Cache Sharing and Isolation Tradeoffs in Multicore Mixed-Criticality Systems [*]

Micaiah Chisholm, Bryan C. Ward, Namhoon Kim, and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

In mixed-critical applications, tension exists between sharing and isolation with respect to hardware resources: while strong isolation might be required for highly critical tasks, somewhat permissive sharing might be reasonable for less critical tasks to improve throughput or average-case performance. In this paper, this tension is examined as it pertains to shared last-level caches (LLCs) on multicore platforms. In particular, criticality-aware optimization techniques based on linear programming are presented for allocating LLC areas in the context of the previously proposed MC^2 ($\underline{m}$ixed-$\underline{c}$riticality on $\underline{m}$ulti$\underline{c}$ore) framework. Experiments are also presented that show that these techniques can result in significant schedulability improvements.

1 Introduction

The adoption of multicore machines in safety-critical domains is being hampered by aspects of such machines that reflect a throughput-oriented design philosophy. For example, it is common practice today to allow hardware components such as last-level caches (LLCs) and memory controllers to be shared across cores; this can be beneficial as long as any detrimental effects due to sharing are not typically seen on average. Unfortunately, such sharing can result in timing behaviors that are exceedingly difficult to characterize in the worst case without excessive pessimism. This is problematic for safety-critical domains, where correct execution must be validated even in worst-case scenarios.

Excessive pessimism due to shared hardware is a key contributing factor to a problem termed here the "one-out-of-m" problem: when checking real-time constraints on a multicore platform with m cores, analysis pessimism can easily negate the processing capacity of the additional $m-1$ cores. In effect, only "one core's worth" of capacity can be utilized even though m cores are available. In domains such as avionics, this problem has led to the common practice of simply disabling all but one core.[1] This problem is the most serious unresolved obstacle in work on real-time multicore resource allocation today.

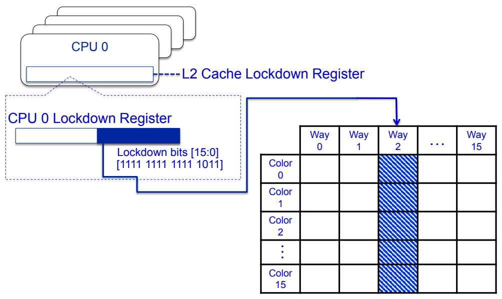
The desire to reduce the pessimism caused by unmanaged shared hardware has led to intense recent interest in hardware management techniques [1, 9, 10, 11, 14, 18, 19]. A common goal here is to provide *isolation* by *partitioning* hardware resources among cores and/or tasks to eliminate sharing altogether. However, this can be an overly strong solution in many contexts: even safety-critical applications often have system components that are not highly critical and that could therefore benefit from less constrained sharing. A better way forward might be to achieve some appropriate *balance* between sharing and isolation based on the criticalities of the software components involved. *In this paper, we investigate this issue of balance as it relates to shared LLCs.*

Mixed-criticality systems. Our work fits within the larger body of research on mixed-criticality (MC) resource allocation spawned by a seminal paper of Vestal [17]. He proposed analyzing the real-time requirements of less critical tasks under less pessimistic analysis assumptions. Specifically, to analyze a system with L criticality levels, he proposed specifying a *provisioned execution time* (*PET*) for *each* task at *every* level and analyzing L *different* system variants: in the Level-ℓ variant, the real-time requirements of all Level-ℓ tasks are verified with Level-ℓ PETs assumed for *all* tasks (at any level). The degree of pessimism in determining PETs is level-dependent: if Level ℓ is of higher criticality than Level ℓ', then Level-ℓ PETs will generally be greater than Level-ℓ' PETs. Vestal's work led to approximately 200 follow-up papers on MC scheduling by a variety of authors. An excellent survey of this work has been prepared by Davis and Burns [4]. They note that the fundamental research question in this area as "reconcil[ing] the conflicting requirements of partitioning for (safety) assurance and *sharing* for efficient resource usage." This is the very issue investigated herein (as it relates to shared LLCs).

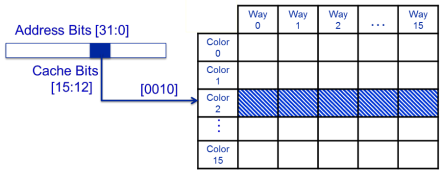
Cache partitioning. Under cache partitioning, designated cache areas are assigned to certain tasks, sets of tasks, or cores. Assuming a set associative cache, this can be achieved through some combination of *page coloring*, to provide *set-based partitioning*, or the use of hardware support in the form of *lockdown registers*, to provide *way-based partitioning*. These alternatives are illustrated in Fig. 1 with respect to a quad-core ARM Cortex A9 machine, which is the canonical hardware platform considered herein. As seen in inset (a), each core on this machine has a lockdown register, the bits of which can be cleared to steer LLC accesses from this core to certain ways of the LLC. Under page coloring, pages of physical memory are assigned colors, and sets of the LLC are colored corresponding to how such pages map to them. As seen in inset (b), this technique ensures that differently colored pages cannot cause conflicts in the LLC.

[*]Work supported by NSF grants CNS 1115284, CNS 1218693, CPS 1239135, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and a grant from General Motors. The second author was also supported by an NSF graduate fellowship.

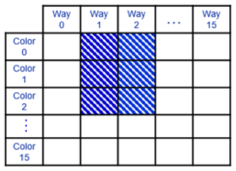
[1]In fact, the U.S. Federal Aviation Administration is currently considering the possibility of *mandating* such an approach when multicore platforms are used in avionic systems.



(a) Way-based partitioning.



(b) Set-based partitioning.



(c) Combined approach.

Figure 1: Allocating LLC areas by way, or set, or both. On this machine, the LLC is an L2 cache shared by four cores.

MC^2. Our examination of LLC allocation tradeoffs in MC systems is based upon the MC^2 (mixed-criticality on multicore) framework [8, 16, 18], which has been the subject of continuing research by our group.[2] In MC^2, four criticality levels exist, denoted A (highest) through D (lowest), as shown in Fig. 2. Higher-criticality tasks are statically prioritized over lower-criticality ones. Level-A tasks are partitioned and scheduled on each core using a time-triggered table-driven cyclic executive.[3] Level-B tasks are also partitioned but are scheduled using a rate-monotonic (RM) scheduler on each core.[3] On each core, the Level-A and -B tasks are required to be simply periodic (all tasks commence execution at time 0 and periods are harmonic), with the Level-B task periods being integer multiples of the Level-A hyper-period. These tasks have hard real-time (HRT) constraints. Level-C tasks have soft real-time (SRT) constraints and are scheduled via a global earliest-deadline-first (G-EDF) scheduler;[3] the considered SRT constraint is that deadline tardiness is provably bounded. Level-D tasks

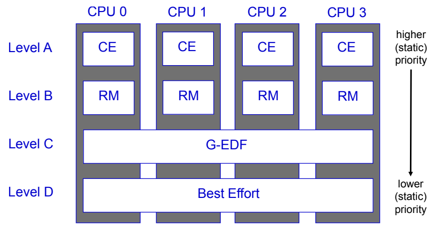


Figure 2: Scheduling in MC^2 on a quad-core machine.

are scheduled with no real-time guarantees (so we do not consider them further). MC^2 is a flexible framework from a research point-of-view. For example, it can be configured to have only two HRT criticality levels (as in most theoretical work on MC scheduling) or to fully assign the Level-A and -B subsystems to distinct, dedicated cores.

In recent work, we extended MC^2 to support partitioning with respect to the LLC and DRAM banks and to isolate the operating system from application tasks. In a companion paper [12], we describe how these features were implemented and present experiments that demonstrate the virtues of the supported isolation mechanisms in MC systems. In that effort, we considered a single generic LLC allocation strategy.

