# WCET-Aware Optimization of Shared Cache Partition and Bus Arbitration for Hard Real-Time Multicore Systems

Man-Ki Yoon, Jung-Eun Kim, and Lui Sha
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, USA
{mkyoon, jekim314, lrs}@illinois.edu

May 2011
(Revised in August 2011)

**Abstract**

In recent years, multicore processors have been receiving a significant amount of attention from avionic and automotive industries as the demand for high-end real-time applications drastically increases. However, the unpredictable worst-case timing behavior that mainly arises from shared resource contention in current multicore architectures has been the biggest stumbling block for a widespread use of multicores in hard real-time systems. A great deal of research efforts have been devoted to address the issue. Among others, the development of a new multicore architecture has emerged as an attractive solution because it is possible to eliminate the sources of unpredictable interferences in the first place, or at least to turn them into predictable ones. Accordingly, this opens a new possibility of system-level optimizations with multicore-based hard real-time systems. To address this issue, this study proposes a new perspective of WCET model called *tunable WCET*, in which the WCET of a task is partitioned into *fixed execution time* and *tunable delay*. Our tunable WCET model enables WCET-aware shared resource allocation/arbitration by elastically deforming the tunable delays of tasks. For this, we also propose novel shared bus arbitration and cache partitioning methods called *harmonic round-robin bus scheduling* and *two-level cache partitioning*. We present a mixed integer linear programming (MILP) formulation as the solution to the optimization problem of tunable WCETs. Our experimental results show that the proposed methods can significantly lower overall system utilization.

## I. Introduction

Multicores have been increasingly adopted by chip manufacturers as a solution to scale the performance beyond the thermal and power walls. Intel has ten core chips on the market [1], and Sun has introduced SPARC T3 processor which has sixteen hyper-threaded cores [2]. Furthermore, a research chip by Intel has as many as 80 cores that can perform more than one trillion floating-point operations per second [3]. While most of the current multicore systems are targeted for servers and desktops, it is also expected that real-time embedded systems and cyber-physical systems will follow the trend in the near future as the demand for high-end real-time applications is rapidly growing. For instance, Freescale's QorIQ P4080 processor [4] and the ARM11 MPCore processors [5] are receiving wide attention from avionic and automotive industries.

However, one of the major obstacles in using multicore processors for these domains is that the execution time of applications can vary noticeably depending on how physical resources, such as cache and system interconnect, are shared and/or contended between co-scheduled tasks on the system. For example, shared cache is one of the most critical and contended resources on multicores [6], [7], [8].

If two tasks start to evict each others cache line out of the shared cache to bring in its own data, they will take much longer to complete than when each of them runs alone or is scheduled together with a non-interfering task with smaller working set. In addition to this, shared bus is another major source of indeterminism in multicore processors [9], [10], [11]. Similar to the situation of shared cache, if two or more tasks on different cores try to access the shared bus simultaneously for cache fetches, some tasks will experience longer delays than others due to the contention. If we consider, furthermore, I/O traffics injected into the system through DMA (Direct Memory Access), the problem becomes more serious since the traffics may impose additional delays on the core-initiated bus accesses.

This unpredictability of the timing behavior of current multicore architecture is a huge barrier, especially for safety-critical systems in which the predictability of the worst-case temporal behavior is of primary importance. One of solutions to this kind of problem is to develop an analysis method that can precisely estimate the worst-case execution times of applications in the presence of shared resource contentions [12], [13], [14], [15], [16]. The assumptions made in the existing analyses are commonly too restrictive, however, and thus the results are often very pessimistic or not even applicable directly to the current multicore architectures. The more serious problem is that, as multicore architectures become more complex, the correlation among the sources of unpredictabilities becomes much larger than before. Hence it becomes harder or even impossible to achieve accurate estimation of WCET with the existing analyses.

Due to such fundamental limitations of the analytic methods, hardware modification of multicore systems has emerged as an attractive and viable solution [17], [18], [19], [20]. While the analytic methods try to analyze interferences caused by resource contentions, the new multicore architectures focus on eliminating such interferences in the first place for higher predictability. However, some of the architectures support only certain type of programming language [17], and/or require software applications to be modified in order to take advantage of the hardware modifications [19]. Moreover, some architectures experience poor average-case performance due to the lack of certain performance-support features such as multithreading [18]. In this point of view, the multicore architecture proposed by Paolieri et al. [20], which this study is based on, provides a good architectural foundation for future hard real-time multicore systems.

### *A. Motivating Hard Real-Time Multicore Architecture*

In [20], Paolieri *et al.* introduced a new hard real-time multicore architecture in which accesses to shared resources, such as shared bus or cache, are controlled by hierarchical bus arbiters (refer to Figure 1 in [20]). The architecture employs *round-robin* as the shared bus arbitration policy; all cores are fairly given the equal chance to access the bus. Through the nature of round-robin policy, the maximum delay that a bus request of a task can suffer from others is bounded by the total number of hard real-time tasks ready to be executed at the same time. In addition to this bus access interference, they also analyzed shared cache interference with regard to two factors - *bank conflict interference* and *storage conflict interference*, both of which are the causes of unpredictable and unanalyzable worst-case timing behavior of shared cache in multicore systems. The maximum delay due to bank conflict interferences for a task is similarly bounded by the number of hard real-time tasks. They also addressed cache partitioning techniques which eliminate storage conflict interference by splitting cache space into separately assigned pieces - banks or columns.

This architecture does not have the limitations of other multicore architectures. First of all, since resource contentions are resolved by hardware arbiters, it does not require any modification on applications' source code, and for a similar reason, it does not impose any restrictions on programming language or OS. Furthermore, it can properly support multithreading of applications with the help of Intra-Core Bus Arbiters (ICBAs), thus it does not compromise system performance in applications' execution times.

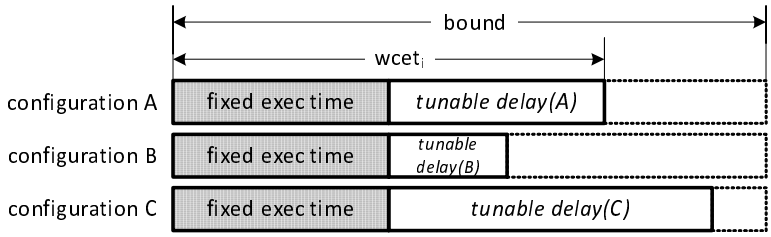



Fig. 1: Tunable WCET model.

### *B. Tunable WCET and Its Optimization*

Paolieri's multicore architecture summarized in the previous subsection provides a high degree of temporal predictability of the applications' WCET; each core or task has its exclusive spatial and temporal partitions to access the shared resources.

