

Multi-Objective Memory Bandwidth Regulation and Cache Partitioning for Multicore Real-Time Systems

Binqi Sun ✉ 

Technical University of Munich, Germany

Andrea Bastoni ✉ 

Technical University of Munich, Germany

Mirco Theile ✉ 


Technical University of Munich, Germany

Rodolfo Pellizzoni ✉ 

University of Waterloo, Canada

Zhihang Wei ✉ 

Technical University of Munich, Germany

Debayan Roy ✉ 

Technical University of Munich, Germany

Tomasz Kloda ✉ 

LAAS-CNRS, Insa de Toulouse, France

Marco Caccamo ✉ 

Technical University of Munich, Germany

Abstract

Memory bandwidth regulation and cache partitioning are widely used techniques for achieving predictable timing in real-time computing systems. Combined with partitioned scheduling, these methods require careful co-allocation of tasks and resources to cores, as task execution times strongly depend on available allocated resources. To address this challenge, this paper presents a 0-1 linear program for task-resource co-allocation, along with a multi-objective heuristic designed to minimize resource usage while guaranteeing schedulability under a preemptive EDF scheduling policy. Our heuristic employs a multi-layer framework, where an outer layer explores resource allocations using Pareto-pruned search, and an inner layer optimizes task allocation by solving a knapsack problem using dynamic programming. To evaluate the performance of the proposed optimization algorithm, we profile real-world benchmarks on an embedded AMD UltraScale+ ZCU102 platform, with fine-grained resource partitioning enabled by the Jailhouse hypervisor, leveraging cache set partitioning and MemGuard for memory bandwidth regulation. Experiments based on the benchmarking results show that the proposed 0-1 linear program outperforms existing mixed-integer programs by finding more optimal solutions within the same time limit. Moreover, the proposed multi-objective multi-layer heuristic performs consistently better than the state-of-the-art multi-resource-task co-allocation algorithm in terms of schedulability, resource usage, number of non-dominated solutions, and computational efficiency.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Embedded software

Keywords and phrases Multi-objective optimization, memory bandwidth regulation, cache partitioning, partitioned scheduling, real-time systems

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2025.2

Supplementary Material Image: <https://zenodo.org/records/15430297>

Acknowledgements Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

1 Introduction

The advent of multiprocessor systems-on-chip (MPSoC) has transformed the landscape of embedded real-time computing by enabling platforms that combine high performance with energy efficiency. Modern MPSoCs, such as the AMD Ultrascale+ ZCU102 [5] and NVIDIA Orin [50], integrate multiple processing cores and specialized accelerators within a single



© Binqi Sun, Zhihang Wei, Andrea Bastoni, Debayan Roy, Mirco Theile, Tomasz Kloda, Rodolfo Pellizzoni, and Marco Caccamo;

licensed under Creative Commons License CC-BY 4.0

37th Euromicro Conference on Real-Time Systems (ECRTS 2025).

Editor: Renato Mancuso; Article No. 2; pp. 2:1–2:23



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

system, sharing a unified memory hierarchy. These platforms are essential in addressing the increasing demand for complex real-time applications. However, fully utilizing their potential presents significant challenges due to cross-core interference arising from the shared nature of critical resources such as memory interconnects, memory controllers, and caches, which complicates the analysis required for real-time guarantees [19, 22, 29, 36, 37, 41].

To address these challenges, both hardware- and software-based partitioning techniques have been developed to manage the allocation of shared resources. Hardware solutions, such as Intel RDT [32] and Arm MPAM [7], offer monitoring and partitioning capabilities for hardware resources like caches and interconnects to ensure quality of service (QoS) levels. Meanwhile, software-based techniques, including memory bandwidth regulation [68, 71] and cache partitioning [33, 34, 42], have become popular because they offer flexibility across multiple platforms and can be integrated at the hypervisor level. This integration ensures transparency to both operating systems and applications. By controlling the allocation of shared resources, these partitioning techniques improve isolation and predictability, enabling tighter bounds on worst-case execution times (WCETs) and response time analysis for real-time tasks [39, 67]. However, they also introduce new complexities in task and resource co-allocation. Task execution times can be affected significantly by the availability of shared resources, necessitating careful coordination to balance resource usage and task allocation for high system performance and real-time schedulability.

This work addresses the problem of task and resource co-allocation in the context of partitioned scheduling, which was shown to be effective in practice (*e.g.*, [12]). Specifically, we focus on systems employing a preemptive Earliest Deadline First (EDF) scheduling policy. The objective is to minimize memory bandwidth and cache resource usage while ensuring real-time schedulability. We formulate this problem as a multi-objective zero-one linear program (0-1 ILP). Although the ILP can be solved using standard mathematical programming solvers to derive optimal solutions for each single objective separately, it suffers from an exponential time complexity and cannot generate multiple Pareto-optimal solutions within a reasonable time. To address these challenges, we propose a multi-objective resource-task co-allocation heuristic that employs a multi-layer optimization framework. The outer layer explores resource allocations using Pareto-pruned search, and the inner layer optimizes task allocation by solving a knapsack problem with dynamic programming.

To evaluate our approach, we derive *slowdown profiles* for real-world benchmarks from the *San Diego Vision Benchmark Suite (SD-VBS)* [48, 64] and *PARSEC* [11] suites, executed on a real embedded MPSoC platform (AMD UltraScale+ ZCU102). These profiles quantify execution time slowdowns as a function of allocated memory bandwidth and cache partitions (compared to the configuration with full resource availability). We conducted extensive benchmarking on the ZCU102, which features four Cortex-A53 cores sharing a 1 MB last-level cache. Specifically, we configured fine-grained resource partitioning schemes using the Jailhouse hypervisor [45, 56], leveraging cache set partitioning (coloring, *e.g.*, [35]) and MemGuard [68] for bandwidth regulation. Our experiments based on these real-world benchmarks demonstrate that the proposed 0-1 ILPs are solved more efficiently than an existing model based on mixed-integer programming (MIP) [67]. Moreover, our proposed heuristic consistently outperforms the state-of-the-art algorithm in terms of schedulability, resource usage, number of non-dominated solutions, and solving efficiency.

In summary, we make the following contributions:

- We develop a 0-1 ILP for the resource-task co-allocation problem, outperforming the existing mixed-integer program developed in [67] that includes non-binary decision variables and non-linear constraints (Section 3).

- We propose a multi-objective multi-layer optimization heuristic (MMO) with effective Pareto-based pruning strategies and knapsack-based task allocation to minimize memory bandwidth and cache usage simultaneously while ensuring schedulability (Section 4).
- We profile real-world benchmarks from two test suites (*SD-VBS* [64] and *PARSEC* [11]) on an embedded MPSoC hardware to evaluate the impact of the memory bandwidth and cache allocation on task execution slowdowns (Section 5).
- We demonstrate the effectiveness and efficiency of the proposed methods by comparing them with state-of-the-art methods through extensive experimental evaluations using real-world benchmarks. Specifically, MMO identifies up to 62.67% and 50.34% more schedulable task sets than the state-of-the-art algorithm on the *SD-VBS* and *PARSEC* benchmarks, respectively, with significant memory bandwidth and cache usage savings and computation time reduction (Section 6).

2 Related Work

In this section, we give an overview of the related works on task and resource allocation strategies for real-time systems and discuss the differences between our proposed algorithm and the state-of-the-art resource-task co-allocation methods.

2.1 Task Allocation

Mapping tasks statically to individual processors is widely used in industry practice due to its low scheduling overhead [12]. However, since the task allocation problem is NP-hard in the strong sense [23], many approximation methods have been developed for both preemptive [9, 14, 21, 40] and non-preemptive [25, 55] scheduling policies. These methods have also been extended to support parallel task scheduling, including directed acyclic graphs (DAGs) [15, 26, 69] and gang tasks [59, 61, 63], as well as to take into account inter-task interference [70]. On the other hand, exact approaches to the partitioning problem use optimization techniques such as mixed-integer linear programming (MILP) [1, 47]. While MILP formulations can provide exact solutions, their scalability remains a challenge, particularly in systems with a large number of tasks or processors. A detailed discussion on the precise complexity classes of a list of real-time task allocation problems can be found in [23].

2.2 Resource Allocation

Cache and memory bandwidth are two critical resources to be partitioned for achieving timing predictability in real-time systems. A widely adopted *software-based* approach to cache partitioning is *cache coloring*, which has been implemented at both the operating system (OS) [33, 34, 42] and hypervisor levels [35, 65]. In this paper, we rely on a cache-coloring implementation available in the Jailhouse hypervisor [45]. Alternatively, caches can also be partitioned via hardware modifications (*e.g.*, [17]) or by exploiting hardware support such as the Arm DSU [6, 53], which is only available on very recent embedded Arm platforms and notably not yet supported on our Ultrascale+. Similarly to caches, memory bandwidth partitions can be assigned in software leveraging hardware features such as Performance Monitoring Units (PMUs). For example, MemGuard [68] and MemPol [71] propose a per-core memory bandwidth partitioning using PMU-based counters. Hardware modifications to generally improve the predictability of memory accesses have also been proposed (*e.g.*, [24, 31]). Intel RDT [32] supports partitioning of both caches and memory bandwidth and has been used in *e.g.*, [67]. Nonetheless, real-time characteristics of Intel

RDT have been found to be not always effective [57]. Arm MPAM [7] is a recent specification with partitioning capabilities similar to Intel RDT, but to date, no available implementations for COTS platforms exist.