Contributions. In this paper, we consider the problem of optimizing LLC allocations in the context of MC^2 for a specific task system. We consider a general criticality-aware LLC allocation framework that allows leeway in precisely determining allocated LLC areas for a specific task system. We study the problem of determining such allocations formally. We first discuss how to model the impacts of a given allocation strategy on LLC-related overheads and task execution times. We then adopt a particular model that allows us to determine LLC allocations by solving a linear program (LP). To analyze the effectiveness of this approach, we present a schedulability study involving randomly generated task systems where generated task execution times were based on measurement data. In this study, the usage of our techniques enabled schedulability improvements of up to 100% for some task-system categories in comparison to two generic task-system-oblivious LLC allocation strategies, including that considered in [12]. The presented LP can be solved as either an ordinary LP or a mixed-integer LP (MILP). In our experiments, both variants exhibited similar runtime performance, both often yielded nearly identical schedulability results, but for some task systems, schedulability was noticeably better under the MILP variant.

To our knowledge, LLC allocation strategies for MC systems have not been considered before, particularly in a context with as many interesting tradeoffs as MC^2. Nonetheless, there has been much prior work on cache partitioning. We review this work later in Sec. 2 to more properly position our contributions.

Organization. In the remainder of the paper, we provide relevant background (Sec. 2), describe our LLC allocation techniques (Secs. 3 and 4), present our experimental evaluation (Sec. 5), and conclude (Sec. 6).

[2] MC^2 should not be confused with a similarly named European project that began several years *after* work on MC^2 commenced.

[3] A RM (EDF) scheduler can be optionally used at Level A (B). Additionally, any G-EDF-like (GEL) scheduler [7] can be used at Level C. Furthermore, Level-C tasks can be defined according to the sporadic task model. For simplicity, we do not consider these options further herein. Other facets of MC^2, such as slack reallocation, schedulability conditions, and execution-time budgeting are discussed in prior papers [8, 16, 18].

2 Background

In this section, we present relevant notation, formally define the problem solved in this paper, and discuss related work.

Task system notation. We consider a set of implicit-deadline periodic tasks $\Gamma \equiv \{\tau_1, \tau_2, \tau_3, ..., \tau_N\}$ to be scheduled under the MC^2 framework on m cores. We only consider Levels A–C in MC^2, as Level D is non-real-time. Each task τ_i has a period T_i, and three PETs, e_i^A, e_i^B, and e_i^C, where e_i^ℓ denotes its Level-ℓ PET (recall the discussion concerning MC schedulability analysis in Sec. 1). We let Γ_A, Γ_B, and Γ_C denote the subset of tasks in Γ at Levels A, B, and C, respectively. Also, we let $\Gamma_{A,p}$ and $\Gamma_{B,p}$ denote the subset of tasks in Γ_A and Γ_B, respectively, that are assigned to core p. We denote the total utilization of all Level-ℓ tasks assuming Level-ℓ' execution times as $U_\ell^{\ell'} \equiv \sum_{\tau_i \in \Gamma_\ell} \frac{e_i^{\ell'}}{T_i}$. We denote the total utilization of all Level-A or -B tasks assigned to core p assuming Level-ℓ' execution times as $U_{A,p}^{\ell'} \equiv \sum_{\tau_i \in \Gamma_{A,p}} \frac{e_i^{\ell'}}{T_i}$ and $U_{B,p}^{\ell'} \equiv \sum_{\tau_i \in \Gamma_{B,p}} \frac{e_i^{\ell'}}{T_i}$, respectively. The schedulability condition for Level C is dependent on the largest utilization of any task at Level C, which we denote as h, and the sum of the $m-1$ largest task utilizations at Level C, which we denote as H. The following are sufficient conditions for ensuring schedulability at all three criticality levels [16].

$$\forall p :: U_{A,p}^A \leq 1 \wedge \tag{1}$$

$$\forall p :: U_{A,p}^B + U_{B,p}^B \leq 1 \wedge \tag{2}$$

$$U_A^C + U_B^C + U_C^C \leq m \wedge \tag{3}$$

$$U_A^C + U_B^C + (m-1)h + H < m \tag{4}$$

We assume that PETs are determined through a measurement process, as often done in practice (indeed, on multicore platforms adequate static timing-analysis tools do not yet exist). Specifically, we assume that Level-C PETs reflect measured average-case execution times[4] (since Level C is SRT) and that Level-B PETs reflect measured worst-case execution times (since Level B is HRT). Further, we assume that Level-A PETs are defined by inflating Level-B PETs by 50% (since Level A is of highest criticality). Such an inflation is in keeping with inflation factors derived from industrial use cases considered by Vestal [17]. These measurement-based PETs will generally depend on allocated LLC areas.[5] We denote the Level-ℓ PET of task τ_i when its allocated LLC area consists of W ways and S colors (refer to Fig. 1) as $e_i^\ell(W, S)$. (We use "S" in denoting colors because colors determine LLC *sets*, and the term "C" has a predefined meaning in the context of MC^2.)

[4]In MC^2, a Level-ℓ task's Level-ℓ PET is treated as an enforced execution budget. As explained in [15], tardiness bounds with respect to deterministic budget allocations at Level C can be used to bound tardiness in expectation when average-case task execution times are assumed.

[5]We often use the term "area" instead of "partition" because we allow for the possibility that such areas overlap.

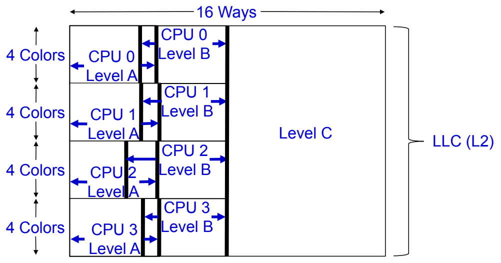


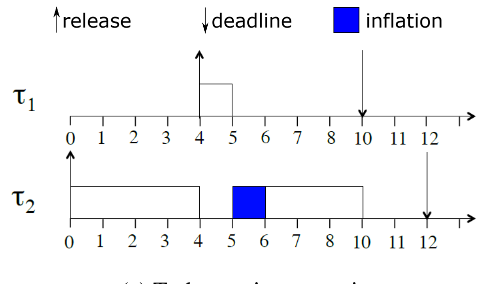
Figure 3: Canonical LLC allocation.

Canonical LLC allocation and problem to be solved. We consider a canonical LLC allocation, which is illustrated in Fig. 3 with respect to our quad-core ARM platform, the LLC of which has 16 colors and 16 ways. Assuming an LLC with W^{max} ways in total, all Level-C tasks together are allocated an LLC area that consists of all colors (sets) associated with ways W_C through $W^{max}-1$ for some W_C. All LLC areas for Level-A and -B tasks are taken from the colors (sets) associated with ways 0 through $W_C - 1$. The Level-A and -B tasks on each core use an LLC area consisting of $1/m$ ($m = 4$ on our platform) of the colors (sets) associated with these ways, as depicted. Each per-core Level-A and -B LLC area is subdivided into potentially overlapping Level-A and -B areas. This allocation scheme provides the following notions of spatial and temporal isolation with respect to the LLC (*spatial isolation* is guaranteed when access to common LLC areas is categorically prevented, and *temporal isolation* is guaranteed when a task's lines in a common LLC area cannot be evicted while it is using them).

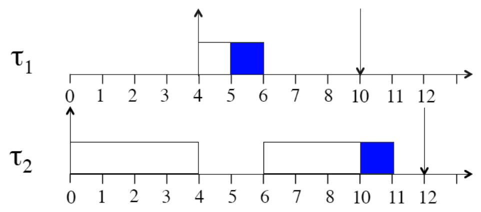
- Level-C tasks are spatially isolated from Level-A and -B tasks.
- Level-A and -B tasks on one core are spatially isolated from Level-A and -B tasks on other cores.
- Level-A and -B tasks on the same core are spatially isolated with respect to the ways that they do not share. Additionally, Level-A tasks are temporally isolated from Level-B tasks with respect to the ways they share because Level-A tasks have higher priority.