While this makes a WCET analysis much easier by eliminating the potential sources of resource contentions, one major limitation is that the resources may have limited capacities to accommodate a given workload. Recall that every application, i.e., task, is assigned to *private* cache banks or columns in their architecture. Bank-level partitioning requires as many banks as the number of tasks in the system. Column-level partitioning may resolve the capacity problem, however tasks may experience additional delays in accessing the banks. Therefore a proper partitioning method which can fully utilize the shared cache while minimizing the bank conflict interferences is needed.

Another possible way of improvement is the use of *application-aware* bus scheduling. While the pure round-robin scheduling can easily bound the worst-case bus access delay, it may be inefficient in that every bus access has to wait for the same amount of worst-case delay regardless of application characteristics; memory-bound tasks are likely to access the shared bus more intensively than others. If we give more frequent chances to such tasks by lengthening others' worst-case waiting times, we can achieve an enhanced overall efficiency, such as lower system utilization. This advantage can be magnified, especially if the system mainly consists of a subclass of numerical real-time tasks, such as signal or image processing applications, that has few branches and the cache footprints rarely change from period to period.

In order to address the above problems, this study proposes a new perspective of WCET model called *tunable WCET*, in which WCET of a task is partitioned into two parts - *fixed execution time* and *tunable delay*, as shown in Figure 1. While most traditional WCET analyses have focused only on how to precisely estimate WCETs, our tunable WCET model enables system-level optimization for certain purposes by elastically deforming shared resource configurations. In particular, we focus on the two major sources of inter-core interference, on-chip interconnect and shared last-level cache. In this study, we investigate how different configurations of bus arbitration and cache partition could affect tasks' tunable delay and on how such different interference sources are correlated. In order to achieve this goal, we adopt Paolieri's multicore architecture (refer to Section I-A or [20]) and propose novel bus arbitration and cache partitioning methods called *harmonic round-robin* and *two-level cache partitioning*, respectively. By our harmonic round-robin (HRR) bus arbitration policy, we can vary the delays incurred for bus accesses of different tasks on different cores and hence realize application-aware bus scheduling. Similarly, our two-level cache partitioning scheme maps banks to cores and columns to tasks in such a way that the delays due to bank access conflict are minimized with the help of our harmonic round-robin bus scheduling. One might easily expect that bus scheduling and cache partitioning, in conjunction with task allocation, are so highly dependent upon each other that it is not straightforward to find out the optimal system configuration for a given set of tasks. Accordingly, we shall also present a *mixed integer linear programming* (MILP) formulation as the solution to the optimization problem of tunable WCETs. As will be seen later in this

report, with our proposed methods, the solutions of the optimization problem are always better than, or at least identical to, those that can be achieved with Paolieri's architecture in terms of minimum achievable system utilization.

The rest of this report is organized as follows: In Section II, we introduce our harmonic round-robin bus arbitration and two-level cache partitioning methods and then formally define the problem of tunable WCET optimization. In Section III, we describe in detail the tunable WCET model and its analysis. In Section IV, we present the mixed integer linear programming (MILP) formulation for our tunable WCET optimization problem. In Section V, we present the experimental results obtained by our MILP optimization. Sec. VI summarizes the related work. Finally, Section VII concludes this report.

## II. Optimization of Tunable WCET

In this section, we first introduce the proposed harmonic round-robin bus arbitration and two-level cache partitioning methods and then describe how these affect our tunable WCET model in the perspective of system-level optimization.

### A. Harmonic Round-Robin Bus Arbitration

As briefly mentioned in Section I-A, the maximum delay that a task can suffer due to bus interference is bounded by the total number of cores with Paolieri's pure round-robin arbitration policy [1]. That is, the worst-case scenario occurs when each task running at every core tries to access the shared bus at the same time. Because the maximum number of slots that a task needs to wait for the next available slot is same with the number of cores in the system, the upper-bound of bus interference delay is $N^{\mathbb{C}} \cdot L_B$, where $N^{\mathbb{C}}$ is the number of cores and $L_B$ is the bus latency. This means that every task has the same upper-bound of bus access delay regardless of how their execution characteristics are different. Accordingly, a more memory-intensive or high-utilization task has to wait for the same amount of delay as the other less memory-intensive or low-utilization tasks. This is inefficient in that the same amount of bus request delay of different tasks affect a certain performance metric differently. For example, suppose that task A in core 1 and task B in core 3 suffer the same worst-case bus delay of $4 \cdot L_B$ as shown in Figure 2(a) and that we want to minimize the overall system utilization. If their period is 50 but the total numbers of cache accesses are 500 and 100, respectively, then the contributions of their bus access delays to the system utilization are

$$u_A^{bus} = \frac{500 \cdot 4 \cdot L_B}{50} \quad \text{and} \quad u_B^{bus} = \frac{100 \cdot 4 \cdot L_B}{50},$$

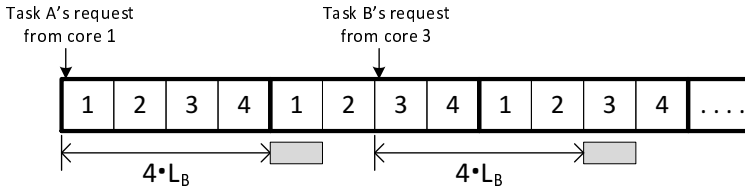
respectively. As another example, suppose now that the period of task A is 10 and the number of its cache accesses is 100. Then, the contributions of their bus access delays to the system utilization become

$$u_A^{bus} = \frac{100 \cdot 4 \cdot L_B}{10} \quad \text{and} \quad u_B^{bus} = \frac{100 \cdot 4 \cdot L_B}{50},$$

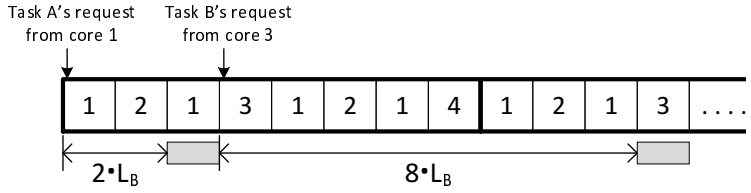
respectively. In both cases, $u_A^{bus}/u_B^{bus}$ is 5, that is, task A affects the system utilization 5 times more than task B. Now let us suppose that core 1 is allowed to access the bus every 2 slots and core 3's period is increased to 8 slots, as shown in Figure 2(b). Then, $u_A^{bus}$ and $u_B^{bus}$ become

$$u_A^{bus} = \frac{100 \cdot 2 \cdot L_B}{10} \quad \text{and} \quad u_B^{bus} = \frac{100 \cdot 8 \cdot L_B}{50},$$

[1] In [20], the authors consider non hard real-time tasks as well as hard real-time tasks. In this report, however, we consider only hard real-time tasks, and we assume that a bus request should arrive before each designated time slot to be granted to send the request.