Building on these cache and memory bandwidth partitioning methods, various allocation strategies have been developed to effectively dedicate resources to real-time tasks and improve schedulability. For caches, approaches such as branch-and-bound [3, 4], genetic algorithms [13, 44], and guided-local search [58, 60] have been proposed to optimize how cache partitions are assigned to real-time workloads. Similar efforts exist for memory bandwidth allocation. Aghilinasab *et al.* [2] present a dynamic scheme that monitors and reallocates memory bandwidth between real-time and best-effort tasks, adapting to runtime variations. Park *et al.* [52] further propose a coordinated approach for LLC and memory bandwidth partitioning, targeting workload fairness rather than hard timing guarantees.

2.3 Task and Cache Co-Allocation

Beyond independent allocation strategies for real-time tasks and resources, the co-allocation of tasks and cache partitions has been explored to further enhance real-time schedulability. Under preemptive EDF scheduling, Chisholm *et al.* [16] introduce MC², a linear programming-based optimization framework for mixed-criticality multicore real-time systems. Kim and Rajkumar [33] develop a cache management scheme for cache-to-task allocation and later proposed a cache-aware task allocation algorithm tailored for virtual machine design. The task and cache allocation under non-preemptive scheduling introduces additional challenges due to blocking effects, making task utilization an insufficient sole indicator for schedulability. Berna and Puaud [10] propose a period-driven task and cache partitioning algorithm under non-preemptive EDF scheduling, prioritizing task period compatibility as the primary partitioning criterion. Paolieri *et al.* [51] introduce IA³, an interference-aware allocation algorithm focusing on WCET sensitivity. Sun *et al.* [62] develop a search-based algorithm leveraging a first-fit heuristic for task allocation and propose two heuristic variants considering task period compatibility and cache sensitivity as the task ordering criteria. However, these works do not consider memory bandwidth allocation. Ignoring contention on the shared memory bus can compromise system predictability while assuming uniform memory bandwidth partitioning limits the flexibility needed to optimize schedulability effectively.

2.4 Task and Multi-Resource Co-Allocation

Recent studies [27, 49, 66, 67] have also explored the problem of task and multi-resource co-allocation. Meng *et al.* [66] propose a multi-resource allocation framework for real-time multicore virtualization, incorporating techniques to mitigate abstraction overhead. Nie *et al.* [49] investigate federated scheduling for parallel tasks, where each task is assigned a set of cores along with cache and memory bandwidth partitions while ensuring schedulability conditions. Gifford *et al.* [27] employ a worst-fit bin-packing approach to allocate soft real-time tasks to cores, dynamically adjusting cache and memory bandwidth partitions based on task deadlines.

The most *closely related work* to ours is [67], which also addresses the co-allocation of tasks and resources, including memory bandwidth and cache partitions. Their algorithm, CaM, represents the current state-of-the-art and outperforms other previous methods. It employs a k-means algorithm to classify tasks into different clusters, followed by a task-cluster-to-core allocation heuristic based on first-fit bin-packing and a resource-to-core allocation heuristic based on resource utility. Our method, MMO, improves upon CaM in three key ways:

- The multi-layer search strategy of MMO provides a more thorough exploration of task and resource co-allocation possibilities.
- The inner-layer task allocation in MMO is formulated as a knapsack problem, enabling the use of dynamic programming to achieve optimal task assignments that maximize core utilization.
- MMO is a multi-objective heuristic that simultaneously optimizes the usage of multiple resources, yielding multiple non-dominated solutions, while CaM can only produce a single solution.

In Section 6, we present extensive experimental evaluations to compare the performance of MMO with CaM in terms of schedulability, resource usage, and computation efficiency.

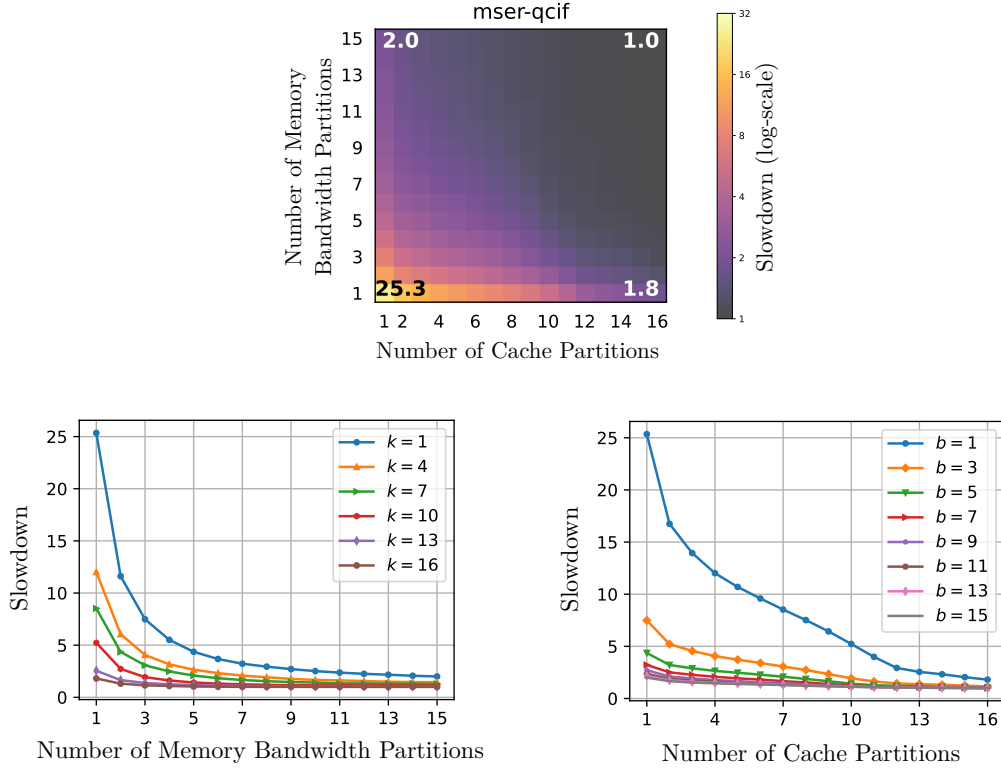
3 System Model and Mathematical Formulation

3.1 System Model

We consider a set of N real-time tasks τ running on M cores that share a last-level cache (LLC) and memory bus. The memory bandwidth and the LLC are divided into B and K uniform partitions, respectively. Each task $\tau_i \in \tau$ generates an infinite sequence of sporadic job instances, with a minimum inter-arrival time of T_i , and each job must complete its execution before the next instance arrives (*i.e.*, has an implicit deadline). Tasks are scheduled by a partitioned scheduling policy, where the task set is divided into M subsets, each statically assigned to a specific core (*i.e.*, jobs of a task remain on their assigned core without migrating at runtime). Within each core, tasks are scheduled by a preemptive Earliest Deadline First (EDF) scheduler. Partitions of shared resources (*i.e.*, memory bandwidth and LLC) are also statically assigned to each core, providing inter-core isolation to reduce interference. The worst-case execution time (WCET) of a task τ_i , given b memory bandwidth and k cache partitions, is characterized by $C_i(b, k)$. We assume that the cache-related preemption delays (CRPD) are accounted for in the task WCETs. The reference WCET, $\hat{C}_i = C_i(K, B)$, gives the WCET when the task has access to full memory bandwidth and LLC. Task utilization is defined as $U(\tau_i, b, k) = C_i(b, k)/T_i$, with $\hat{U}(\tau_i) = \hat{C}_i/T_i$ as the reference utilization. Under the EDF scheduling policy [38], a subset of tasks executing on the same core, denoted as τ' , is schedulable if and only if the sum of their utilizations does not exceed 1 (*i.e.*, $\sum_{\tau_i \in \tau'} U(\tau_i, b, k) \leq 1$), given b memory bandwidth and k cache partitions allocated to the core. The slowdown of a task under b memory bandwidth and k cache allocation is defined as the ratio of its WCET compared to its reference WCET, *i.e.*, $C_i(b, k)/\hat{C}_i$.

The slowdown of an example benchmark, `mser-qcif` from the *SD-VBS* test suite, is visualized in Figure 1. The upper inset of the figure presents the slowdown profile as a heat map for various numbers of b memory bandwidth and k cache partitions, with the numbers at the corners representing the slowdowns under extreme resource allocations. The bottom insets explicitly visualize the slowdown trends when increasing one of the controlled dimensions (b or k) while keeping the other constant. The details of the profiling experiments and more benchmark profiles are presented in Section 5.

A summary of the notations used in the paper is given in Table 1.



■ **Figure 1** Slowdown profile (relative to unrestricted execution) of the `mser-qcif` benchmark with variable memory bandwidth b and cache partition k allocations. The upper inset presents the slowdown as a heatmap where both dimensions vary; the bottom insets present the resource sensitivity of each dimension separately.

■ **Table 1** Summary of notations.

Notations	Description
System model	
τ	a set of N tasks $\tau = \{\tau_1, \tau_2, \dots, \tau_N\}$;
M, B, K	total number of cores, memory bandwidth, and cache partitions;
$U(\tau_i, b, k)$	τ_i 's utilization with b memory bandwidth and k cache partitions;
$\hat{U}(\tau_i)$	τ_i 's reference utilization;
Mathematical formulation	
x_{im}	1, if τ_i is assigned to core m ; 0, otherwise
y_{bm}	1, if b memory bandwidth partitions are assigned to core m ; 0, otherwise
z_{km}	1, if k cache partitions are assigned to core m ; 0, otherwise
α_{ibkm}	auxiliary variables: $\alpha_{ibkm} = x_{im} \wedge y_{bm} \wedge z_{km}$
Optimization heuristic	
ω, ω'	a (new) partial solution;
$\omega, \vec{\tau}, \omega, \vec{b}, \omega, \vec{k}$	ω 's task, memory bandwidth, and cache allocation vectors;
$\omega, \bar{\tau}, \omega, \bar{b}, \omega, \bar{k}$	ω 's remaining tasks, memory bandwidth, and cache partitions;
Ω, Ω'	a (new) set of partial solutions;
Ω^*	a set of complete non-dominated solutions;

3.2 Mathematical Formulation

As shown in Figure 1, the task execution times can be significantly influenced by the resources (*i.e.*, memory bandwidth and cache partitions) allocated to the core on which they execute. Therefore, ensuring schedulability on all cores requires a careful co-allocation of tasks and resources such that no task misses its deadline. Moreover, simply meeting deadlines is often insufficient in embedded systems, where resources are inherently limited and frequently shared among multiple applications. Because these constrained platforms must also address cost and power limitations, minimizing resource usage becomes a critical objective.

