitch, 52 nm metal pitch, and 0.0499 um2 SRAM cells, optimized for low power, high performance and high density SoC products," in *Symposium on VLSI Technology (VLSI Technology)*, June 2015, pp. T12–T13.

[38] "SPEC CPU 2017," https://www.spec.org/cpu2017/.

[39] A. Pahlevan *et al.*, "Exploiting cpu-load and data correlations in multi-objective vm placement for geo-distributed data centers," in *DATE*. IEEE, 2016, pp. 1333–1338.



**Darong Huang** is currently a Ph.D. candidate in Electrical Engineering (EE) in the Embedded Systems Laboratory (ESL) at Swiss Federal Institute of Technology Lausanne (EPFL). He received his B.Sc. and M.Sc. degrees in Electrical Engineering from the University of Electronic Science and Technology of China in 2016 and 2019, respectively. His research interests focus on the thermal and reliability management of microprocessors.

**Ali Pahlevan** is a post-doctoral researcher in the Embedded Systems Laboratory (ESL) at Swiss Federal Institute of Technology Lausanne (EPFL). He received his Ph.D. degree in Electrical Engineering (EE) from EPFL in 2019, M.Sc. degree in Computer Engineering from Sharif University of Technology (SUT) in 2012, and B.Sc. degree in Computer Engineering from Ferdowsi University of Mashhad (FUM) in 2010. His research interests focus on system-level energy optimization techniques and reliability in the area of computing systems, datacenters, cloud and green computing. He has published over 15 research papers in top international journals and conferences such as the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), IEEE Transactions on Services Computing (TSC), Springer Handbook of Hardware/Software Codesign, and Design, Automation & Test in Europe Conference & Exhibition (DATE).

**Marina Zapater** is associate professor in the School of Engineering and Management of Vaud (HEIG-VD) at the University of Applied Sciences Western Switzerland (HES-SO) since 2020. She was a post-doctoral research associate in the Embedded System Laboratory (ESL) at the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland, from 2016 to 2020. She received her Ph.D. degree in electronic engineering from Universidad Politécnica de Madrid, Spain, in 2015. Her research interests include thermal, power and performance design and optimization of complex heterogeneous architectures, from embedded edge devices to high-performance computing processors; and energy efficiency in servers and datacenters. In these fields, she has co-authored more than 75 papers in top-notch conferences and journals. She is an IEEE and CEDA member, and CEDA Assistant VP of finance (2019-2020).

**David Atienza** (M'05-SM'13-F'16) is professor of electrical and computer engineering, and head of the Embedded Systems Laboratory (ESL) at the Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland. He received his PhD in computer science and engineering from UCM, Spain, and IMEC, Belgium, in 2005. His research interests include system-level design methodologies for high-performance multi-processor system-on-chip (MPSoC) and low power Internet-of-Things (IoT) systems, including new 2-D/3-D thermal-aware design for MPSoCs and many-core servers, and edge AI architectures for wireless body sensor nodes and smart consumer devices. He is a co-author of more than 350 publications in peer-reviewed international journals and conferences, several book chapters, and twelve licensed U.S. patents in these fields. He has earned several best paper awards and he is (or has been) an Associate Editor of IEEE TC, IEEE D&T, IEEE T-TCAD, IEEE TETC, IEEE T-SUSC and Elsevier Integration, among others. Dr. Atienza received the ICCAD 2020 10-Year Retrospective Most Influential Paper Award, the DAC Under-40 Innovators Award in 2018, IEEE TCCPS Mid-Career Award in 2018, an ERC Consolidator Grant in 2016, the IEEE CEDA Early Career Award in 2013, the ACM SIGDA Outstanding New Faculty Award in 2012, and a Faculty Award from Sun Labs at Oracle in 2011. He served as DATE 2015 Program Chair and DATE 2017 General Chair. He is an IEEE Fellow, an ACM Distinguished Member, and has served as IEEE CEDA President (period 2019-2020).