(a) Pure RR (4,4,4,4).




(b) HRR (2,4,8,8).

Fig. 2: The impact of bus schedule and task allocation on bus access delay.

respectively. Accordingly, the net contribution, i.e., $u_A^{bus} + u_B^{bus}$, is reduced from

$$\frac{2400 \cdot L_B}{50}$$

to

$$\frac{1800 \cdot L_B}{50}.$$

As can be observed from this example, by giving more frequent slots to cores on which memory-intensive or high-utilization tasks run, we can lower the overall system utilization. If the tasks are not pre-assigned to specific cores, we can further reduce it by grouping such tasks and assigning them to the more *prioritized* cores. Thus, as a method of core prioritization in bus scheduling, we propose *Harmonic Round-Robin* arbitration policy, which can be defined as follows:

$$HRR \triangleq (N^{\mathbb{C}}, T^{min}, T^{max}, T^{RR}, \mathbb{P}),$$

where $\mathbb{P}$ is a set of HRR periods denoted by $\{T_1, T_2, \cdots, T_{N^{\mathbb{C}}}\}$, $\forall_j\ T^{min} \leq T_j \leq T^{max}$, and $T^{RR}$ is the hyper period of $\mathbb{P}$, i.e., the length of one round. The periods of cores *harmonize* with each other if and only if they satisfy the following conditions:

$$\forall_{1 \leqq j \leqq N^{\mathbb{C}}-1}\ \frac{T_{j+1}}{T_j} \in \mathbb{N} \text{ (positive integer)} \quad and \quad \sum_{j=1}^{N^{\mathbb{C}}} \frac{1}{T_j} = 1. \tag{1}$$

Because HRR periods are bounded by $[T^{min}, T^{max}]$, only finite number of harmonic sets can be made within the given range. Here the first condition enforces every $T_{j+1}$ to be a positive integer multiple of $T_j$ so that the periods to be in non-decreasing order, $T_1 \leq T_2 \leq \cdots \leq T_{N^{\mathbb{C}}}$. In addition, the second condition further requires that the scheduling table has to be a complete round-robin. By *complete* we mean if we can create a scheduling table in which every core $j$ has slots every $T_j$. For example, the schedules shown in Figure 2 are complete round-robins. However, $(2, 4, 4, 8)$ is not a complete one since there is no way to build such a table, that is, some core $j$ cannot be guaranteed to be able to access to

the bus at every $T_j$. $(2, 4, 8, 16)$ is not complete either, however it can be transformed into a complete one by adjusting $T_4$ from 16 to 8. In fact, this holds for any sets of periods $\{T_j\}$, where $\sum_{j=1}^{N^{\mathbb{C}}} \frac{1}{T_j} < 1$, satisfy the first condition of 1. Once a set of HRR periods $\{T_1, T_2, \cdots, T_{N^{\mathbb{C}}}\}$ satisfying Condition (1) is obtained, we can create the unique corresponding harmonic round-robin scheduling table of length $T^{RR} = T_{N^{\mathbb{C}}}$ by constraints **C9–C11** in Section IV. For example, Figure 2 shows the scheduling tables corresponding to $(4, 4, 4, 4)$ and $(2, 4, 8, 8)$, respectively.

With our harmonic round-robin arbitration, we can also achieve the same argument of [20] that the maximum bus access delay of a task can be obtained without the knowledge of other real-time tasks. Also, due to its regular pattern, the bus access delay analysis is done straightforwardly, as will be explained in Section III-A. Furthermore, more importantly, it helps to reduce bank conflicts, in conjunction with our two-level cache partitioning scheme which will be introduced in the following section.

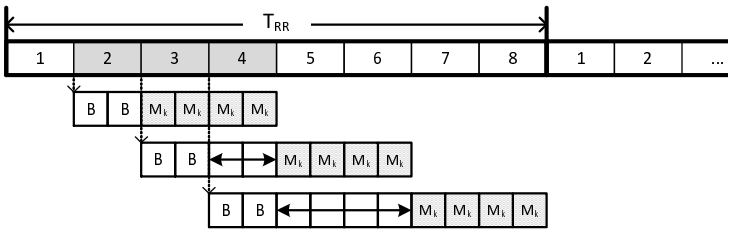
### *B. Two-Level Cache Partitioning*

As explained in Section I, the shared last-level cache is another major source of inter-core interferences in multicore systems. As a solution to eliminate such an interference, the authors of [20] consider a column-level cache partitioning method called *columnization* [21]. In columnization, a task suffers a *bank conflict delay* if another one is already accessing the same bank as shown in Figure 3(a), which illustrates the worst-case scenario of bank conflicts among core 2, 3, and 4 sharing a same bank, $k$. As a way of avoiding such an interference, we can allocate a set of private banks to each task, which is called *bankization* in the same paper. Because no two tasks can share any bank, the bank conflict delay totally disappears. This is restrictive, however, in that the number of banks of the shared cache should be at least the number of tasks.

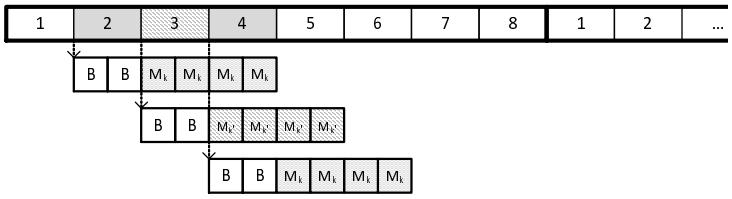
However we can reduce or even eliminate bank conflicts without giving private banks to every task by considering the bus schedule. An example of such situations is shown in Figure 3(a) and (b). From the figure we can observe that core 2 and core 4 are able to share a bank without suffering any bank conflict delay since the bank access request from core 4 begins after core 2 completes its request. However, core 2 and 3 cannot share the bank without suffering any bank conflict delay since their bank requests will be overlapped. For the same reason, core 3 and core 4 cannot share any bank without suffering the delays. However, as shown in Figure 3(b), if we assign another bank $k'$ to core 3, any of those cores does not experience bank conflict delays. Moreover, in an extreme case, by assigning bank $k$ to core 1, 3, 5, and 7 and bank $k'$ to core 2, 4, 6, and 8, we can totally eliminate bank conflict delays. However, if the number of banks needed by core 1, 3, 5, and 7 exceeds one, we need to assign more banks to those cores. As long as the number of available banks is sufficient, nevertheless, all tasks can still avoid suffering bank conflict delays.