By reducing the resources reserved for real-time tasks to only what is strictly necessary, system designers can allocate remaining resources to best-effort tasks or other value-added services without jeopardizing the timing guarantees of critical workloads. In some cases, disabling unused resources leads to lower power consumption and reduced heat dissipation, thus extending battery life and enhancing overall system reliability [8, 18, 46]. This balance between schedulability and resource efficiency cannot be captured by a feasibility formulation focusing solely on meeting deadlines.

A multi-objective optimization approach naturally addresses these competing demands by finding configurations that simultaneously guarantee real-time performance and minimize multiple resource usage. In this way, system designers gain a set of Pareto-optimal (non-dominated) solutions—each reflecting a distinct trade-off between the usage of different resources. Such flexibility is highly beneficial in the diverse and evolving landscape of real-time applications, where a single, one-size-fits-all solution often falls short.

Based on these considerations, we formulate a multi-objective optimization problem as follows with notations in Table 1.

$$\min. \quad \sum_{m=1}^M \sum_{b=1}^B y_{bm} \cdot b, \quad (1)$$

$$\min. \quad \sum_{m=1}^M \sum_{k=1}^K z_{km} \cdot k, \quad (2)$$

$$\text{s.t.} \quad \sum_{m=1}^M x_{im} = 1, \quad \forall i \in [1, N], \quad (3)$$

$$\sum_{b=1}^B y_{bm} = 1, \quad \forall m \in [1, M], \quad (4)$$

$$\sum_{k=1}^K z_{km} = 1, \quad \forall m \in [1, M], \quad (5)$$

$$\sum_{m=1}^M \sum_{b=1}^B y_{bm} \cdot b \leq B, \quad (6)$$

$$\sum_{m=1}^M \sum_{k=1}^K z_{km} \cdot k \leq K, \quad (7)$$

$$3 \cdot \alpha_{ibkm} \leq x_{im} + y_{bm} + z_{km}, \quad \forall i \in [1, N], b \in [1, B], k \in [1, K], m \in [1, M], \quad (8)$$

$$\alpha_{ibkm} \geq x_{im} + y_{bm} + z_{km} - 2, \quad \forall i \in [1, N], b \in [1, B], k \in [1, K], m \in [1, M], \quad (9)$$

$$\sum_{i=1}^N \sum_{b=1}^B \sum_{k=1}^K \alpha_{ibkm} \cdot U(\tau_i, b, k) \leq 1, \quad \forall m \in [1, M], \quad (10)$$

where (1) and (2) define multi-objective functions to minimize memory bandwidth and cache usage, respectively. Constraints (3) ensure each task is assigned to exactly one core, while (4) and (5) guarantee each core is assigned a specific number of memory bandwidth and cache partitions, respectively. Constraints (6) and (7) enforce the total allocated memory bandwidth and cache partitions do not exceed their capacities. Constraints (8) and (9) provide linear equivalents for the conjunction $x_{im} \wedge y_{bm} \wedge z_{km}$, where $\alpha_{ibkm} = 1$ if task i is assigned to core m , which allocates b memory bandwidth and k cache partitions. Constraints (10) ensure that the assigned tasks are schedulable (*i.e.*, satisfy the utilization bound) on each core.

The resulting model is a zero-one linear program (0-1 ILP) since it contains only binary variables and linear constraints. In Section 6.2, we will show that our 0-1 ILP model is solved more efficiently than the mixed-integer program (MIP) developed by [67], which includes non-binary decision variables and non-linear constraints.

3.3 Optimization Challenges

The 0-1 ILP can be solved using a standard mathematical programming solver to optimize every single objective, *i.e.*, memory bandwidth (1) or cache partitions (2), separately. However, there remain two significant challenges to be addressed:

- **Scalability.** Even though the 0-1 ILP only contains linear constraints, the solution space grows exponentially with respect to the number of tasks, cores, and resource partitions, making it challenging for solvers to find feasible solutions within a reasonable time. This limits its practicality in real-world system design.
- **Multi-objective optimization.** The formulation requires minimizing the usage of multiple resources simultaneously. While multi-objective optimization problems can be transformed into multiple single-objective problems using a weighted combination of objectives, the resulting solutions do not necessarily correspond to the pre-determined weights, failing to reflect the desired trade-off [20]. A more effective approach is to develop a multi-objective optimization algorithm that generates a set of Pareto-optimal solutions, where no solution dominates another in terms of all objectives. This allows system designers to choose the most suitable allocation from the Pareto set based on their specific preferences and trade-offs.

Therefore, we propose a new heuristic to address these challenges.

4 Multi-Objective Multi-Layer Optimization

This section introduces MMO, a multi-objective, multi-layer optimization heuristic for task-resource co-allocation. As discussed in Section 3, task and resource allocations are inherently interdependent, necessitating a thorough exploration of the joint design space, which is known to be NP-hard [67]. To address this challenge, we propose a multi-layer design space exploration heuristic, where the outer layer explores different resource allocations in a core-by-core manner, and the inner layer determines the task allocation under the resource allocation specified by the outer layer. To deal with the complexity of the large design space, we (i) propose Pareto-optimality-based pruning techniques in the outer layer to reduce the exploration space by systematically eliminating suboptimal resource allocations and (ii) model the inner-layer task allocation as a knapsack problem that can be efficiently solved by dynamic programming (DP).

4.1 Outer Layer: Resource Allocation Based on Pareto-Pruned Search

Algorithm 1 describes the procedures in the outer layer. The algorithm takes the system specifications as input, including the set of real-time tasks τ , the number of cores M , the number of memory bandwidth partitions B , the number of cache partitions K , and an algorithmic parameter γ used for controlling the precision of the dynamic programming in the inner layer. The output of the algorithm is a set of Pareto-optimal solutions within the explored search space, where no solution dominates another in terms of all objective values.

The algorithm employs a breadth-first search strategy to explore resource and task allocations on a core-by-core basis, with the maximum search depth equal to the total number of cores M . The search begins by initializing an empty set of complete solutions Ω^*

Algorithm 1 Multi-Objective Search with Pareto-Based Pruning (Outer Layer).

Input: τ, M, B, K, γ ;
Output: Ω^* : The non-dominated solutions;

```

1  $\Omega^* \leftarrow \emptyset$ ;
2 Initialize  $\omega_0$  with empty allocation vectors:  $\omega_0.\vec{\tau}, \omega_0.\vec{b}, \omega_0.\vec{k}$ , and full remaining
   tasks and resources:  $\omega_0.\bar{\tau} \leftarrow \tau, \omega_0.\bar{B} \leftarrow B, \omega_0.\bar{K} \leftarrow K$ ;
3  $\Omega \leftarrow \{\omega_0\}$ ;
4 for  $m \leftarrow 1$  to  $M$  do
5    $\Omega' \leftarrow \emptyset$ ;
6   for  $\omega \in \Omega$  do
7     for  $b \leftarrow 1$  to  $\omega.\bar{B}$  do
8       for  $k \leftarrow 1$  to  $\omega.\bar{K}$  do
9         if  $(\omega.\bar{B}-b, \omega.\bar{K}-k)$  not dominated by  $\Omega^*$  then // Rule 1
10           $\omega' \leftarrow \text{TaskAlloc}(\omega.\bar{\tau}, b, k, \gamma)$ ;
11          if  $\omega'.\bar{\tau} = \emptyset$  then
12             $\Omega^* \leftarrow \Omega^* \cup \{\omega'\}$ ;
13             $\text{PruneCompleteSols}(\Omega^*)$ ; // Rule 1
14          else
15            if  $\text{IsFeasible}(\omega')$  then // Rule 2
16               $\Omega' \leftarrow \Omega' \cup \{\omega'\}$ ;
17               $\text{PrunePartialSols}(\Omega')$ ; // Rule 3
18    $\Omega \leftarrow \Omega'$ ;
19 return  $\Omega^*$ ;
```

and a single initial partial solution ω_0 (lines 1-2). This initial solution has empty task and resource allocations, denoted by $\omega_0.\vec{\tau}, \omega_0.\vec{b}$, and $\omega_0.\vec{k}$, and retains all system resources, represented by $\omega_0.\bar{B}$ and $\omega_0.\bar{K}$ (lines 1-2). The set of current partial solutions Ω is initialized with the initial solution ω_0 (line 3).

During each search iteration (lines 4-18), indexed by m , each partial solution in Ω is extended into multiple new partial solutions, which include the task and resource allocations for the corresponding core m . Resource allocation is generated by enumerating all possible combinations of memory bandwidth and cache partitions, ranging from one partition to the number of remaining available partitions in the system (lines 7-8). The resource allocation of each extended partial solution is checked for dominance against the complete solutions in Ω^* using Pruning Rule 1:

► **Pruning Rule 1** (Dominance against complete solutions). *A solution ω_1 is not dominated by a complete solution ω_2 if and only if*

$$(\omega_1.\bar{B} > \omega_2.\bar{B}) \vee (\omega_1.\bar{K} > \omega_2.\bar{K}). \quad (11)$$

If the resource usage of an extended partial solution is dominated by any complete solution in Ω^* , it is pruned from further exploration, skipping the generation of its task allocation (line 9). For non-dominated solutions, the inner layer (Algorithm 2) is invoked to determine the task allocation under the specified resource allocation, resulting in an extended partial solution ω' (line 10).