This general allocation strategy reflects two assumptions: Level-C tasks, being SRT and provisioned on the average case, might benefit from rather unrestricted LLC sharing; Level-A and -B tasks, being HRT and more critical, might require stronger LLC isolation guarantees. With regard to the latter, note that the set of Level-A tasks on one core is completely isolated (either spatially or temporally) from all other tasks in the system with respect to the LLC.

The technical problem considered in this paper is to determine how to precisely size these LLC areas so as to enhance schedulability given the characteristics of the task system in question. That is, we seek to determine how the bold lines in Fig. 3 should be set. In addressing this problem, we assume that an assignment of all Level-A and -B tasks to cores has already been determined.



(a) Task-centric accounting.



(b) Preemption-centric accounting.

Figure 4: Forms of overhead accounting.

Overhead accounting. Depending on how task systems are analyzed, execution times and schedulability conditions may not include the impact of system overheads. In this paper, we consider one such overhead, *cache-related preemption delays* (*CRPDs*), and how this overhead is affected by our LLC allocation methods. CRPDs are delays a task may incur to reload lines evicted from the LLC (and other caches) due to a preemption. We discuss how to quantify CRPDs with respect to LLC allocation sizes, so that these delays can be integrated into schedulability analysis.

There are two basic ways to account for CRPD costs, as shown in Fig. 4. Under *task-centric accounting*, the execution time of the *preempted* job is inflated to account for the preemption. Under *preemption-centric accounting*, the execution time of the *preempting* job is inflated to "pay" for the CPRD cost of any preempted job that *resumes* execution when the preempting job completes. We consider preemption-centric accounting here, because it usually introduces less pessimism in schedulability analysis, and because it can be linearly modeled by simply adding an inflation term to each execution cost. (Task-centric accounting entails the introduction of non-linear ceiling and/or floor operators.)

Related work. Having fully specified the problem to be solved in this paper, and some of the assumptions we make in solving it, we now discuss related work.

The use of cache partitioning in real-time systems has been investigated before. A good overview of early work on this topic has been given by Kirk [13]. In more recent work, Kim *et al.* [11] presented a cache-partitioning scheme that allows multiple tasks to share the same cache partition on a single processor (as we do for Level-A and -B tasks), but they did not consider MC systems. Altmeyer *et al.* [1] considered uniprocessor scheduling on a system with a direct-mapped cache and examined worst-case execution time (WCET) estimates as a function of cache size. They also presented a cache-partitioning algorithm that is optimal under certain cache-modeling assumptions. As an alternative to cache partitioning, a technique called *cache lockdown* can be used that prevents designated cached data or instructions from being evicted [5]. Also, it is possible to redesign the cache allocator itself to provide a replacement policy that enables greater predictability [9].

3 MC^2 LLC-Managed Overhead Accounting

In Sec. 2, we discussed preemption-centric CRPD accounting. In this section, we discuss our methods for determining required overhead inflations for task execution times under a managed LLC. These methods for overhead accounting ensure task execution-time properties used by our LP programs, discussed in Sec. 4, hold for both inflated and non-inflated execution times.

The inflation term we add is generally a function of a task's allocated LLC area size. For example, we can inflate the Level-ℓ execution time of any Level-B or -C task that has an LLC area consisting of W ways and S colors as follows:

$$\forall i : \tau_i \in \Gamma_B \cup \Gamma_C :: {e'}^{\ell}_i(W,S) \equiv e^{\ell}_i(W,S) + E^{\ell}(W,S), \tag{5}$$

where $E^{\ell}(W,S)$ is the time required, according to Level-ℓ analysis, to reload all cache lines within a region of the LLC consisting of W ways and S colors. Note that this is the LLC area of *both* the preempting and preempted task: for preemptions of Level-B tasks by Level-B tasks (or Level-C tasks by Level-C tasks), the preempting job shares the same LLC area as the preempted job. We assume a constant time b^{ℓ} is required under Level-ℓ analysis assumptions to load the lines within an LLC area consisting of only one way and one color. Under this assumption, our inflation term is

$$E^{\ell}(W,S) = W \cdot S \cdot b^{\ell}. \tag{6}$$

We now explain how to introduce inflations into the schedulability conditions (1)–(4) discussed earlier. To do so, we can substitute for each utilization term a corresponding inflated utilization term. We denote the inflated Level-ℓ utilizations of each task τ_i as ${u'}^{\ell}_i \equiv \frac{{e'}^{\ell}_i}{T_i}$. We can then define inflated Level-B and -C utilizations as follows.

$$\forall p :: {U'}^{B}_{B,p} \equiv \sum_{\tau_i \in \Gamma_{B,p}} {u'}^{B}_i$$

$$\forall p :: {U'}^{C}_{B,p} \equiv \sum_{\tau_i \in \Gamma_{B,p}} {u'}^{C}_i$$

$${U'}^{C}_{C} \equiv \sum_{\tau_i \in \Gamma_C} {u'}^{C}_i$$

We also replace h and H in condition (4) with inflated terms h' and H'. h' is the highest inflated Level-C utilization of any Level-C task, and H' is the sum of the $m-1$ highest inflated Level-C utilizations at Level C.

Note that we do not apply the inflation described in

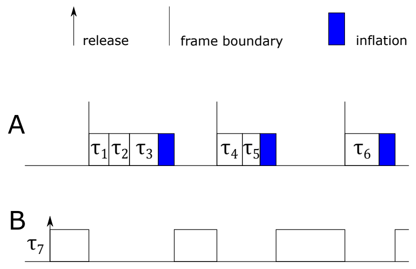


Figure 5: Per-frame Level-B inflation for Level A.

Equation (5) to Level-A jobs. That is, we have

$$\forall i : \tau_i \in \Gamma_A :: e'^{\ell}_i(W,S) \equiv e^{\ell}_i(W,S),$$

$$\forall p :: U'^{A}_{A,p} = U^{A}_{A,p}.$$

Under the cyclic-executive model [2], scheduling is based on fixed-length *frames*. Each Level-A job runs non-preemptively within a frame unless it is *sliced*. Job slicing allocates different portions of a job (job slices) to different frames. We assume the execution time of each job slice is measured independently of other slices when PETs are initially determined. This ensures the PETs of one slice are not affected by cache lines loaded by other job slices.

Level-A jobs may still produce other CRPD overheads. In the event that the LLC areas for Levels A and B on core p overlap, the Level-A tasks on core p may evict all of the cache lines of Level-B tasks within the overlap. This might suggest that the Level-B execution time of each Level-A job requires inflation. However, the required inflation can be less pessimistically determined. As shown in Fig. 5, Level-A jobs allocated to a frame f run sequentially at the beginning of each frame. In this scenario, Level B is only preempted by Level A at most once per frame.

Depending on the replacement policy of the cache, evictions by Level-A tasks within overlapping sets may cause Level-B tasks to evict additional lines throughout the ways allocated to Level-B in the overlapping sets. For Level B, we make the pessimistic assumption that the number of evictions directly or indirectly caused by a Level-A task is equal to the area allocated to Level B in sets it shares with Level A. For Level-C, we make the more optimistic assumption that, on average, the number of evicted cache blocks is equal to the size of the overlap.