However one may encounter a situation where no more banks are available for core 3 in Figure 3(a). In that case, we can eliminate or reduce their bank conflict delays by scheduling the requests with a harmonic round-robin schedule, as shown in Figure 3(c). The harmonic round-robin schedule in this example enables the bank access requests from core 2, 3, and 4 to be *pipelined* so that any bank requests are not overlapped.

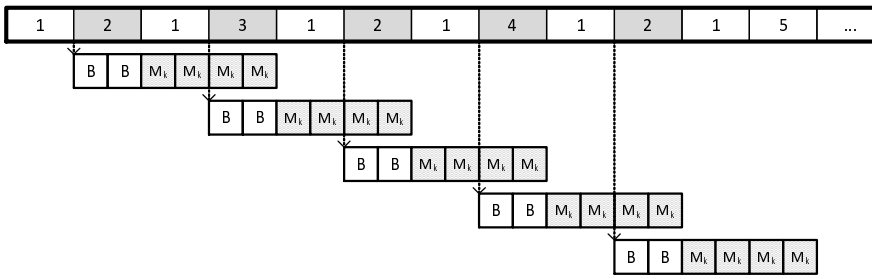
Another important factor that influences on bank conflict delay and bank-sharing is the ratio of bank access latency, $L_M$, to the bus access latency, $L_B$. To make the point clear, let us consider Figure 3(c) again. Since the bank latency, which is 4 in this example, is two times of the bus latency, i.e, 2, and each slot of the cores sharing the bank is apart from each other by $2 \cdot L_B$, there is no bank conflict delay among them. However once the bank latency becomes 5, then those cores start experiencing bank conflict delays, as shown in Figure 3(d). Furthermore, the delays could be accumulated even beyond the first round. In this case, the accumulated delay will affect the same slots in the next round making each delay increase without a bound, which we call *unbounded bank conflict delay* problem (see Figure 4 for an example).



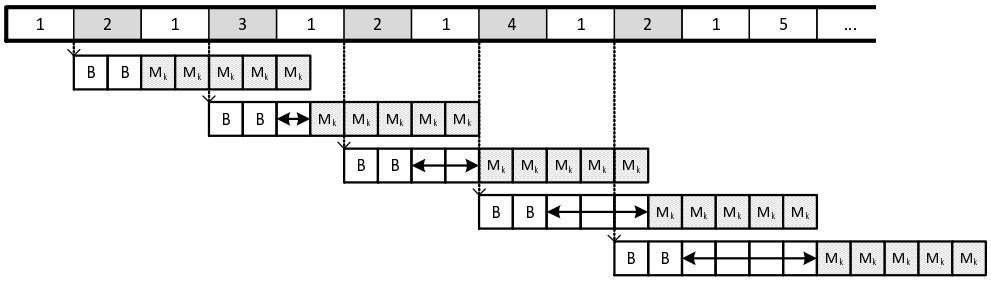

(a) Pure RR. Core *2*, *3*, and *4* share Bank *k*. Core *3* and *4* suffer bank conflict delays.




(b) Pure RR. Core *2* and *4* share Bank *k*, and Core *3* uses Bank *k'*. By assigning a different bank to Core *3*, all bank conflict delays become eliminated.




(c) HRR (2,4,24,24,24,24,24,24). Core *2*, *3*, and *4* share Bank *k*. $L_M=2\cdot L_B$. The bank conflict delays can be eliminated by scheduling the bus with a harmonic round-robin arbitration.




(d) HRR (2,4,24,24,24,24,24,24). Core *2*, *3*, and *4* share Bank *k*. $L_M=(5/2)\cdot L_B$. An increase of $L_M$ causes bank conflict delays to Core *2*, *3*, and *4*.

Fig. 3: Bank conflict delays with different bus and cache configurations.

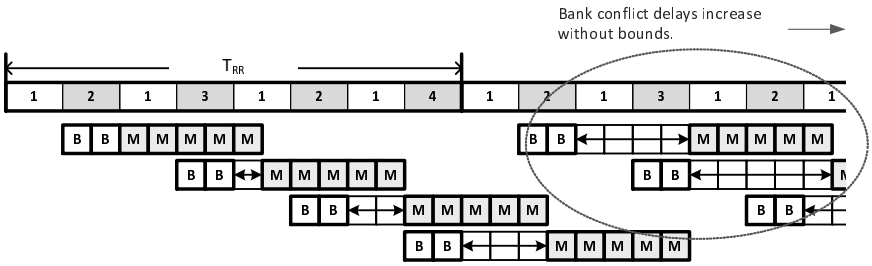



Fig. 4: An example of unbounded bank conflict delay.

To prevent such a situation, the net workload, i.e., the sum of bank access delays including latencies, going to a shared bank should be less than or equal to the hyper period of the harmonic round-robin schedule. Note that, in the worst-case, each core $j$ generates $\frac{T_{RR}}{T_j}$ bank requests within one hyper period, $T_{RR}$, and thus the condition that prevents such unbounded bank conflict delays for shared bank $k$, can be expressed as follows:

$$\sum_{\forall \text{ core } j \text{ using bank } k} \frac{L_M}{L_B \cdot T_j} \leq 1. \tag{2}$$

Thus, in the case of Figure 4, we should make one of the cores use another bank.

Sharing banks among cores may introduce unnecessary bank conflict delays if they are not properly coordinated, as illustrated in Figure 3, and the problem becomes aggravated when there are insufficient banks to be allocated to tasks. One efficient way to utilize the shared cache and thus to minimize bank-sharing is to allocate a contiguous subset of banks to each core and to allow any two cores to share at most one bank - the leftmost or the rightmost bank allocated to each core [2]. Sharing only one bank between two cores is sufficient and better than sharing multiple banks in that the latter only increases the chance of bank conflict delays. Assume that core $A$ and $B$ share two banks, e.g., $k$ and $k+1$, each of which has 10 columns. Suppose that core $A$ uses 7 columns of bank $k$ and 5 columns of bank $k+1$, and core $B$ uses the rest. This is, however, equivalent to give all columns of bank $k$ and 2 columns of bank $k+1$ to core $A$, and to let only bank $k+1$ be shared between core $A$ and $B$. That is, sharing of $m$ banks between any two cores can be transformed to single sharing. By this *core-level* partitioning, the chance of bank conflict delay can be reduced and moreover the memory address mappings can be simplified. In addition to the core-level partitioning, we subdivide each bank into several columns and then map each task to a set of contiguous private columns of the banks allocated to the core where the task runs on, as shown in Figure 5. By this *core-level* partitioning, the chance of bank conflict delay can be reduced and moreover the memory address mappings can be simplified. In addition to the core-level partitioning, we subdivide a bank into several columns and then map each task to a set of contiguous columns belonging to the banks allocated to the core where the task resides in, as shown in Figure 5. By this *task-level subpartitioning*, we can eliminate interferences due to cache thrashing among tasks in the same core. Moreover, we can prioritize the tasks even within a core in allocating their columns. Let us consider the example shown in Figure 5, where bank $k$ is shared between core $i$ and $j$. While the columns of task $B$ stretch from bank $k$ to bank $k+1$, those of task $C$ are in bank $k+1$. Since a part of bank $k$ is mapped to task $A$, it may be possible that task $B$ suffers bank conflict delays. However, bank $k+1$ of core $j$ cannot be shared with any other cores by our core-level partitioning, and hence the cache accesses from task $C$ are free from interferences caused by bank conflicts. We call such banks as

[2]It does not mean that a bank can be shared by at most two cores. Instead, three or more cores can share a same bank as long as they do not cause the unbounded bank conflict delay problem.