Each ω' is then checked for completeness. If no tasks remain unassigned in ω' , it is added to the complete solution set Ω^* (line 11 - 12). Then, Ω^* is updated to remove any solutions dominated by ω' (line 13). If there still exist remaining tasks to be assigned, the extended partial solution is checked for feasibility (line 15) using Pruning Rule 2:

► **Pruning Rule 2 (Feasibility).** *A partial solution ω is infeasible if*

$$\sum_{\tau_i \in \omega.\bar{\tau}} U(\tau_i, \omega.\bar{B}, \omega.\bar{K}) > \bar{M}, \quad (12)$$

where $\bar{M} = M - m$ is the number of remaining cores at search depth m .

This rule gives a sufficient condition for infeasibility. Specifically, the total utilization of the remaining tasks, calculated under the assumption that all remaining resources are fully allocated to every core, represents a lower bound on the total remaining task utilization. If this lower bound exceeds the number of remaining cores, it indicates that the solution cannot satisfy the scheduling constraint and should be pruned from further exploration. It is important to note that this condition is sufficient but not necessary. In other words, while solutions that fail this check are guaranteed to be infeasible, solutions that pass the check may still turn out to be infeasible upon further exploration.

If a solution fails the feasibility test, it will be pruned from further exploration. Otherwise, the solution is added to the new partial solution set Ω' (line 16). To maintain efficiency, we further remove dominated partial solutions in Ω' using Pruning Rule 3:

► **Pruning Rule 3 (Dominance between partial solutions).** *A partial solution ω_1 is not dominated by another partial solution ω_2 if and only if*

$$(\omega_1.\bar{B} > \omega_2.\bar{B}) \vee (\omega_1.\bar{K} > \omega_2.\bar{K}) \vee (\omega_1.\bar{U} < \omega_2.\bar{U}), \quad (13)$$

where $\omega_1.\bar{U}$ represents the remaining scheduling demand, defined as the sum of reference utilizations of the remaining tasks, i.e., $\omega_1.\bar{U} = \sum_{\tau_i \in \omega_1.\bar{\tau}} \hat{U}(\tau_i)$.

This rule intuitively prioritizes solutions with a lower remaining scheduling demand and higher remaining resources, which are more likely to be schedulable. We note that the reference utilization is not an exact indicator of scheduling demand, and thus, a dominated partial solution by this rule is not guaranteed to be worse than the non-dominated ones. However, this pruning rule effectively limits the number of partial solutions generated in each search iteration (at most $B \cdot K$, since in the worst case, for each combination of remaining memory bandwidth and cache partitions, only the partial solution with the lowest scheduling demand is kept). Evaluation results in Section 6.2 demonstrate the effectiveness of this strategy.

The algorithm terminates when all possible resource allocations have been explored. The final set of complete solutions Ω^* contains all Pareto-optimal (non-dominated) solutions found by the algorithm.

4.2 Inner Layer: Task Allocation Based on Dynamic Programming

The inner layer determines the task assignment to a core with the resource allocation specified by the outer layer. The key idea is to assign tasks to the core to maximize the scheduling demand of assigned tasks τ' while satisfying the schedulability constraint, given the allocated resources (b memory bandwidth and k cache partitions). The scheduling demand of a task τ_i is measured by its reference utilization $\hat{U}(\tau_i)$, and the schedulability constraint is expressed as $\sum_{\tau_i \in \tau'} U(\tau_i, b, k) \leq 1$. This problem can be modeled as a 0-1 knapsack problem [43], where:

Algorithm 2 Dynamic Programming Based Task Allocation (Inner Layer).

Input: ω, b, k, γ ;
Output: ω' : The extended solution;
 1 $\omega'.\vec{\tau} \leftarrow \omega.\vec{\tau}$; Append b to $\omega'.\vec{b}$; Append k to $\omega'.\vec{k}$; $n \leftarrow |\omega.\vec{\tau}|$;
 2 $\omega'.\bar{b} \leftarrow \omega.\bar{b} - b$; $\omega'.\bar{k} \leftarrow \omega.\bar{k} - k$;
 3 Initialize a 2D array $dp[0 \dots n, 0 \dots \gamma]$ with zeros;
 4 **for** $i = 1$ **to** n **do**
 5 $u \leftarrow \lceil U(\omega.\bar{\tau}_{i-1}, b, k) \cdot \gamma \rceil$;
 6 $\hat{u} \leftarrow \hat{U}(\omega.\bar{\tau}_{i-1})$;
 7 **for** $j = 0$ **to** γ **do**
 8 **if** $u > j$ **then**
 9 $dp[i, j] \leftarrow dp[i-1, j]$;
 10 **else**
 11 $dp[i, j] \leftarrow \max(dp[i-1, j], dp[i-1, j-u] + \hat{u})$;
 12 $\tau' \leftarrow \emptyset$; $j \leftarrow \gamma$;
 13 **for** $i = n$ **to** 1 **do**
 14 **if** $dp[i, j] \neq dp[i-1, j]$ **then**
 15 $\tau' \leftarrow \tau' \cup \{\omega.\bar{\tau}_{i-1}\}$;
 16 $j \leftarrow j - \lceil U(\omega.\bar{\tau}_{i-1}, b, k) \cdot \gamma \rceil$;
 17 Append τ' to $\omega'.\vec{\tau}$;
 18 **return** ω' ;

- The remaining tasks to be assigned $\omega.\bar{\tau}$ are regarded as items to be packed into the knapsack;
- The value of each item is the task's reference utilization $\hat{U}(\tau_i)$;
- The size of each item is the task's utilization under the current resource allocation $U(\tau_i, b, k)$;
- The knapsack's capacity corresponds to the utilization bound of the EDF scheduling policy, which is 1.

The knapsack problem is solved by a dynamic programming (DP) approach [43], as outlined in Algorithm 2.

The algorithm takes the partial solution to be extended ω , along with the allocated memory bandwidth b and cache partitions k , and a scaling parameter γ . Since dynamic programming requires integer item sizes, the scaling parameter γ is introduced to inflate fractional task utilizations and the utilization bound to integer values. In particular, each task's utilization is scaled and rounded up using the ceiling function to ensure a safe utilization bound (line 5). At the beginning of the algorithm, we initialize the new partial solution ω' by appending allocated resources (b and k) to the corresponding resource allocation vectors and recalculating the remaining resources available in the system (lines 1-2). Then, a 2D DP array $dp[0 \dots n, 0 \dots \gamma]$ is initialized, where n is the number of unassigned tasks. The value in $dp[i, j]$ stores the maximum total reference utilization achievable for the first i tasks with a scaled utilization bound j . Then, the DP array is updated iteratively. If the task's scaled utilization exceeds the current bound j , the task is excluded (line 8); otherwise, the maximum reference utilization is determined by taking the better of two options: excluding the task or including it and adding its reference utilization (line 9). After constructing the DP array,

the selected tasks are retrieved by backtracking through the array (lines 13–17). Finally, the algorithm appends the assigned set of tasks to the solution’s task allocation vector and returns the extended partial solution ω' (lines 17–18).

4.3 Time Complexity

In the outer layer, the maximum number of search iterations is given by M , and the number of partial solutions to be extended in each iteration is bounded by $B \cdot K$ under the dominance relationship defined in Pruning Rule 3. In the worst case, all possible resource allocations are considered for each partial solution, resulting in $\mathcal{O}(B^2 K^2)$ invocations of the inner layer to determine task allocations. Since the DP approach has a time complexity of $\mathcal{O}(N\gamma)$, the overall complexity of the algorithm is $\mathcal{O}(NB^2 K^2 M\gamma)$. In this paper, we did preliminary experiments with different γ values ranging from 1000 to 10000. As the results are consistent across these values, we only present the results with $\gamma = 1000$.

5 Real-World Benchmarking with Resource Partitioning

5.1 Resource Partitioning Configuration

We profile real-world benchmarks on the embedded AMD UltraScale+ ZCU102 [5] platform, equipped with four Cortex-A53 cores. Each core has a private 32 KB, 4-way set-associative L1 data cache¹ and shares a 1 MB, 16-way set-associative last-level cache (LLC). We use benchmarks from two test suites: 43 benchmarks from *SD-VBS* [48, 64] and 20 benchmarks from *PARSEC* [11].² Both test suites have also been previously adopted in similar problem settings [62, 67]. Notably, *PARSEC* was ported to the ARM 64-bit architecture to ensure compatibility with our platform.

The profiles were generated by systematically limiting the availability of the LLC and memory bandwidth resources allocated to the benchmarks. This was achieved transparently through the Jailhouse hypervisor [45, 56], which provides mechanisms for cache set partitioning and memory bandwidth control.³ Specifically, the hypervisor employs coloring techniques [35] for cache set partitioning and integrates the MemGuard [54, 68] to regulate bandwidth allocation. The benchmarks were executed on a single core of the ZCU102 under a virtualized Linux operating system (Kernel 6.1.30). Slowdown profiles were computed relative to configurations with full hardware resource availability (*i.e.*, the Jailhouse configuration gives full hardware resources to the virtual machine running Linux). Each configuration was repeated 10 times, with a total runtime cap of 10 minutes per benchmark.

The granularity of cache set partitioning depends on the architecture of ZCU102. With a 1 MB, 16-way set associative L2 and 64-byte cache lines (6 offset bits), the cache has 1024 sets (10 index bits). Given a 4 KB page size, which aligns to the last 12 address bits, four bits in the index range are available for partitioning. This allows for a maximum of 16 partitions, each corresponding to 64 KB of the L2 cache. While this scheme offers high granularity, it can lead to underutilization of the L1 data cache due to its associativity, as up to half of the L1 cache might remain unused in certain configurations. To evaluate this trade-off,

¹ Each core also has a private 2-way set-associative L1 instruction cache, but this paper focuses on the data cache.