The frame size for the cyclic executive of Level-A tasks on core p is equal to the smallest period of any task in $\Gamma_{A,p}$, which we denote $T^{min}_{A,p}$. If the overlap on core p consists of W^O_p ways and S^O_p colors and Level-B is allocated $W_{B,p}$ ways, we can model the overhead associated with reloading cache lines allocated to Level-B once per frame by inflating the Level-B and -C utilizations of Level A.

$$\forall p :: U'^{B}_{A,p} = U^{B}_{A,p} + \frac{E^B(W_{B,p}, S^O_p)}{T^{min}_{A,p}}$$

$$\forall p :: U'^{C}_{A,p} = U^{C}_{A,p} + \frac{E^C(W^O_p, S^O_p)}{T^{min}_{A,p}}$$

Our schedulability conditions with CRPD overheads accounted for are the following.

$$\forall p :: U'^{A}_{A,p} \leq 1 \tag{7}$$

$$\forall p :: U'^{B}_{A,p} + U'^{B}_{B,p} \leq 1 \tag{8}$$

$$\sum_{p=1}^{m} \left(U'^{C}_{A,p} + U'^{C}_{B,p}\right) + U'^{C}_{C} \leq m \tag{9}$$

$$\sum_{p=1}^{m} \left(U'^{C}_{A,p} + U'^{C}_{B,p}\right) + (m-1)h' + H' < m \tag{10}$$

4 Linear Programming

In this section, we show how to solve the canonical LLC allocation problem described in Sec. 2 via a linear program (LP). The LP we obtain determines a choice of ways for each allocated LLC area such that the schedulability conditions (7)–(10) are maintained. This requires treating ways as continuous variables. We explain later how to ultimately obtain an integral solution. We now describe the various sets of constraints in our final LP.

LLC size constraints. The simplest constraint set ensures that there is no overlap between Level-C's partition and any allocated LLC areas at higher criticality levels. We let $\hat{W}_{A,p}$ and $\hat{W}_{B,p}$ denote continuous LP variables indicating the number of ways allocated to Levels A and B, respectively, on core p. We let $\hat{W}_C$ denote a continuous LP variable indicating the number of ways allocated to Level C.

LLC size constraints also determine the overlap between Levels-A and -B LLC areas. $\hat{W}^O_p$ denotes a continuous LP variable modeling the overlap on core p. Recall that W^{max} is the total number of ways in the considered LLC cache. If Level-A and -B LLC areas overlap on core p, the overlap is

$$W^O_p = W_{A,p} + W_{B,p} + W_C - W^{max}.$$

Constraint Set 1. *The LLC size constraints are as follows.*

$$\forall p :: \hat{W}_{A,p} + \hat{W}_C \leq W^{max}$$

$$\forall p :: \hat{W}_{B,p} + \hat{W}_C \leq W^{max}$$

$$\hat{W}^O_p \geq \hat{W}_{A,p} + \hat{W}_{B,p} + W_C - W^{max}.$$

Modeling execution times. The manner in which we model the impact of allocated LLC area sizes on execution times affects the choice of algorithms that can be applied to determine such sizes. Without a clear relationship between execution times and area sizes, there may be no way to determine how adjustments to such sizes impact schedulability except through brute force trial and error. Given the manner in which tasks are prioritized in MC^2, and the canonical LLC allocation framework described above, we require both worst- and average-case execution-time measurements of Level-A and -B tasks, and average-case measurements for Level-C tasks. The Level-A and -B measurements may be taken in a system under load but with isolation provided with respect to the LLC, as described above. The Level-C measurements also need to be taken in a system under load to account for the impact of concurrent evictions and memory-bus contention at Level-C (although it may be appropriate to obtain average-case measurements under a less heavy load than for worst-case measurements).

Such execution-time measurements often exhibit a property we will exploit:

Execution Time Assumption. *The derivative of a task's execution time (at any level) with respect to its allocated LLC area size is non-increasing. That is, the execution time function is non-convex.*

Bui et al. [3] presented graphs for execution times of several avionics applications that approximately meet this condition, suggesting that this behavior is not uncommon. Our measurements for several benchmark programs on our Cortex A9 platform exhibit similar behavior [12]. We note three properties that directly follow from this assumption.

Lemma 1. *The derivative of a task's **inflated** execution time (at any level) with respect to its allocated LLC area size is non-increasing.*

Proof: For our LLC allocation problem, colors are fixed at each level, such that the execution time function for each task τ_i is a function over the number of ways allocated to τ_i. By (6), the inflation function E^ℓ varies linearly with allocated ways, and is thus non-convex. The sum of two non-convex functions is non-convex. □

Lemma 2. *The derivative of a task's inflated **utilization** (at any level) with respect to its allocated LLC area size is non-increasing.*

Proof: This follows from the fact that task utilizations are directly proportional to task execution times. □

We could proceed with the construction of our LP by treating individual task utilizations as variables, but this would entail having $O(N)$ variables. We can limit the number of variables to $O(m)$ by instead considering the combined utilizations of sets of tasks. This is supported by one final property.

Lemma 3. *The derivative of the inflated utilizations (at any level) of a **set** of tasks with respect to their allocated LLC area size is non-increasing.*

Proof: As stated earlier, the sum of non-convex functions is non-convex. □

While some of the assumptions made here concerning execution times may result in over-approximations of such execution times so that these assumptions are met, we show later via a schedulability study that our LLC allocation methods yield substantial schedulability improvements.

PET- and overhead-based constraints. Consider the hypothetical utilization plot shown in Fig. 6 for U'^C_C with respect to some integer number of allocated LLC ways W. We can construct such a plot from execution-time measurements, known task periods, and known values for b^B and b^C. In Fig. 6, we create a set of lines from each pair of adjacent data points, using the standard two-point line formula

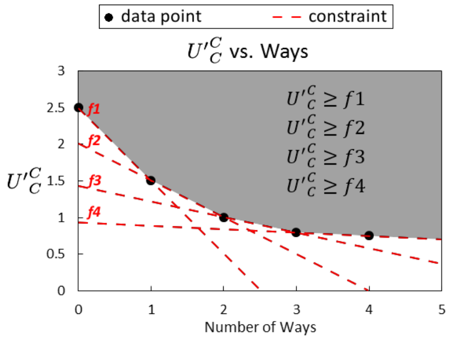


Figure 6: Converting utilizations derived from execution time data to linear constraints. The shaded region is the continuous region in which utilization will be constrained in our LP.

$f(x) = f(x_0) + (x - x_0)(f(x_0 + 1) - f(x_0))$. This is the formula for the line that contains the points $f(x_0 + 1)$ and $f(x_0)$. We can describe the value of f over a continuous domain with LP variables $\hat{f}$ and $\hat{x}$ constrained by such lines. Let x^{max} denote the maximum value of x for which we have a data point for $f(x)$. A value can be determined for $\hat{f}$ by solving the following LP.

$$\begin{aligned}
&\text{minimize } \hat{f}\\
&\text{subject to:}\\
&\forall x \in \{0, 1, ..., x^{max} - 1\} :\\
&\hat{f} \geq f(x) + (\hat{x} - x)(f(x+1) - f(x))\\
&\hat{f} \geq 0\\
&0 \leq \hat{x} \leq x^{max}
\end{aligned}$$

If this LP produces an integer value for $\hat{x}$, then $\hat{f}$ will equal $f(\hat{x})$. In the case considered in Fig. 6, our discrete function is $U'^C_C(W)$, for which we define the LP variable $\hat{U}^C_C$ to describe $U'^C_C(W)$ over a continuous domain.

$$\hat{U}^C_C \geq U'^C_C(W) + (\hat{W}_C - W)(U'^C_C(W+1) - U'^C_C(W))$$

Note that $\hat{W}_C$ is the only variable in the right-hand-side expression above, *i.e.*, this is a linear expression. We define similar LP variables $\hat{U}^A_{A,p}$, $\hat{U}^B_{A,p}$, $\hat{U}^C_{A,p}$, $\hat{U}^B_{B,p}$, and $\hat{U}^C_{B,p}$ for the inflated utilizations of Levels A and B for each core p. To simplify the constraints presented for these variables, we introduce the following shorthand functions for lines constructed from data points for utilizations.

$$\mathbb{U}^\ell_{A,p}(\hat{W}_{A,p}, W) \equiv$$
$$U^\ell_{A,p}(W) + (\hat{W}_{A,p} - W)(U^\ell_{A,p}(W+1) - U^\ell_{A,p}(W))$$

$$\mathbb{U}^\ell_{B,p}(\hat{W}_{B,p}, W) \equiv$$
$$U'^\ell_{B,p}(W) + (\hat{W}_{B,p} - W)(U'^\ell_{B,p}(W+1) - U'^\ell_{B,p}(W))$$

$$\mathbb{U}^C_C(\hat{W}_C, W) \equiv$$
$$U'^C_C(W) + (\hat{W}_C - W)(U'^C_C(W+1) - U'^C_C(W))$$

Note that $\mathbb{U}^{\ell}_{B,p}(\hat{W}_{B,p}, W)$ and $\mathbb{U}^{C}_{C}(\hat{W}_{C}, W)$ depend on inflated utilizations, while $\mathbb{U}^{\ell}_{A,p}(\hat{W}_{A,p}, W)$ does not. This is because of the different manner in which CRPDs are dealt with at Level A, as discussed earlier.