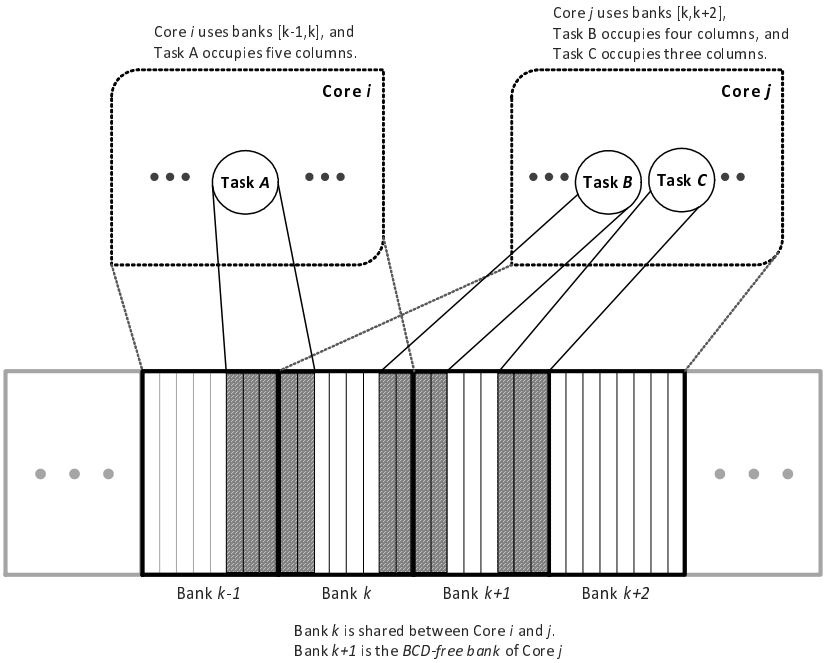



Fig. 5: Two-level cache partitioning.

*BCD-free banks* - a set of banks in which tasks using the columns of the banks cannot experience any bank conflict delays [3]. Accordingly, a more memory-intensive, high-utilization, or higher-criticality task can benefit from using the columns in a BCD-free bank.

As has been seen, our two-level cache partitioning method does not rely only on tasks' cache usages. Instead, both of bus configuration and task allocation also affect the decision of how to map core-to-banks and task-to-columns and thus tasks' bank conflict delays. In Section III-B, we will analyze in detail how bank conflict delays are affected by different configurations of bus, cache partition, and task allocation.

### C. System Model

We consider a multicore system that consists of $N^{\mathbb{C}}$ homogeneous cores denoted by $\mathbb{C} = \{C_1, C_2, \cdots, C_{N^{\mathbb{C}}}\}$. This system has an unified shared cache $\mathbb{B}$ which is partitioned into $N^{\mathbb{B}}$ banks $\{B_1, B_2, \cdots, B_{N^{\mathbb{B}}}\}$, each of which is again divided into $N^W$ columns (see Figure 5 for an example). Thus the cache has total $N^{\mathbb{X}} = N^{\mathbb{B}} \cdot N^W$ columns which are denoted by $\mathbb{X} = \{X_1, X_2, \cdots, X_{N^{\mathbb{X}}}\}$. Each core uses contiguous blocks of banks, however any two cores cannot share two or more banks. Cache requests from the cores are delivered to the shared cache via a shared bus arbitrated by a harmonic round-robin policy, which is introduced in Section II-A (see Figure 2(b)).

On that system, we assume $N^{\Gamma}$ real-time tasks, which is denoted by $\{\tau_1, \tau_2, \cdots, \tau_{N^{\Gamma}}\}$. Each task $\tau_i$ is assigned to one of the cores and is periodically executed on the core with the execution time of $e_i$ and the period of $p_i$. Also, each $\tau_i$ uses a set of cache columns as the place to store/load its instructions and data. Accordingly, a task $\tau_i$ can be represented by

$$\tau_i = (e_i, p_i, N_i^{\mathbb{X}}, N_i^M),$$

[3]Note that a core may not have its BCD-free banks if only one or two banks are allocated to the core and all the banks are shared with other cores. If a core is given three or more banks, there always exists a BCD-free bank.

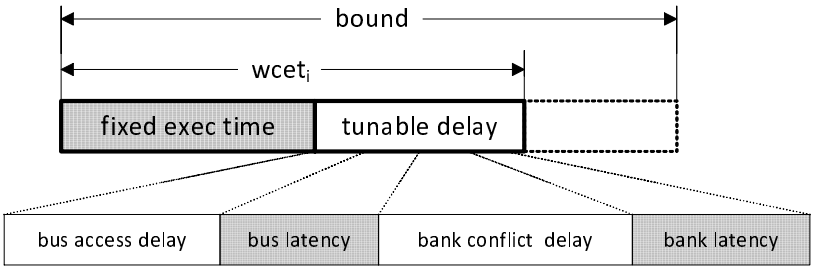



Fig. 6: Delay components of proposed tunable WCET model.

where $N_i^{\mathbb{X}}$ is the number of cache columns that $\tau_i$ would need during its execution and $N_i^M$ is the total number of cache accesses to the columns. Here we assume that the size of each cache request is fixed and thus there is no variance in the latencies of bus access, $L_B$, and bank access, $L_M$. Each task is allowed to own only the columns in the banks allocated to the core at which the task is located. However no two tasks can share a column even if they are in the same core, as explained in Section II-B. With these constraints, we further assume that there is no task migration, and the number of required cache columns and that of cache accesses are pre-profiled by a static analysis.