² We do not use all the benchmarks from the test suites because the remaining benchmarks either fail to be built or take too long to execute on our platform.

³ We use the publicly available Minervasys (<https://github.com/Minervasys/Jailhouse>) fork of Jailhouse that implements cache and bandwidth partitioning.

we chose to enable all possible coloring configurations, prioritizing higher granularity over limiting the maximum number of partitions to avoid L1 underutilization. We experimentally validated the impact of this choice by running a dedicated set of experiments on *SD-VBS* benchmarks with various cache-partitioning configurations, including those that colored the L1 cache. For each configuration, we measured execution time while keeping the number of cache partitions constant. The results showed that across all benchmarks, the variation in execution time between configurations was under 5%. For larger data sets, such as *VGA*, this variation dropped below 1%, indicating that the penalty of L1 underutilization is minimal in practice. Therefore, we use 16 cache partitions each of 64 KB size of L2 in our benchmarking.

Bandwidth partitioning was similarly controlled using MemGuard in the Jailhouse hypervisor. We validated a MemGuard period of 1 ms by evaluating its overheads and benchmark execution times. This value aligns with the findings from previous studies, *e.g.*, [68, 71]. To establish the maximum usable budget, we increased the allocated bandwidth in steps of 64 MB/s until the execution time for each benchmark stabilized, with variations below 5%. This approach yielded a total of 15 distinct equally-sized bandwidth partitions, with a maximum guaranteed bandwidth of 960 MB/s. We note that the approach and the obtained bandwidth values are consistent with prior works, *e.g.*, [67, 68, 71].

5.2 Profiling Results

Figure 2 reports the obtained slowdown profiles (relative to the execution without partitioning) for representative benchmarks from the *SD-VBS* and *PARSEC* test suites.⁴ In the figures, each column corresponds to the slowdowns of a single benchmark, while different rows within the same column show results for varying input sizes. The color scale (logarithmic) indicates the slowdown compared to execution without any partitioning: lighter shades denote more significant slowdowns, whereas darker shades represent near-ideal performance. The benchmarks have been chosen to illustrate representative slowdown trends, but we note that our evaluation presented in Section 6 is based on the full set of benchmarks, demonstrating how our co-allocation strategies handle both extreme and intermediate resource sensitivities.

From the *SD-VBS* suite (Figure 2a), we observe that the **disparity** and **mser** benchmarks are highly sensitive to changes in both memory and cache allocations. When these resources are scarce, slowdowns exceeding 20× can occur, but they decrease steadily as additional resources are assigned. Notably, **disparity-vga** and **mser-vga** show a significant slowdown of 11.3× and 8.7× respectively when only minimal memory bandwidth is available. The **stich** benchmark exhibits less uniform behavior. For most inputs, its performance remains stable across a wide range of cache and memory bandwidth settings. However, the **stich-vga** input is a notable exception: increasing the number of cache partitions sharply reduces the slowdown from 34.3× to 1.2×. **sift** is generally less sensitive to memory bandwidth and mostly insensitive to the number of cache partitions (except for $k = 1$ with inputs **vga** and **cif**, where the workload cannot finish execution within the allotted time). In fact, with only one memory bandwidth partition assigned, there is a minor slowdown difference between one cache partition and a fully allocated cache, and it is sufficient to assign a few additional memory bandwidth partitions to bring the slowdown to 1.0.

⁴ The complete profiling results are available in the supplemental material of the submission: <https://zenodo.org/records/15430297>.

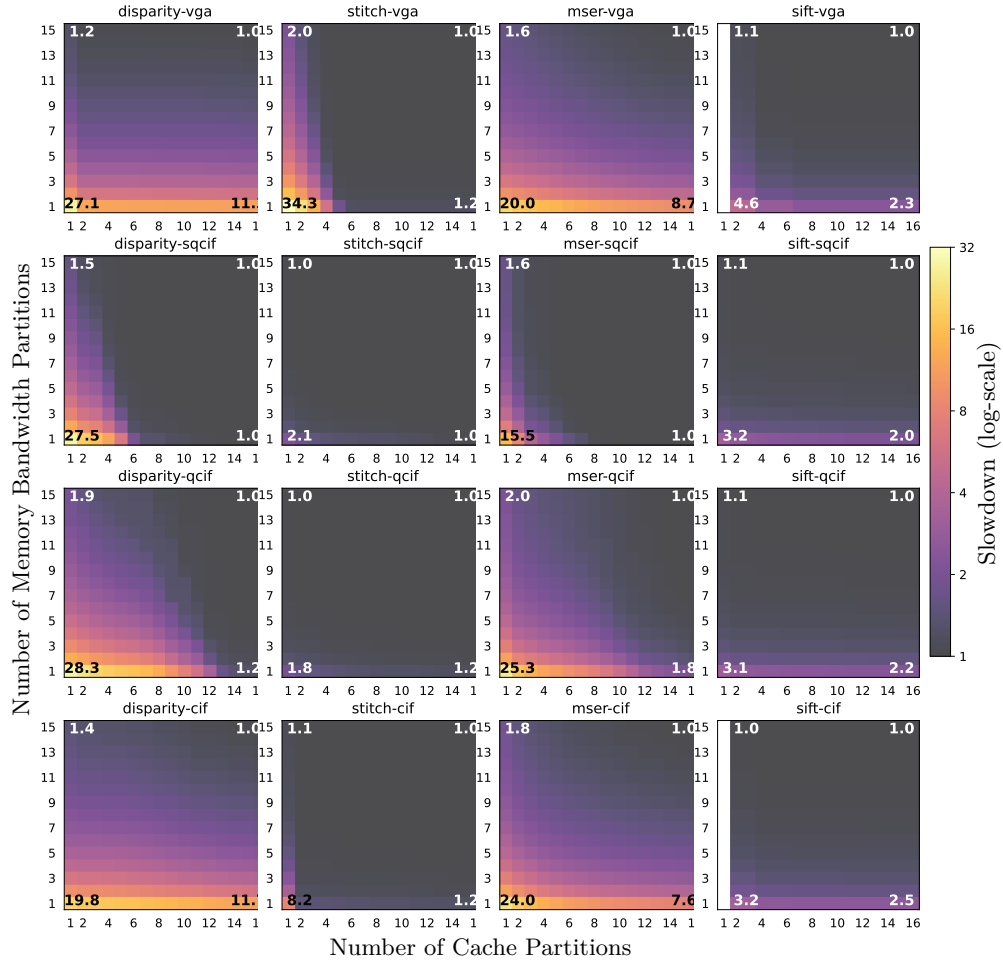
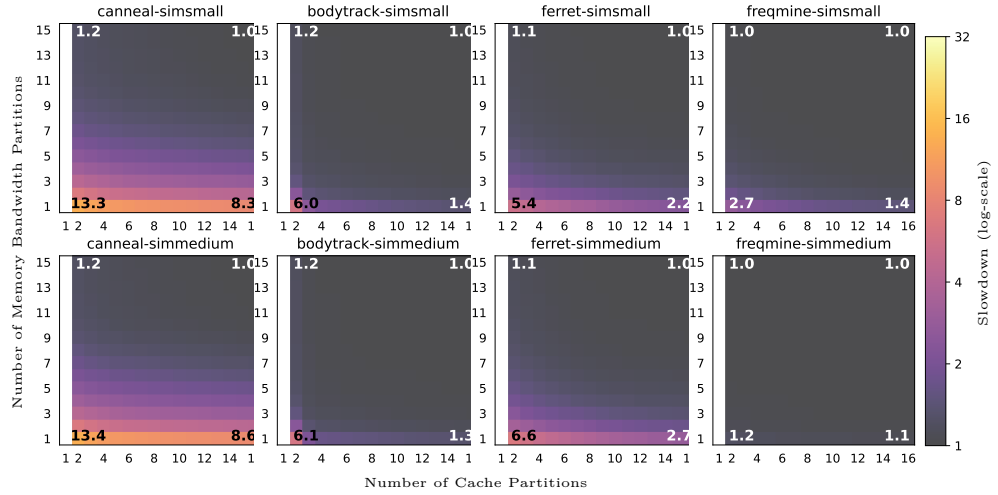
(a) Representative slowdown profiles for the *SD-VBS* test suite.(b) Representative slowdown profiles for the *PARSEC* test suite.

Figure 2 Slowdown profiles for representative benchmarks from different test suites. Columns present the results for one benchmark for different inputs (rows). Numerical slowdown values for the extreme configurations are indicated in the corners of each plot.

Compared to *SD-VBS*, the benchmarks from the *PARSEC* suite (in Figure 2b) show comparatively milder slowdowns overall, typically within the 1–13 \times range. Notably, in all *PARSEC* benchmarks, one single cache partition is not sufficient for the workload to be completed within the allotted time. This is reflected in Figure 2b, where results start from a minimum of two cache partitions. Among these benchmarks, *canneal* reaches its highest slowdown of around 13 \times when both cache and memory bandwidth are minimized. *bodytrack* and *ferret* are moderately sensitive to resource allocations, showing gradual improvement as more resources become available. *freqmine* is among the least sensitive in this set, seldom exceeding a 2–3 \times slowdown even under constrained conditions.

6 Performance Evaluation

6.1 Evaluation Setup

6.1.1 Task Sets

We evaluate the proposed algorithm using real-time task sets generated based on the benchmarks profiled in Section 5. The total number of cores and resources available in the system is set according to the real-world embedded hardware used in our profiling, *i.e.*, $M = 4$, $B = 15$, and $K = 16$. We generate task sets with size $N \in \{20, 40, 60\}$ and consider different task set reference utilizations $\hat{U} = \sum_{\tau_i \in \tau} \hat{U}(\tau_i)$ ranging from 1.0 to M , with a step of 0.1. For each combination of N and \hat{U} , we generate 100 random task sets per benchmark suite, resulting in a total of 18,600 task sets. For each task set, we use the *DRS* generator [28] to assign task reference utilizations, ensuring their sum matches the target \hat{U} . For each task in a task set, a slowdown profile is randomly sampled from the benchmark programs and used to calculate the task utilizations under different resource allocations.