To handle Level-A inflations, we incorporate them into the constraints for utilization variables separately from data points, as shown in the following constraint set. Letting S^{max} denote the total number of colors of the considered LLC ($S^{max} = 16$ on our ARM platform), we let $I^{B}_{A,p}$ and $I^{C}_{A,p}$ denote the needed Level-A inflations on core p at Levels B and C, respectively, with respect to our LP variables for ways.

$$I^{B}_{A,p} \equiv \frac{E^{B}(\hat{W}_{B,p}, S^{max}/m)}{T^{min}_{A,p}}$$
$$I^{C}_{A,p} \equiv \frac{E^{C}(\hat{W}^{O}_{p}, S^{max}/m)}{T^{min}_{A,p}}$$

S^{max}/m gives the total number of colors allocated to Levels A and B on each core. Note, that at Level B, an inflation is applied even without overlap. This conservatively models Level-A inflations at Level B to avoid non-linear constraints. This completes the LP variable relations needed to describe constraints derived from measured execution times.

Constraint Set 2. *The linear constraints for utilization variables based on task execution-time data with CRPD overheads added are as follows.*

$$\begin{aligned}
&\forall W \in \{0, 1, ..., W^{max} - 1\} :: \\
&\quad \hat{U}^{C}_{C} \geq \mathbb{U}^{C}_{C}(\hat{W}_{C}, W) \\
&\quad \forall p \in \{1, ..., m\} :: \\
&\qquad \hat{U}^{B}_{B,p} \geq \mathbb{U}^{B}_{B,p}(\hat{W}_{B,p}, W) \\
&\qquad \hat{U}^{C}_{B,p} \geq \mathbb{U}^{C}_{B,p}(\hat{W}_{B,p}, W) \\
&\qquad \hat{U}^{A}_{A,p} \geq \mathbb{U}^{A}_{A,p}(\hat{W}_{A,p}, W) \\
&\qquad \hat{U}^{B}_{A,p} \geq \mathbb{U}^{B}_{A,p}(\hat{W}_{A,p}, W) + I^{B}_{A,p} \\
&\qquad \hat{U}^{C}_{A,p} \geq \mathbb{U}^{C}_{A,p}(\hat{W}_{A,p}, W) + I^{C}_{A,p}
\end{aligned}$$

Modeling h and H. To construct an LP that applies all schedulability conditions to task systems, linear constraints are also required for quantities specific to Expression (10). We let $\hat{h}$ and $\hat{H}$ be our LP variables for $h'(W)$ and $H'(W)$, respectively. Our constraints for these variables are constructed in a similar fashion to the constraints for utilization variables. Values for $h'(W)$ and $H'(W)$ are determined from measured execution times for each integer number of ways W allocated to Level C after inflation. Linear constraints are then constructed from adjacent data points.

This requires $h'(W)$ and $H'(W)$ to be non-convex as well. These data functions, in fact, *are* non-convex under our Execution Time Assumption. Let $\tau_h(W)$ denote the Level-C task with highest inflated utilization when W ways are allocated to Level C. If $\tau_h(W)$ does not change with W, then the non-convexity of $h'(W)$ follows trivially, because the utilizaton of τ_h is non-convex. Non-convexity still holds if $\tau_h(W)$ changes with W. Consider way values W and $W + 1$ such that $\tau_h(W) \neq \tau_h(W + 1)$. This implies that the derivative of $\tau_h(W + 1)$'s utilization is greater than the derivative of $\tau_h(W)$'s utilization at W. Hence, the derivative of h' is greater at $W + 1$ than W, and h remains non-convex at $W + 1$. By similar logic, $H'(W)$ is guaranteed to be non-convex.

Constraint Set 3. *The linear constraints for $\hat{h}$ and $\hat{H}$ based on measured task-set utilizations with CRPD overheads are as follows.*

$$\begin{aligned}
&\forall W \in \{0, 1, ..., W^{max} - 1\} :: \\
&\quad \hat{H} \geq h'(W) + (\hat{W}_{C} - W)(h'(W + 1) - h'(W)) \\
&\quad \hat{h} \geq H'(W) + (\hat{W}_{C} - W)(H'(W + 1) - H'(W))
\end{aligned}$$

Schedulability constraints. To fully characterize all constraints on utilizations and ways, we must include the schedulability constraints based on Expressions (7)–(10). Expression (10) is a strict inequality. We apply a small decrease, $\epsilon = 10^{-6}$ to its right-hand side to change this.

Constraint Set 4. *The linear constraints based on the schedulability conditions* (7)–(10) *are as follows.*

$$\begin{aligned}
&\forall p :: \hat{U}^{A}_{A,p} \leq 1 \\
&\forall p :: \hat{U}^{B}_{A,p} + \hat{U}^{B}_{B,p} \leq 1 \\
&\sum_{p=1}^{m} \left(\hat{U}^{C}_{A,p} + \hat{U}^{C}_{B,p}\right) + \hat{U}^{C}_{C} \leq m \\
&\sum_{p=1}^{m} \left(\hat{U}^{C}_{A,p} + \hat{U}^{C}_{B,p}\right) + (m - 1)\hat{h} + \hat{H} \leq m - \epsilon
\end{aligned}$$

Linear program for LLC allocation. From Lemmas 1-3 and the discussion above, we have the following.

LP Allocation Theorem. *An allocation scheme that produces the minimum Level-C utilization for a task set while maintaining all schedulability conditions can be determined by solving the following LP.*

$$\begin{aligned}
\text{minimize } & \sum_{p=1}^{m} \left(\hat{U}^{C}_{A,p} + \hat{U}^{C}_{B,p}\right) + \hat{U}^{C}_{C} \leq m \\
\text{subject to: } & \text{Constraint Sets 1-4} \\
& \text{Non-negativity constraints on all variables.}
\end{aligned}$$

The objective of minimizing total Level-C utilization is used here as a greedy heuristic because this reduces tardiness bounds for Level-C tasks [6]. However, this objective function serves a secondary purpose. Recall from our discussion of the LP variable $\hat{f}$ that if $\hat{f}$ is minimized, then it will equal $f(\hat{x})$ at integer values of $\hat{x}$. Minimizing total Level-C utilization ensures that utilization variables reflect actual system utilization values determined from PETs when LLC area variables are at integer values.

Approximations. Under certain scenarios, the LP above will converge to integer way values for many task systems. Consider the LP with Constraint Set 4 removed. The remaining constraints on Level-C utilizations from Constraint Set 2 intersect at integer way values. Level-C utilization is minimized at the intersection of linear constraints, and the LP will thus converge to integer values. However, the way-parameter values that minimize Level-C utilization may violate schedulability conditions (7), (8), or (10). In this scenario, the LP *with* Constraint Set 4 may not converge to integer way values.