### *D. Tunable WCET*

Figure 6 illustrates the rationale behind our tunable WCET model proposed in this study. In this model, the worst-case execution time of task $i$ is partitioned into two parts - *fixed execution time* and *tunable delay*. The fixed execution time of a task is the maximum time duration that the task could take to execute instructions over its critical path. On the other hand, the tunable delay is the sum of the delays incurred for all of the memory accesses over the same path, and it can be variable according to the configurations of bus and cache, as described in Section II-A and II-B. Since the delay factors that affect a memory operation are the bus and cache, the tunable part is subdivided into bus access delay and cache access delay. It should be noted here that the latencies of accesses to the both are fixed. Now let $L$ be the sum of the fixed latencies, $d_i$ be the variable delays, and $e_i$ be the fixed execution time of task $i$. Then, without loss of generality, the worst-case execution time of task $i$ can be defined as follows:

$$wcet_i = e_i + N_i^M \cdot (L + d_i), \tag{3}$$

where $N_i^M$ is the number of task $i$'s cache accesses, and $L = 2 \cdot L_B + L_M$ [4].

One may argue that the critical path of, and thus $N_i^M$ of, $\tau_i$ can be changed according to the variable delays. That is, the critical path cannot be derived without knowing HRR schedule and bank assignments in advance. While this is true in general, the analysis of tunable WCET and its optimization will become significantly more complex if we take a variable critical path into account. Thus, we assume in this paper that there exists an execution path whose fixed execution time, $e_i$, is so long enough that other paths cannot be longer than the obtained critical path even if they would experience maximum possible delays.

Our tunable WCET model enables optimization of bus scheduling and cache partitioning as mentioned before. Here the optimization objective can vary; one may want to shorten some certain application's worst-case execution time or want to minimize the number of required cache banks and so on. In this study, among various objectives, we focus on the optimization of overall system utilization. In optimizing

[4]We assume the bus is full-duplex as was assumed by [20], which means that requests from cores to the shared cache do not conflict with the ones fetched from the cache. Therefore, only the core-to-cache requests can be delayed. Note that the cache-to-core requests are not shown in any figure of this report for the simplicity of illustrations.

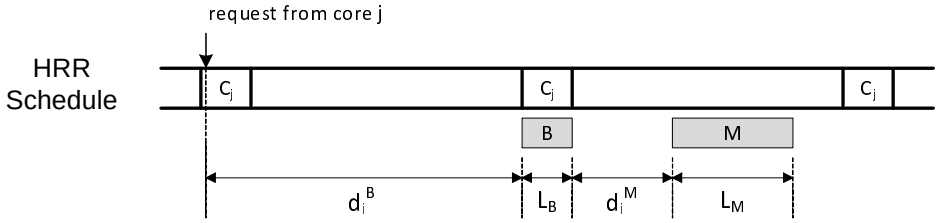



Fig. 7: Worst-case tunable delay of a cache access.

such a global metric, it is important to consider how to utilize limited resources. For example, reducing the bus access delay of a task and/or allocating private banks to a task may increase the delays of other tasks. Moreover, the optimization factors are correlated to each other, for example the shorter a core's HRR period, the more private banks the core needs to occupy. Accordingly, it is important to understand how to parameterize the optimization model. In Section III, we will describe in detail how different configurations of our harmonic round-robin arbitration and two-level cache partitioning methods affect tunable WCET of tasks.

### *E. Problem Description*

For a given set of $N^{\mathbf{\Gamma}}$ real-time tasks $\{\tau_1, \tau_2, \cdots, \tau_{N^{\mathbf{\Gamma}}}\}$, $N^{\mathbb{C}}$ homogeneous cores $\{C_1, C_2, \cdots, C_{N^{\mathbb{C}}}\}$, and $N^{\mathbb{B}}$ banks $\{B_1, B_2, \cdots, B_{N^{\mathbb{B}}}\}$ consisting of $N^{\mathbb{X}}$ columns $\{X_1, X_2, \cdots, X_{N^{\mathbb{X}}}\}$, our problem is to find the optimal task assignments, harmonic round-robin schedule, and core-to-banks and task-to-columns mappings that minimize the overall system utilization, i.e.,

$$\text{Minimize} \sum_{i=1}^{N^{\mathbf{\Gamma}}} \frac{wcet_i}{p_i}. \tag{4}$$

Low system utilization is generally preferred in system development because 1) the lower-utilized system can be more utilized by accommodating additional tasks. The other way around is also true - 2) the same set of tasks can be implemented with lower-speed cores, which can reduce the unit cost of production. In Section IV, we will present our mixed integer linear programming (MILP) formulation for this optimization problem.

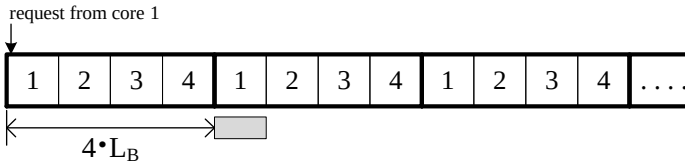
## III. Tunable WCET Analysis

As described in Section II-D, the worst-case execution time of a task is modeled as the sum of fixed execution time and tunable delay. The latter, in turn, is divided into fixed latencies and variable delays incurred during bus and bank accesses, as illustrated in Figure 7 [5]. Now let $d_i^B$ and $d_i^M$ be the upper-bound of bus access delay and that of bank conflict delay, respectively. Then, Equation (3) can be redefined as follows:

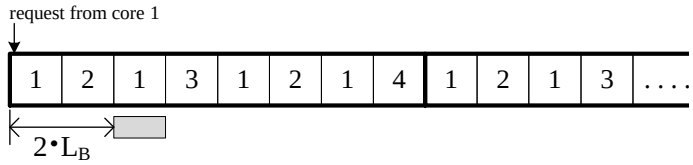
$$wcet_i = e_i + N_i^M \cdot \{L + (d_i^B + d_i^M)\}. \tag{5}$$

In this model, only $d_i^B$ and $d_i^M$ are variable depending on bus schedules, cache partitions, and task allocations. Accordingly, in the rest of this section, we describe in detail how such configurations affect the worst-case execution time of a task and how these are correlated.

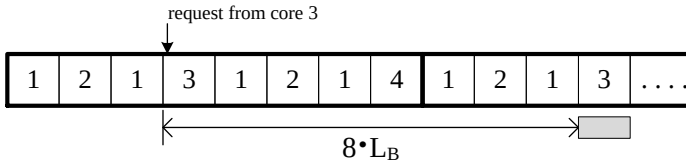
[5]Note that cache-to-core requests are not shown.




(a) Pure RR (4,4,4,4). Task *i* is allocated to core *1*.




(b) HRR (2,4,8,8). Task *i* is allocated to core *1*.




(c) HRR (2,4,8,8). Task *i* is allocated to core *3*.

Fig. 8: Different bus access delays with different bus schedules.