6.1.2 Baseline Algorithms and Implementation

We compare the proposed MMO framework with the following baselines:

- CaM [67]: the state-of-the-art heuristic that addresses the same task-resource co-allocation problem;
- ILP-B, ILP-K: the single-objective 0-1 ILPs to minimize memory bandwidth and cache usage, respectively, presented in Section 3;
- MIP-B, MIP-K: the single-objective mixed-integer programs (MIPs) to minimize memory bandwidth and cache usage, respectively, developed in the extended version⁵ of [67].

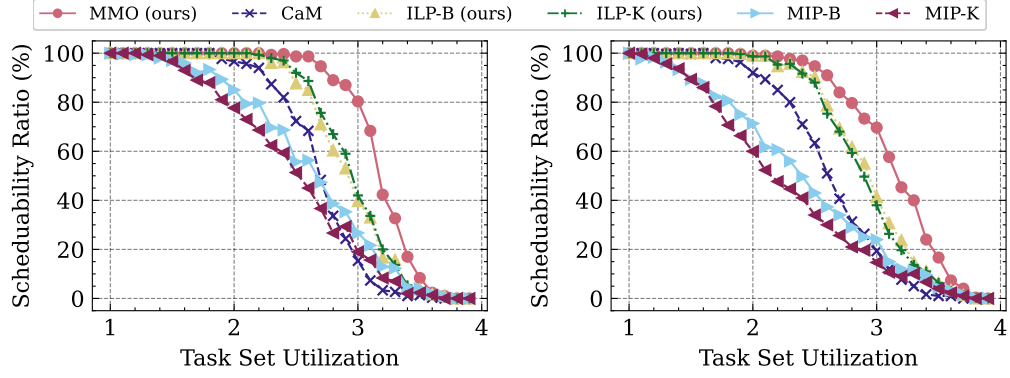
The 0-1 ILPs and MIPs are solved by a standard mathematical programming solver, Gurobi [30]. Given the scalability challenges of mathematical programming solvers for large decision spaces, we set a one-hour time limit per task set. If the solver fails to prove optimality within this limit, the best solution found is used for evaluation.

All algorithms are implemented in Python 3.12, with Gurobi version 11.0. Experiments were conducted on a workstation equipped with AMD EPYC 7763 CPUs running GNU/Linux.

6.2 Evaluation Results

We evaluate the algorithms on the 18,600 task sets and compare them in terms of schedulability ratio, memory bandwidth usage, cache usage, and solving efficiency. Additionally, we report the number of Pareto-optimal solutions obtained by MMO.

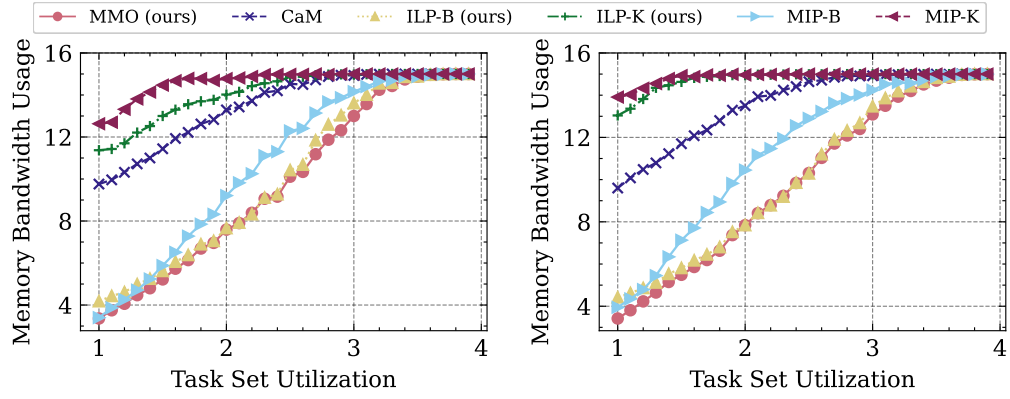
⁵ <https://www.cis.upenn.edu/~linhphan/papers/rtas19-CaM-techreport.pdf>



(a) PARSEC Benchmarks.

(b) SD-VBS Benchmarks.

■ **Figure 3** Schedulability ratio (*i.e.*, $\frac{\text{\#schedulable task sets}}{\text{\#total task sets}} \times 100\%$).



(a) PARSEC Benchmarks.

(b) SD-VBS Benchmarks.

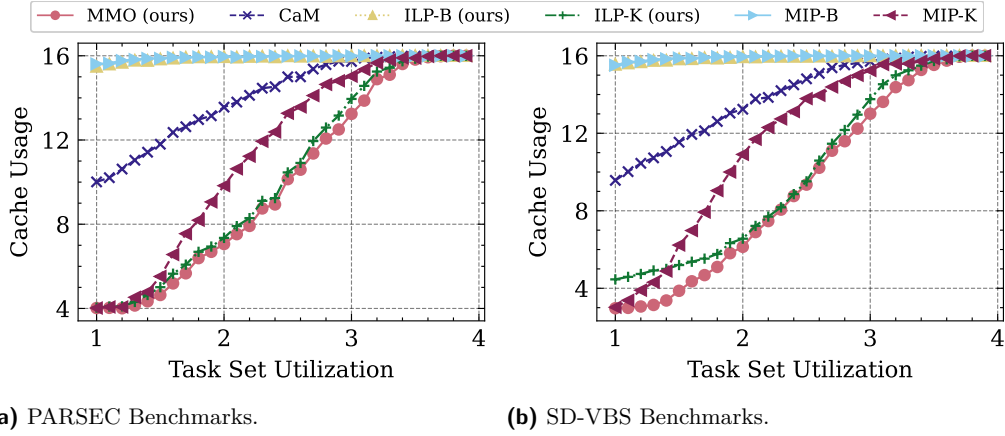
■ **Figure 4** Memory bandwidth usage (number of partitions).

6.2.1 Schedulability

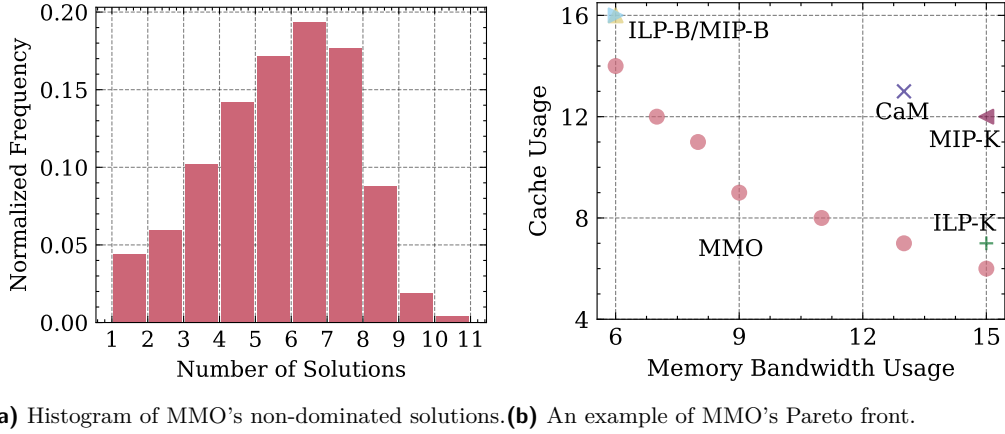
Figure 3 presents schedulability ratio *w.r.t.* reference utilization across all tested task set sizes ($N \in \{20, 40, 60\}$). MMO consistently achieves the highest schedulability ratio across all utilizations and benchmark suites. In particular, MMO identifies up to 62.67% and 50.34% more schedulable task sets than CaM on the *PARSEC* and *SD-VBS* benchmark suites, respectively. The results validate the effectiveness of MMO's design improvement over CaM, as discussed in Section 2.4. Among the mathematical programming models, ILP-B and ILP-K outperform MIP-B and MIP-K because the linear constraint formulation in the developed 0-1 ILP is solved more efficiently by the solver than the non-linear constraints in the MIP. However, they still fail to obtain the optimal solutions within one hour for many task sets, where the best-found solutions are used for evaluation.

6.2.2 Resource Usage

Figure 4 and Figure 5 illustrate the memory bandwidth and cache usage, respectively, measured in the number of partitions required for scheduling. If a task set is found to be unschedulable by an algorithm, the resource usage is considered as the total available partitions of that resource in the system. The y-axis in each figure represents the average



■ **Figure 5** Cache usage (number of partitions).



■ **Figure 6** Pareto-optimal solutions generated by the comparison algorithms.

resource usage across all evaluated task sets at a given system utilization level (x-axis). At each utilization level, the solutions with the lowest memory bandwidth and cache usage among the Pareto-optimal set are selected for comparison in Figures 4 and 5, respectively. Results show that MMO effectively schedules the task sets with the fewest average resources compared to the baselines. For memory bandwidth and cache usage, MMO achieves similar performance to ILP-B and ILP-K, the single objective derivatives of the 0-1 ILP, respectively. CaM outperforms MIP-K/ILP-K and MIP-B/ILP-B for memory bandwidth and cache usage, respectively, because these mathematical programs are not configured to minimize the corresponding resource usage in their objective functions. However, all these models achieve better performance than CaM for the objectives they optimize.