If the program solution does not return integer values, we can round way values, or convert the LP to a mixed-integer LP (MILP). In Sec. 5, we compare schedulability for rounded LP-based LLC allocation sizes to schedulability for MILP-based LLC allocation sizes. Note that non-integral LP-based LLC allocation sizes are not necessarily guaranteed to be nearest to integral LLC allocations that are schedulable when schedulable allocations exist. As shown in Sec. 5, however, the schedulability loss due to rounded LP-based programming is fairly small in many cases.

5 Evaluation

We now discuss experiments we conducted to assess the impact of our LP-based LLC allocation approach on task-set schedulability.

Experimental framework. We randomly generated task sets and determined the fraction that were schedulable on our target hardware platform, the quad-core ARM Cortex A9 machine mentioned earlier, the LLC of which has 16 ways and 16 colors. To determine the benefit of LP-based LLC allocation relative to other alternatives, we compared our approach to two fixed LLC allocation schemes. We call the first alternative the *default scheme* because it is the one considered in the companion paper mentioned earlier [12]. For any task set, it allocates eight ways and 16 colors (half the LLC space) to Level C, and splits the remaining LLC space evenly into per-core areas; core p's area consists of four colors and four ways (1/8 the LLC space) and is shared by all Level-A and -B tasks on core p. We call the second alternative the *bypass scheme*. Under it, all Level-A and -B tasks bypass the LLC entirely (they have a zero-area LLC allocation), and the Level-C tasks share the entire LLC without restriction. This scheme is reflective of the intuition that the provisioning of Level-A and -B tasks might be so conservative that they derive almost no benefit from the LLC.

We consider both the original LP formulation of our approach, where the returned ways must be rounded if non-integral, and the corresponding MILP formulation. We compare these two formulations both in terms of accuracy and runtime performance.

Task-set categories. Our schedulability study consisted of 81 separate experiments, each pertaining to a distinct category of task sets. For each experiment, task sets were generated first for the bypass scheme and then per-task execution times were altered to obtain corresponding task sets for

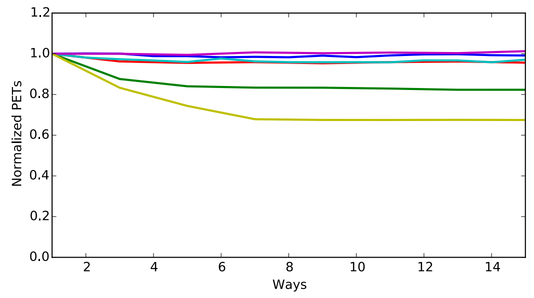
	Type	A	B	C
Level-C Util. Alloc. (%)	C-heavy	[10, 30)	[10, 30)	remainder
	B-heavy	[20, 30)	[40, 60)	remainder
	AB-moderate	[35, 45)	[35, 45)	remainder
Period (ms)	Light	{3,6}	{6,12}	[3,33)
	Contrasting	{3,6}	{96,192}	[10,100)
	Heavy	{48,96}	{96,192}	[50,250)
Task Util.	Light	[0.001,0.03)	[0.001,0.05)	[0.001,0.1)
	Moderate	[0.02,0.1)	[0.05,0.2)	[0.1,0.4)
	Heavy	[0.1,0.3)	[0.3,0.5)	[0.5,0.9)
WS Load Time	Light	[0.01, 0.1)	[0.01, 0.1)	[0.01, 0.1)
	Moderate	[0.1, 0.25)	[0.1, 0.25)	[0.1, 0.25)
	Heavy	[0.25, 0.5)	[0.25, 0.5)	[0.25, 0.5)

Table 1: Task set categories.

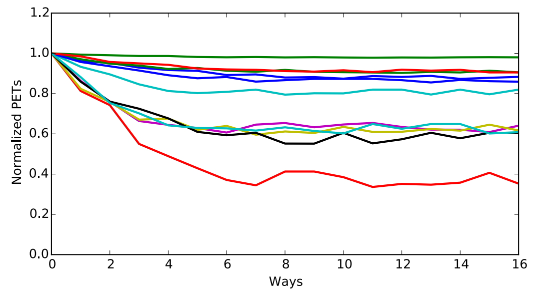
the other considered schemes. Task sets were generated for the bypass scheme by first selecting the distributions to use in generating task parameters. These distribution choices, which are listed in Table 1, are as follows.

- **Selection 1:** Choose the distributions to use in determining the fraction of the overall Level-C utilization that is consumed at each criticality level. There are three overall choices here, as shown in Table 1. For example, under the C-heavy choice, the Level-C utilization of each of Levels A and B will be between 10% and 30% (exclusive) of the total Level-C utilization, with the remainder going to Level C.
- **Selection 2:** Choose the distributions to use in generating task periods. Again, there are three overall choices here, as shown in the table.
- **Selection 3:** Choose the distributions to use in generating Level-C utilizations for individual tasks. Again, there are three overall choices.
- **Selection 4:** Choose the distributions to use in determining the time required to load a task's working set (WS) from memory. The load time is expressed as a percentage of the task's Level-C execution time. As before, there are three overall choices, as shown in the table. For example, under the Light choice, the load time for any task will be between 1% and 10% (exclusive) of its overall Level-C execution time.

Generating task sets. In generating task sets for the bypass scheme, we allowed the total Level-C utilization to vary from 0.1 to 6.1 in steps of 0.2. For each total Level-C utilization in this range, we evaluated between 100 and 2,000 randomly generated task sets to estimate mean schedulability with 95% confidence to within a confidence interval of 0.05. Each individual task set was generated by randomly selecting relevant parameters using the distributions chosen above. If a given task is a Level-A (Level-B) task, then it also requires a Level-A and -B (Level-B) utilization. A task's Level-B utilization (if required) was defined to be s times its Level-C utilization, where the scaling factor s ranges uniformly within $[10/3, 20/3]$. This choice of scaling factor was based on measurement data from our ARM platform. A task's Level-A utilization (if required) was defined to be 1.5 times its Level-B utilization. This reflects



(a) Measured PETs.



(b) Generated PETs.

Figure 7: Measured vs. generated PETs with respect to ways.

the previously mentioned 50% inflating of Level-B PETs to obtain Level-A PETs. Each task's per-level PETs are implicitly determined by its period and per-level utilizations. Given these PETs, and the WS load times selected using the distribution discussed under Selection 4 above, we determined a task's actual WS size (WSS) using documented memory-access latencies for our ARM machine. WSSs affect how PETs vary with LLC-allocation schemes, which we discuss next.

The above process yields a task set for the bypass scheme. To obtain a corresponding task set for the other schemes, we merely have to scale PETs (and hence utilizations) to reflect allocated LLC areas. The scaling factors we used in determining PETs for the non-bypass schemes were based on measurement data obtained from our ARM platform, with an adjustment applied for WSS-related reasons. In particular, a task's WSS determines the maximum amount of cache space it uses. As we allocate additional LLC space for a task beyond its WSS, its execution time should not change significantly. We account for this when determining scaled PETs.

Fig. 7(a) shows some of the measured Level-B PET data we collected on our ARM platform, and Fig. 7(b) shows some of the PETs obtained via our generation process. Note that our generated PETs are only approximately non-convex. To apply our LP techniques, any non-convexity must be masked by upper bounding. Such upper bounding introduces pessimism in the analysis.

In order to complete the specification of a task set, its Level-A and -B tasks must be assigned to cores. We obtained such an assignment by using the worst-fit-decreasing bin-packing heuristic to first assign Level-A tasks, based on their Level-A utilizations under the bypass scheme, and then to assign Level-B tasks using the remaining capacities, based on the Level-B utilizations under the bypass scheme.

This concludes our overview of the task-set generation process we used. This process is described in much greater detail in an appendix.