### *A. Bus Access Delay $d_i^B$*

The upper-bound of a bus access delay is defined as the maximum length of time that a bus request could take until it is granted. In pure round-robin, as illustrated in Figure 8(a), a bus access delay is upper-bounded by $N^{\mathbb{C}} \cdot L_B$ and is independent of which core the task is allocated to [6]. In a harmonic round-robin scheduling, on the other hand, task allocation is a delay factor since different cores may have different HRR periods. For instance, let us suppose that the bus is scheduled by HRR of $(2, 4, 8, 8)$ (Figure 8(b) and (c)). If task $i$ is allocated to core 1, a bus access from the task can be delayed at most two bus slots in the case where it just missed the beginning of core 1's turn. However, if the task is not in core 1, but rather in core 3, $d_i^B$ becomes $8 \cdot L_B$. Accordingly, the upper-bound of a bus access delay of task $i$ can be expressed as the following simple equation:

$$d_i^B = T_j \cdot L_B, \tag{6}$$

where $T_j$ is the HRR period of core $j$ to which task $i$ is allocated.

### *B. Bank Conflict Delay $d_i^M$*

As explained in Section II-B, a bank conflict delay occurs when a task tries to access a bank that is already being accessed by another request (see Figure 3). Hence it is highly likely for a task to suffer such a delay if the core which the task is in shares a bank with many other cores. Furthermore, if each core is near each other in HRR scheduling table, the chances become even greater. However, if they are far enough apart from each other, as illustrated in Figure 3c, we can reduce or even eliminate such bank

[6]Recall that we assume that a bus request should arrive before each turn in order to be granted to send the request.

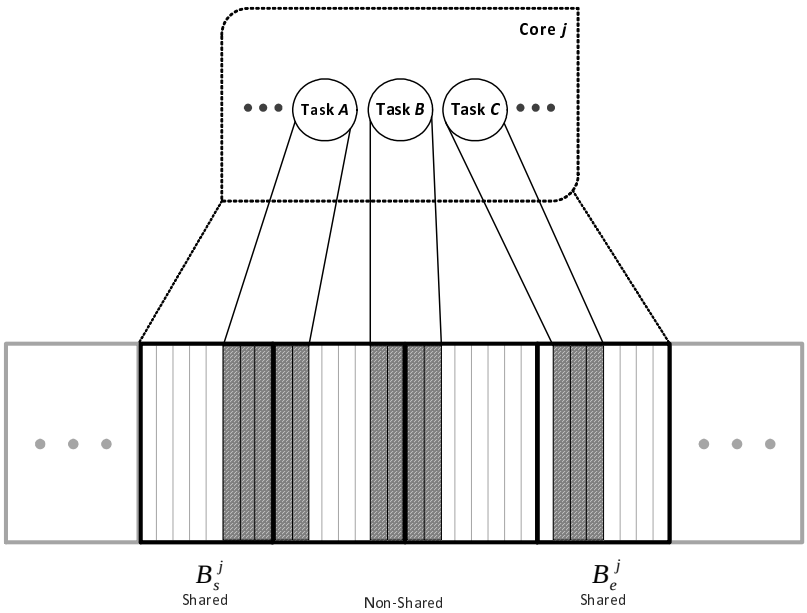



Fig. 9: Task to column mappings.

conflict delays. Accordingly, task allocation, HRR scheduling, and cache partitions should be taken into account together in analyzing the upper-bound of bank conflict delays.

Now let us suppose that task $i$ is allocated to core $j$. From our system model in Section II-C, the core uses a contiguous subset of banks. Let us denote the banks as

$$\mathbf{B_j} = \{B_s^j, B_{s+1}^j, \cdots, B_{s+n_j^B-1}^j\},$$

where $n_j^B$ is the number of cache banks required by core $j$, which depends on the total number of cache columns required by the all tasks in the core. Recall that, again by our system model, only the leftmost and rightmost banks, i.e., $B_s$ and $B_{s+n_j^B-1}$, can be shared with others. For simplicity of notations, let us denote $B_{s+n_j^B-1}$ by $B_e$.

It is now important to note that task $i$ could suffer a bank conflict delay *if and only if* the columns that the task occupies are a part of the shared banks. In other words, if the all columns of the task are in the *BCD-free banks* of the core, the task can never suffer any bank conflict delays, as explained in Section II-B. In this case, $d_i^M$ is always 0. Otherwise, the upper-bound of bank conflict delays could be non-zero, and it depends on in which banks the columns are located and whether the banks are shared with others or not. Let us consider the following cases, which is also shown in Figure 9:

Case 1. a part or all of the columns reside in $B_s^j$, but not in $B_e^j$,
Case 2. a part or all of the columns reside in $B_e^j$, but not in $B_s^j$, and
Case 3. none of the columns reside in either $B_s^j$ or $B_e^j$.

The upper-bound of bank conflict delays can be different in each case. For example, in Figure 9, Task $B$ is free from any bank conflict delays since all of its columns are in the BCD-free banks of Core $j$ (Case 3). On the other hand, Task $A$ and Task $C$ use some of the columns in Bank $B_s^j$ (Case 1) and $B_e^j$ (Case 2), respectively, and thus could experience bank conflict delays. Here different bank accesses of Task $A$ may suffer different delays since its columns stretch from a shared bank to a non-shared bank. However, we assume that the target address of, and thus the target column index of, an access is not

known in analyzing the WCET of a task. Thus, as long as at least one column of a task is located in a shared bank, we consider that every access of the task goes to the shared bank. Otherwise, the analysis would become intractable since we need to consider which access goes to which bank and when.

Meanwhile, the columns of a task may stretch from $B_s^j$ to $B_e^j$, especially if the number of available banks is small or/and the task requires many columns (not shown in Figure 9).

Case 4. the columns stretch from $B_s^j$ to $B_e^j$.

In this case, an access from the task could go to one of $B_s^j$ and $B_e^j$, or other non-shared banks. Thus, as mentioned before, all accesses of the task are assumed to experience the worst-case delay, which will depend on the bank conflict delay of $B_s^j$ and $B_e^j$.