6.2.3 Number of Non-Dominated Solutions

Unlike the baselines, MMO can generate multiple solutions with non-dominated objective values. Figure 6a shows the histogram of the number of non-dominated solutions found by MMO for all schedulable task sets. Results show that MMO finds at least 3 non-dominated solutions for over 90% of schedulable task sets. Figure 6b provides an example Pareto front obtained by MMO and the best solutions found by the baseline algorithms for a task set

with $N = 40$ and $\hat{U} = 2.4$. The Pareto front is comprised of 7 non-dominated solutions, with memory bandwidth usage ranging from 6 to 15 partitions and cache usage ranging from 6 to 14 partitions. For the baselines, ILP-B and MIP-B obtain the same solution with memory bandwidth usage equal to the minimum one found by MMO, but their cache usage is larger and therefore dominated by MMO’s Pareto front. In terms of cache usage, ILP-K yields a solution that requires one additional cache partition compared to MMO’s minimum, both with the same memory bandwidth requirement. Although ILP-K is designed to minimize cache usage, the solution obtained within the one-hour time limit is suboptimal, as the solver did not converge to the optimal solution within that time. The non-dominated solutions in MMO’s Pareto front offer system designers the flexibility to select the most suitable solution based on their preferences.

6.2.4 Solving Efficiency

Table 2 summarizes the runtime performance of each algorithm concerning different task set sizes. MMO achieves the lowest average running time among all algorithms in comparison. Notably, the average solving time of MMO is around 3x lower than CaM. The maximum running time of MMO across all task sets is 17.1 seconds, significantly lower than the baselines.

■ **Table 2** Solving time comparison. Each entry gives the average, minimum, and maximum running times in seconds of an algorithm considering all task sets with a certain size (N). “OOT” represents “out of time,” meaning the solver reaches the one-hour limit.

Algorithm	$N = 20$			$N = 40$			$N = 60$		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
MMO	2.16	0.04	8.73	3.75	0.06	12.0	5.06	0.08	17.1
CaM	7.94	0.02	45.6	12.3	0.09	67.3	16.8	0.14	91.6
ILP-B	1294	8.43	OOT	2536	31.2	OOT	2878	58.9	OOT
ILP-K	1185	5.67	OOT	2270	21.7	OOT	2469	51.3	OOT
MIP-B	1756	1.73	OOT	3390	21.1	OOT	3485	34.4	OOT
MIP-K	1881	1.17	OOT	3019	5.29	OOT	3205	10.5	OOT

7 Conclusion

This paper studies a multi-objective task-resource co-allocation problem to minimize cache and memory bandwidth usage while ensuring task schedulability under a partitioned preemptive EDF scheduling policy. To this end, we first formulate a zero-one linear programming model (0-1 ILP), which can be solved to optimize every single objective separately using a standard mathematical solver. To overcome the problem of exponential time complexity of the ILP and its inability to generate multiple non-dominated solutions within a reasonable time, we propose MMO, a multi-objective multi-layer optimization heuristic that optimizes both objectives simultaneously. Our approach adopts a multi-layer optimization framework where the outer layer explores resource allocations using Pareto-pruned search, while the inner one optimizes task allocation.

We evaluate our approach by deriving slowdown profiles for real-world benchmarks from the *SD-VBS* and *PARSEC* suites executed on a real embedded AMD UltraScale+ ZCU102 platform. We configure fine-grained resource partitioning on this MPSoC using the Jailhouse hypervisor, leveraging cache set partitioning and Memguard for memory bandwidth

regulation. Our experiments based on these real-world benchmarks demonstrate that our heuristics perform consistently better than the state-of-the-art in terms of schedulability, resource usage, number of non-dominated solutions, and solving efficiency.

In future work, we will investigate task-resource co-allocation strategies for parallel real-time tasks with data dependencies (*e.g.*, modeled by directed acyclic graphs).

References

- 1 Luca Abeni, Alessandro Biondi, and Enrico Bini. Partitioning real-time workloads on multi-core virtual machines. *Journal of Systems Architecture*, 131:102733, 2022. doi:10.1016/J.SYSARC.2022.102733.
- 2 Homa Aghilinasab, Waqar Ali, Heechul Yun, and Rodolfo Pellizzoni. Dynamic memory bandwidth allocation for real-time gpu-based soc platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3348–3360, 2020. doi:10.1109/TCAD.2020.3012210.
- 3 Sebastian Altmeyer, Roeland Douma, Will Lunniss, and Robert I. Davis. Outstanding paper: Evaluation of cache partitioning for hard real-time systems. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 15–26, 2014. doi:10.1109/ECRTS.2014.11.
- 4 Sebastian A. Altmeyer, Roeland Douma, Will Lunniss, and Robert I. Davis. On the effectiveness of cache partitioning in hard real-time systems. *Real Time Systems*, 52(5):598–643, 2016. doi:10.1007/S11241-015-9246-8.
- 5 AMD. Zynq UltraScale+ Device Technical Reference Manual. URL: <https://docs.amd.com/r/en-US/ug1085-zynq-ultrascale-trm/Zynq-UltraScale-Device-Technical-Reference-Manual>.
- 6 Arm. Arm DynamIQ Shared Unit Technical Reference Manual. Last accessed 25 April 2025. URL: <https://developer.arm.com/documentation/100453/>.
- 7 Arm. Arm Memory System Resource Partitioning and Monitoring (MPAM) System Component Specification. Last accessed 25 April 2025. URL: <https://developer.arm.com/documentation/ih10099/>.
- 8 Arm Developer. Optimization Basics: Memory bandwidth, 2024. Last accessed 25 April 2025. URL: <https://developer.arm.com/documentation/101897/0303/Optimization-basics/Memory-bandwidth>.
- 9 Sanjoy Baruah and Nathan Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *IEEE International Real-Time Systems Symposium (RTSS)*, 2005.
- 10 Brice Berna and Isabelle Puaut. PDPA: Period driven task and cache partitioning algorithm for multi-core systems. In *International Conference on Real-Time and Network Systems (RTNS)*, pages 181–189, 2012. doi:10.1145/2392987.2393010.
- 11 Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: characterization and architectural implications. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 72–81, 2008. doi:10.1145/1454115.1454128.
- 12 Björn B. Brandenburg and Mahircan Gul. Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 99–110, 2016. doi:10.1109/RTSS.2016.019.
- 13 Bach D. Bui, Marco Caccamo, Lui Sha, and Joseph Martinez. Impact of cache partitioning on multi-tasking real time embedded systems. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 101–110, 2008.
- 14 Almut Burchard, Jörg Liebeherr, Yingfeng Oh, and Sang H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, 1995. doi:10.1109/12.477248.
- 15 Daniel Casini, Alessandro Biondi, Geoffrey Nelissen, and Giorgio Buttazzo. Partitioned fixed-priority scheduling of parallel tasks without preemptions. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 421–433, 2018. doi:10.1109/RTSS.2018.00056.

- 16 Micaiah Chisholm, Bryan C. Ward, Namhoon Kim, and James H. Anderson. Cache sharing and isolation tradeoffs in multicore mixed-criticality systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 305–316, 2015. doi:10.1109/RTSS.2015.36.
- 17 Purnendu Das, Nurulla Mansur Barbhuiya, and Bishwa Ranjan Roy. A survey on way-based cache partitioning. In *IEEE Silchar Subsection Conference (SILCON)*, pages 1–7, 2023.
- 18 Howard David, Chris Fallin, Eugene Gorbato, Ulf R Hanebutte, and Onur Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 31–40, 2011. doi:10.1145/1998582.1998590.
- 19 Robert I Davis, David Griffin, and Iain Bate. A framework for multi-core schedulability analysis accounting for resource stress and sensitivity. *Real-Time Systems*, 58(4):456–508, 2022. doi:10.1007/S11241-022-09377-8.
- 20 Kalyanmoy Deb. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*, pages 3–34. Springer, 2011. doi:10.1007/978-0-85729-652-8_1.
- 21 Sudarshan K Dhall and Chung Laung Liu. On a real-time scheduling problem. *Operations research*, 26(1):127–140, 1978. doi:10.1287/OPRE.26.1.127.
- 22 Zheng Dong, Cong Liu, Soroush Bateni, Kuan-Hsun Chen, Jian-Jia Chen, Georg von der Brüggen, and Junjie Shi. Shared-resource-centric limited preemptive scheduling: A comprehensive study of suspension-based partitioning approaches. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 164–176, 2018. doi:10.1109/RTAS.2018.00026.
- 23 Pontus Ekberg and Sanjoy Baruah. Partitioned scheduling of recurrent real-time tasks. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 356–367, 2021. doi:10.1109/RTSS52674.2021.00040.
- 24 Farzad Farshchi, Qijing Huang, and Heechul Yun. BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 364–375, 2020. doi:10.1109/RTAS48715.2020.00011.
- 25 Nathan Fisher and Sanjoy Baruah. The partitioned multiprocessor scheduling of non-preemptive sporadic task systems. In *International Conference on Real-Time and Network Systems (RTNS)*, 2006.
- 26 José Fonseca, Geoffrey Nelissen, Vincent Nelis, and Luís Miguel Pinho. Response time analysis of sporadic dag tasks under partitioned scheduling. In *IEEE Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, 2016.
- 27 Robert Gifford, Neeraj Gandhi, Linh Thi Xuan Phan, and Andreas Haeberlen. DNA: Dynamic resource allocation for soft real-time multicore systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 196–209, 2021. doi:10.1109/RTAS52030.2021.00024.
- 28 David Griffin, Iain Bate, and Robert I Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 76–88, 2020. doi:10.1109/RTSS49844.2020.00018.
- 29 Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Cache-aware scheduling and analysis for multicores. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 245–254, 2009. doi:10.1145/1629335.1629369.
- 30 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2025. Last accessed 25 April 2025. URL: <https://www.gurobi.com>.
- 31 Mohamed Hassan. Reduced latency DRAM for multi-core safety-critical real-time systems. *Real-Time Systems*, pages 1–36, 2019.
- 32 Intel. Resource Director Technology. Last accessed 25 April 2025. URL: <https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html>.