Results. Our study resulted in 81 schedulability plots. Due to space constraints, we discuss only the plots shown in Fig. 8, which reflect generally seen trends across all plots; the other plots can be found in an online appendix (available at `http://www.cs.unc.edu/~anderson/papers.html`). In insets (a)–(c) of Fig. 8, schedulability plots are given for three categories of task systems; these plots depict the fraction of the generated task systems deemed schedulable, as a function of overall Level-C utilization under the bypass scheme, for each considered LLC allocation method. Insets (d)–(f) give corresponding probability distributions for the number of allocated ways at each level under a MILP-based allocation. For example, inset (f) indicates that for the task-system category considered in inset (c), 10–14 ways tended to be allocated to Level C, 3–7 to Level B, and 3–7 to Level A. We make the following observations from this data.

Obs. 1. Using MILP- and LP-based LLC allocations significantly improved schedulability in approximately a third of the tested task-system categories, increasing schedulability by 20-50% in some cases, and by a factor of two in others. For the other categories, only moderate improvements resulted.

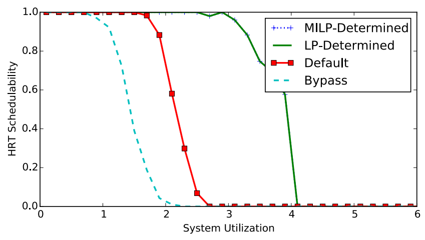
Fig. 8(a) depicts one of several categories that exhibited significant improvements. Fig. 8(d) suggests that, for this category, the usage of LP techniques adapts LLC allocations to account for the high CPRD overheads expected in this case. As seen, little to no LLC cache space is given to any level, suggesting that CRPD overheads outweigh any performance gains provided by the LLC. Fig. 8(b) depicts one of several categories that yielded only mild improvements. Fig. 8(e) suggests that, for this category, the usage of LP techniques results in LLC allocations that vary dramatically. The low impact of LLC allocation choice on schedulability is not surprising, since this is a light memory utilization task set, and therefore task utilizations are not very sensitive to LLC area size.

Obs. 2. The usage of LP-based allocations resulted in little to no degradation in schedulability in comparison to MILP-based allocations in all tested task-system categories.

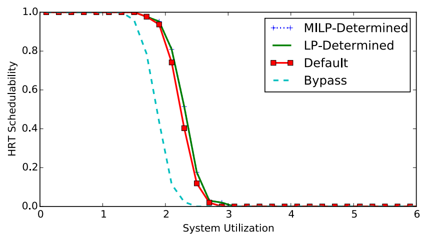
Insets (a)–(c) of Fig. 8 show very little difference in schedulability results for these two algorithms. Due to the similarities of the LP and MILP algorithms, both produce similar LLC-allocation schemes for each task set. These schemes have similar effects on schedulability as a result.

Obs. 3. While MILPs have exponential time complexity, the actual runtime performance of our MILP allocation scheme was roughly equivalent to that of our LP scheme.

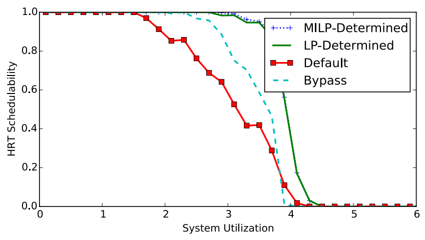
Across all task systems in all experiments, our MILP scheme took 151 ms on average and 1377 ms in the worst



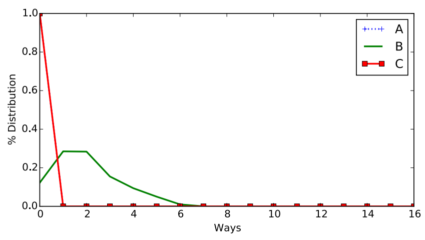
(a) C-Heavy, Contrasting, Light, Light.



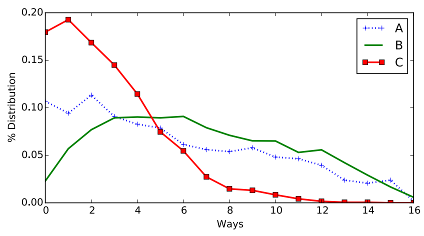
(b) AB-Moderate, Long, Medium, Light.



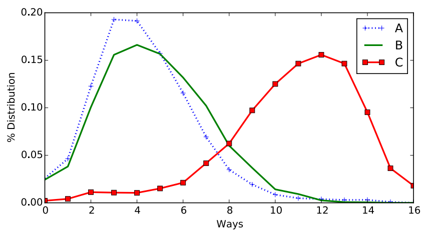
(c) C-Heavy, Long, Medium, Medium.



(d) Figure (a) allocation.



(e) Figure (b) allocation.



(f) Figure (c) allocation.

Figure 8: Schedulability and LP-based LLC allocation results for three categories of task sets. Category choices are listed in order of categories.

case, while or LP scheme took 148 ms on average and 315 ms in the worst case.

6 Conclusion

To our knowledge, this is the first paper to consider the optimization problem of allocating LLC areas among tasks of different criticality levels in a mixed-criticality multicore system. We addressed this problem in the context of MC^2 through the use of LP techniques that take into account both schedulability and CRPD overheads. We demonstrated the efficacy of these techniques by presenting an experimental evaluation that shows that their usage can have significant benefits from a schedulability viewpoint. Our LP techniques achieved similar schedulability improvements as our MILP variant. In our experiments, the LP and MILP variants proved to have similar runtime overheads.

In the LLC allocation problem considered herein, only the number of allocated ways is viewed as a variable. The number of allocated colors (which determine the allocated sets) can be varied as well. However, varying both parameters creates an optimization problem that is difficult to address using LP techniques. Nonetheless, this more general optimization problem warrants further study.

References

[1] S. Altmeyer, R. Douma, W. Lunniss, and R.I. Davis. Evaluation of cache partitioning for hard real-time systems. In *ECRTS '14*.

[2] T. Baker and A. Shaw. The cyclic executive model and ada. In *RTSS '88*.

[3] B. Bui, M. Caccamo, L. Sha, and J. Martinez. Impact of cache partitioning on multi-tasking real time embedded systems. In *RTCSA '08*.

[4] A. Burns and R. Davis. Mixed criticality systems – a review. Technical report, Department of Computer Science, University of York, 2014.

[5] M. Campoy, A. Ivars, and J. Mataix. Static use of locking caches in multitask preemptive real-time systems. In *Real-Time Embedded Sys. Workshop '01*.

[6] U. Devi and J. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. *The Journal of Real-Time Systems*, 38:133–189, Feb. 2008.

[7] J. Erickson, B. Ward, and J. Anderson. Fair lateness scheduling: Reducing maximum lateness in G-EDF-like scheduling. *Real-Time Systems*, 50(1):5–47, 2014.

[8] J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson. RTOS support for multicore mixed-criticality systems. In *RTAS '12*.

[9] J. Herter, P. Backes, F. Haupenthal, and J. Reineke. CAMA: A predictable cache-aware memory allocator. In *ECRTS '11*.

[10] R. Kessler and M. Hill. Page placement algorithms for large real-indexed caches. *ACM Trans. on Comp. Sys.*, 10:338–359, 1992.

[11] H. Kim, A. Kandhalu, and R. Rajkumar. A coordinated approach for practical OS-level cache management in multi-core real-time systems. In *ECRTS '13*.

[12] N. Kim, B. Ward, M. Chisholm, C.-Y. Fu, J. Anderson, and F.D. Smith. Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning. In submission.

[13] D. Kirk. SMART (strategic memory allocation for real-time) cache design. In *RTSS '89*.

[14] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu. A software memory partition approach for eliminating bank-level interference in multicore systems. In *ICPACT '12*.

[15] A. Mills and J. Anderson. A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand. In *RTCSA '11*.

[16] M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos. Mixed criticality real-time scheduling for multicore systems. In *ICESS '10*.

[17] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS '07*.

[18] B. Ward, J. Herman, C. Kenna, and J. Anderson. Making shared caches more predictable on multicore platforms. In *ECRTS '13*.

[19] H. Yun, R. Mancuso, Z. Wu, and R. Pellizzoni. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicoore platforms. In *RTAS '14*.

Appendix A

In this appendix we describe our process for generating LLC-area dependencies for task utilization at each criticality level. We start by generating task systems with a single execution time cost, τ.cost, for each task. To simulate LLC-area dependencies, we applied a separate non-increasing, non-concave function, $\mathcal{F}_i^\ell(W)$, over the number of allocated ways W to the utilization of task τ for criticality level ℓ:

$$u_i^\ell(W) \equiv \frac{\tau_i.cost}{T_i} \cdot \mathcal{F}_i^\ell(W).$$

We also have an upper-bounding function $\overline{\mathcal{F}}_i^\ell(W) \geq \mathcal{F}_i^\ell(W)$ that is non-convex. This upper-bounding function is used for our linear programs.

Task execution times may have varying sensitivities to LLC area size based on a task's memory use behavior. To describe more precisely different categories of LLC area size dependencies, we must first describe how the function $\mathcal{F}_i^\ell(W)$ was generated. We start by generating a non-convex function $F_i^\ell(W)$ for each task τ_i at criticality level ℓ, where $F_i(0) = 1.0$, where ℓ is Level B or C. The Level-A function is generated last using our Level-B function, since Level-A utilizations should be 50% greater than Level-B costs for each number of ways W.

To constrain the initial decrease in the function from ways 0 to 1, we define parameters $\check{D}^\ell \leq 1$ and $\hat{D}^\ell < \check{D}^\ell$ for each criticality level ℓ. Letting $rand(x, y)$ denote a function that returns a random value in the interval $[x, y]$, we define

$$F_i^\ell(1) = rand(\hat{D}^\ell, \check{D}^\ell).$$

We assume that, prior to overhead inflation, task execution times should not increase as LLC-allocation size increases. Hence, we ensure F_i^ℓ is non-increasing by upper-bounding each value $F_i^\ell(W)$ where $W > 1$ by $\overline{F}_i^\ell(W)$, defined as follows:

$$\overline{F}_i^\ell(W) \equiv F_i^\ell(W-1).$$

For the function to be non-convex, each value $F_i^\ell(W)$ where $W > 2$ must be lower-bounded by another function $\underline{F}_i^\ell(W)$. This function is determined by taking the decrease in function value from $W-2$ to $W-1$ - that is, $F_i^\ell(W-2) - F_i^\ell(W-1)$ - and ensuring that the decrease from $W-1$ to W is not greater

$$\underline{F}_i^\ell(W) \equiv F_i^\ell(W-1) - (F_i^\ell(W-2) - F_i^\ell(W-1)).$$

We may additionally want to limit these bounds further such that the function "flattens out" slower or faster. We use parameters $\check{\omega} \leq 1$ and $\hat{\omega}^\ell < \check{\omega}^\ell$ to define more restricted bounds $\check{F}_i^\ell(W)$ and $\hat{F}_i^\ell(W)$. The parameters $\hat{\omega}^\ell$ and $\check{\omega}^\ell$ are used to calculate interpolations between the values of $\check{F}_i^\ell(W)$ and $\hat{F}_i^\ell(W)$

$$\check{F}_i^\ell(W) \equiv \underline{F}_i^\ell(W) + \check{\omega}^\ell(\overline{F}_i^\ell(W) - \underline{F}_i^\ell(W)),$$

$$\hat{F}_i^\ell(W) \equiv \underline{F}_i^\ell(W) + \hat{\omega}^\ell(\overline{F}_i^\ell(W) - \underline{F}_i^\ell(W)).$$

We now define all remaining values of $F_i(W)$ as follows:

$$F_i^\ell(W) = rand(\hat{F}_i^\ell(W), \check{F}_i^\ell(W)), \quad W > 1.$$

$\mathcal{F}_i^\ell(W)$ is derived from $F_i^\ell(W)$ by considering two modifications to the curves generated so far.

Modificaton 1. These curves have no lower bound. Our scaling function for a task's utilization should have a lower bound (we would not expect the LLC to ever reduce utilizations by 95%, for instance). For each criticality level, we define a lower bound on utilization scaling $\mathcal{F}_{min}^\ell$.

Modification 2. Each task τ has a WSS $\tau.wss$. As stated in Sec. 5, we expect execution times, and thus utilizations, to not decrease significantly as a task is allocated addition LLC space beyond its WSS. We denote W_i^{wss} to be the least number of ways required for a task to fit its entire working set into the allocated area. This completes the functions and parameters we need to derive a non-convex function with which to upper-bound task utilizations. For Level B, we define

$$\begin{aligned} \overline{\mathcal{F}}_i^B(W) &= max(F_i^B(W), \mathcal{F}_{min}^B), && \text{for } W \leq W_i^{wss} \\ \overline{\mathcal{F}}_i^B(W) &= \overline{\mathcal{F}}_i^B(W-1), && \text{for } W > W_i^{wss} \end{aligned}$$

At each level, we want τ.cost for $\tau \in \Gamma_\ell$ to represent the bypass-scheme utilization at level ℓ. Remember, in this scheme, Level B is given 0 ways on all cores and Level C is given W^{max} ways. Hence, for $\tau_i \in \Gamma_B$, $\mathcal{F}_i^B(0)$ should be 1.0, but for $\tau_i \in \Gamma_C$, $\mathcal{F}_i^C(W^{max})$ should be 1.0. For Level C, we first define a helper function $\mathcal{M}_i^C(W)$ derived in a similar fashion as $\overline{\mathcal{F}}_i^B(W)$ and then normalize this function with respect to $\mathcal{M}_i^C(W^{max})$.

$$\begin{aligned} \overline{\mathcal{M}}_i^C(W) &= max(F_i^C(W), \mathcal{F}_{min}^C), && \text{for } W \leq W_i^{wss} \\ \overline{\mathcal{M}}_i^C(W) &= max(F_i^C(W), \mathcal{F}_{min}^C), && \text{for } W > W_i^{wss} \\ \overline{\mathcal{F}}_i^C(W) &= \frac{\overline{\mathcal{M}}_i^C(W)}{\overline{\mathcal{M}}_i^C(W^{max})} \end{aligned}$$

$\mathcal{F}_i^\ell(W)$ is derived by applying slight variations to values of $\overline{\mathcal{F}}_i^\ell(W)$. We randomly pick up to eight different way values $\{W_1, ...W_{tot}\}, W_{tot} \leq 8$ between one and fifteen. We determine the range $R \equiv \overline{\mathcal{F}}_i^\ell(0) - \overline{\mathcal{F}}_i^\ell(16)$ over which the non-convex function varies, and for each way value $W \in \{W_1, ...W_{tot}\}$, we assign $\mathcal{F}_i^\ell(W)$ the following:

$$\mathcal{F}_i^\ell(W) = \overline{\mathcal{F}}_i^\ell(W) - rand(0, 0.05) \cdot R$$

For all other way values, $\mathcal{F}_i^\ell(W) = \overline{\mathcal{F}}_i^\ell(W)$.

At this point, we have a unique set of parameters

$$P^B \equiv \{\hat{D}^B, \check{D}^B, \hat{\omega}^B, \check{\omega}^B, \mathcal{F}min^B\}$$

for generating worst-case execution time behavior at Level B and a similar set of parameters

$$P^C \equiv \{\hat{D}^C, \check{D}^C, \hat{\omega}^C, \check{\omega}^C, \mathcal{F}min^C\}$$

for describing average-case execution time behavior at Level C. For our paper, we chose $\hat{D}^{\ell}$ and $\check{D}^{\ell}$ values in the range [0.9,0.97) to produce utilizations that initially declined steadily as ways increase from 0. We chose $\hat{\omega}^{\ell}$ and $\check{\omega}^{\ell}$ in the range [0, 0.15) to ensure initially steady declines in utilization tend to not flatten out as way sizes increase.