- 33 Hyoseung Kim and Ragunathan Rajkumar. Real-time cache management for multi-core virtualization. In *International Conference on Embedded Software (EMSOFT)*, pages 1–10, 2016. doi:10.1145/2968478.2968480.
- 34 Namhoon Kim, Bryan C. Ward, Micaiah Chisholm, Cheng-Yang Fu, James H. Anderson, and F. Donelson Smith. Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning. *Real-Time Systems*, 53:709–759, 2017. doi:10.1007/S11241-017-9272-9.
- 35 Tomasz Kloda, Marco Solieri, Renato Mancuso, Nicola Capodieci, Paolo Valente, and Marko Bertogna. Deterministic memory hierarchy and virtualization for modern multi-core embedded systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–14, 2019. doi:10.1109/RTAS.2019.00009.
- 36 Benjamin Lesage, Xiaotian Dai, Shuai Zhao, and Iain Bate. Reducing loss of service for mixed-criticality systems through cache-and stress-aware scheduling. In *Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS)*, pages 188–199, 2023. doi:10.1145/3575757.3593654.
- 37 Yuhan Lin, Jinghao Sun, Qingxu Deng, Meilin Han, Zhiwei Feng, and Shumo Wang. Lag-based analysis for preemptive global scheduling with dynamic cache allocation. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 107–116, 2023. doi:10.1109/RTCSA58653.2023.00022.
- 38 Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973. doi:10.1145/321738.321743.
- 39 Mingsong Lv, Nan Guan, Jan Reineke, Reinhard Wilhelm, and Wang Yi. A survey on static cache analysis for real-time systems. *Leibniz Transactions on Embedded Systems*, 3(1):05–1, 2016.
- 40 José María López, Manuel Garcia, Jose Díaz, and Frk Daniel Garcia. Worst-case utilization bound for edf scheduling on real-time multiprocessor systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2000.
- 41 Claire Maiza, Hamza Rihani, Juan M Rivas, Joël Goossens, Sebastian Altmeyer, and Robert I Davis. A survey of timing verification techniques for multi-core real-time systems. *ACM Computing Surveys*, 52(3):1–38, 2019. doi:10.1145/3323212.
- 42 Renato Mancuso, Roman Dudko, Emiliano Betti, Marco Cesati, Marco Caccamo, and Rodolfo Pellizzoni. Real-time cache management framework for multi-core architectures. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 45–54, 2013. doi:10.1109/RTAS.2013.6531078.
- 43 Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.
- 44 Carlo Meroni, Silviu S. Craciunas, Anaïs Finzi, and Paul Pop. Mapping and integration of event- and time-triggered real-time tasks on partitioned multi-core systems. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2023. doi:10.1109/ETFA54631.2023.10275547.
- 45 Minerva Systems. Memory-aware Jailhouse hypervisor. Last accessed 25 April 2025. URL: <https://github.com/Minervasys/jailhouse>.
- 46 Sparsh Mittal. A survey of architectural techniques for improving cache power efficiency. *Sustainable Computing: Informatics and Systems*, 4(1):33–43, 2014. doi:10.1016/J.SUSCOM.2013.11.001.
- 47 Lei Mo, Qi Zhou, Angeliki Kritikakou, and Xianghui Cao. Energy optimized task mapping for reliable and real-time networked systems. *ACM Trans. Sen. Netw.*, 19(4), April 2023. doi:10.1145/3584985.
- 48 Mattia Nicolella, Shahin Roozkhosh, Denis Hoornaert, Andrea Bastoni, and Renato Mancuso. Rt-bench: An extensible benchmark framework for the analysis and management of real-time applications. In *International Conference on Real-Time Networks and Systems (RTNS)*, pages 184–195, 2022. doi:10.1145/3534879.3534888.

- 49 Lanshun Nie, Chenghao Fan, Shuang Lin, Li Zhang, Yajuan Li, and Jing Li. Holistic resource allocation under federated scheduling for parallel real-time tasks. *ACM Trans. Embed. Comput. Syst.*, 21(1), 2022. doi:10.1145/3489467.
- 50 NVIDIA. Jetson Orin. Last accessed 25 April 2025. URL: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/>.
- 51 Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, Robert I. Davis, and Mateo Valero. IA³: An interference aware allocation algorithm for multicore hard real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 280–290, 2011. doi:10.1109/RTAS.2011.34.
- 52 Jinsu Park, Seongbeom Park, and Woongki Baek. Copart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers. In *Proceedings of the Fourteenth EuroSys Conference (Eurosys)*, pages 1–16, 2019. doi:10.1145/3302424.3303963.
- 53 Ashutosh Pradhan, Daniele Ottaviano, Yi Jiang, Haozheng Huang, Alexander Zuepke, Andrea Bastoni, and Marco Caccamo. Arm DynamIQ Shared Unit and Real-Time: An Empirical Evaluation. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 269–282, 2025.
- 54 Gero Schwaerliche, Rohan Tabish, Rodolfo Pellizzoni, Renato Mancuso, Andrea Bastoni, Alexander Zuepke, and Marco Caccamo. A Real-Time virtio-based Framework for Predictable Inter-VM Communication. In *IEEE Real-Time Systems Symposium (RTSS)*, 2021.
- 55 Ikram Senoussaoui, Houssam-Eddine Zahaf, Mohammed Kamel Benhaoua, Giuseppe Lipari, and Richard Olejnik. Allocation of real-time tasks onto identical core platforms under deferred fixed preemption-point model. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems, RTNS '20*, pages 34–43, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3394810.3394821.
- 56 Siemens AG. Jailhouse hypervisor. Last accessed 25 April 2025. URL: <https://github.com/siemens>.
- 57 Parul Sohal, Michael Bechtel, Renato Mancuso, Heechul Yun, and Orran Krieger. A Closer Look at Intel Resource Director Technology (RDT). In *Proceedings of the 30th International Conference on Real-Time Networks and Systems (RTNS)*, pages 127–139, 2022.
- 58 Binqi Sun, Tomasz Kloda, Sergio Arribas Garcia, Giovanni Gracioli, and Marco Caccamo. Minimizing cache usage for real-time systems. In *Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS)*, pages 200–211, 2023. doi:10.1145/3575757.3593651.
- 59 Binqi Sun, Tomasz Kloda, and Marco Caccamo. Strict partitioning for sporadic rigid gang tasks. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 252–264, 2024. doi:10.1109/RTAS61025.2024.00028.
- 60 Binqi Sun, Tomasz Kloda, Sergio Arribas Garcia, Giovanni Gracioli, and Marco Caccamo. Minimizing cache usage with fixed-priority and earliest deadline first scheduling. *Real-Time Systems*, pages 1–40, 2024.
- 61 Binqi Sun, Tomasz Kloda, Chu-ge Wu, and Marco Caccamo. Partitioned scheduling and parallelism assignment for real-time dnn inference tasks on multi-tpu. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2024. doi:10.1145/3649329.3655979.
- 62 Binqi Sun, Debayan Roy, Tomasz Kloda, Andrea Bastoni, Rodolfo Pellizzoni, and Marco Caccamo. Co-optimizing cache partitioning and multi-core task scheduling: Exploit cache sensitivity or not? In *IEEE Real-Time Systems Symposium (RTSS)*, pages 224–236, 2023. doi:10.1109/RTSS59052.2023.00028.
- 63 Niklas Ueter, Mario Günzel, Georg von der Brüggen, and Jian-Jia Chen. Hard Real-Time Stationary GANG-Scheduling. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, volume 196 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECRTS.2021.10.

- 64 Sravanthi Kota Venkata, Ikkjin Ahn, Donghwan Jeon, Anshuman Gupta, Christopher Louie, Saturnino Garcia, Serge Belongie, and Michael Bedford Taylor. SD-VBS: The San Diego vision benchmark suite. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 55–64, 2009. doi:10.1109/IISWC.2009.5306794.
- 65 Xilinx. Xilinx Xen Support with Cache-Coloring. Last accessed 25 April 2025. URL: <https://github.com/Xilinx/xen/releases/tag/xilinx-v2020.2>.
- 66 Meng Xu, Robert Gifford, and Linh Thi Xuan Phan. Holistic multi-resource allocation for multicore real-time virtualization. In *Design Automation Conference (DAC)*, 2019.
- 67 Meng Xu, Linh Thi Xuan Phan, Hyon-Young Choi, Yuhang Lin, Haoran Li, Chenyang Lu, and Insup Lee. Holistic resource allocation for multicore real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 345–356, 2019. doi:10.1109/RTAS.2019.00036.
- 68 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 55–64, 2013. doi:10.1109/RTAS.2013.6531079.
- 69 Houssam-Eddine Zahaf, Giuseppe Lipari, Smail Niar, and Abou El Hassan Benyamina. Preemption-aware allocation, deadline assignment for conditional dags on partitioned edf. In *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10, 2020. doi:10.1109/RTCSA50079.2020.9203643.
- 70 Houssam-Eddine Zahaf, Ignacio Sanudo Olmedo, Jayati Singh, Nicola Capodieci, and Sebastien Faucou. Contention-aware gpu partitioning and task-to-partition allocation for real-time workloads. In *Proceedings of the 29th International Conference on Real-Time Networks and Systems, RTNS '21*, pages 226–236, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3453417.3453439.
- 71 Alexander Zuepke, Andrea Bastoni, Weifan Chen, Marco Caccamo, and Renato Mancuso. MemPol: Policing core memory bandwidth from outside of the cores. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 235–248, 2023. doi:10.1109/RTAS58335.2023.00026.