Now let us define $D_s^j$ and $D_e^j$ as the upper-bounds of bank conflict delays that a task could experience by accessing bank $B_s^j$ and $B_e^j$, respectively. It is important to know here that $D_s^j$ and $D_e^j$ are independent of other tasks. That is, any task in core $j$ can suffer a same amount of $D_s^j$ ($D_e^j$) if at least one column of a task resides in $B_s^j$ ($B_e^j$). To identify whether task $i$ could access a bank, let us first denote the cache columns occupied by task $i$ as

$$\mathbf{X_i} = \{X_k^i, X_{k+1}^i, \cdots, X_{k+N_i^{\mathbb{X}}-1}^i\},$$

where $N_i^{\mathbb{X}}$ is the number of columns required by task $i$. Then, Case 1 – 4 can be restated as the following conditional expressions:

Case 1. $idx(X_k^i) \leq idx(B_s^j) \cdot N^W$ AND $idx(X_{k+N_i^{\mathbb{X}}-1}^i) \leq (idx(B_e^j) - 1) \cdot N^W$,
Case 2. $idx(B_s^j) \cdot N^W + 1 \leq idx(X_k^i)$ AND $(idx(B_e^j) - 1) \cdot N^W + 1 \leq idx(X_{k+N_i^{\mathbb{X}}-1}^i)$,
Case 3. $idx(B_s^j) \cdot N^W + 1 \leq idx(X_k^i)$ AND $idx(X_{k+N_i^{\mathbb{X}}-1}^i) \leq (idx(B_e^j) - 1) \cdot N^W$, and
Case 4. $idx(X_k^i) \leq idx(B_s^j) \cdot N^W$ AND $(idx(B_e^j) - 1) \cdot N^W + 1 \leq idx(X_{k+N_i^{\mathbb{X}}-1}^i)$,

where $idx$ is a function that returns the index of a bank or a column. If Case 1 holds for task $i$, the bank conflict delays that the task could suffer, i.e., $d_i^M$, is upper-bounded by $D_s^j$. Similarly, $d_i^M$ for Case 2 is $D_e^j$. For Case 3, $d_i^M$ is 0 since all columns of the task are in non-shared banks. Lastly, $d_i^M$ for Case 4 is the maximum of $D_s^j$ and $D_e^j$. Accordingly, the upper-bound of bank conflict delays of task $i$ in core $j$ can be expressed by the following single equation:

$$d_i^M = \max(\delta_{j,s}^i \cdot D_s^j, \delta_{j,e}^i \cdot D_e^j), \tag{7}$$

where $\delta_{j,s}^i$ ($\delta_{j,e}^i$) is 1 if task $i$ uses $B_s^j$ ($B_e^j$) and 0 otherwise.

As has been seen, the way the shared cache is partitioned affects only whether a task would experience bank conflict delays or not and a possible set of delays, i.e., $D_s^j$, $D_e^j$, or 0. In fact, the calculation processes of $D_s^j$ and $D_e^j$ are same because it only depends on which other cores share the bank and how they are scheduled in the shared bus. Now let us denote by $D_k^j$ the upper-bound of bank conflict delays that every task in core $j$ could suffer due to using bank $k$. As will be described in the following subsection, we can calculate $D_k^j$ for every pair of core and bank by using bus schedule, i.e., HRR periods of cores, and core-to-bank mappings.

*1) Computation of $D_k^j$:* To help to understand the analysis presented in this section, let us first consider an example shown in Figure 10. The figure shows a situation where eight cores are scheduled by the HRR of $(4, 4, 12, 12, 12, 12, 12, 12)$, core 2, 5, 6, and 8 are sharing bank $k$, and each of the cores tries to access the bus at each slot making the worst-case scenario. We assume here that a task under analysis,

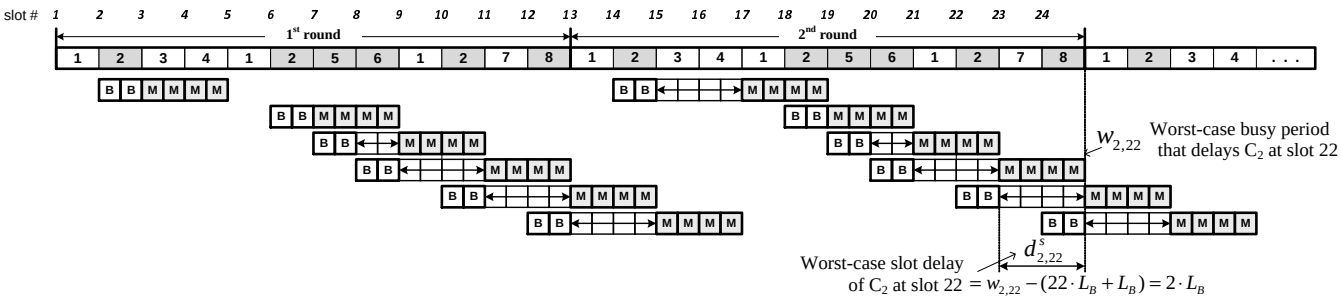



Fig. 10: The worst-case bank access scenario of core $2, 5, 6,$ and $8$ which share $B_k$. HRR : $(4, 4, 12, 12, 12, 12, 12, 12)$.

say $i$, is executing on core 2, and a part of its columns is residing in bank $k$. Now suppose that we want to find the upper-bound of bank conflict delay that task $i$ can suffer.

We can first observe from the figure that the different bank accesses from core 2 may experience variable bank conflict delays. This is because the cores have different HRR periods; core 3, 4, and 1 may affect core 2 at slot 6, and similarly core 5 and 6 may delay core 2 at slot 10, and so on. Furthermore, it depends on which cores share bank $k$ with core 2. Thus, we need to calculate each delay that task $i$ could suffer at each slot of core 2. Now let us call each such delay *slot delay* and define it as follows:

$$d^s_{j,\varphi},$$

where $j$ and $\varphi$ are the index of core and of slot, respectively. For example, in Figure 10, $d^s_{2,6}$ is 0, $d^s_{2,14}$ is $2 \cdot L_B$, and so on. If core $j$ does not use slot $\varphi$, $d^s_{j,\varphi}$ is 0.

Although the slot delays of a core can be different with each other, we can find the maximum slot delay in the second HRR round due to the following lemma:

**Lemma 1.** *Slot delay $d^s_{j,\varphi+T^{RR}}$ is always equal to $d^s_{j,\varphi}$ except for $\varphi = \phi_j$, where $\phi_j$ is the first slot index of $C_j$ in a given HRR table. For $\varphi = \phi_j$, $d^s_{j,\phi_j} \leq d^s_{j,\phi_j+T^{RR}}$ always holds.*

*Proof:* If core $j$ does not use slot $\varphi$, then $d^s_{j,\varphi+i\cdot T^{RR}}$ for all $i = 0, 1, 2, \ldots,$ is always 0. Thus, in what follows, let us consider the cases when core $j$ uses slot $\varphi$. We will prove i) $d^s_{j,\varphi} \geq d^s_{j,\varphi+T^{RR}}$ and ii) $d^s_{j,\varphi} \leq d^s_{j,\varphi+T^{RR}}$. For the simplicity of notation, let us denote $d^s_{j,\varphi}$ and $d^s_{j,\varphi+T^{RR}}$ by $d_1$ and $d_2$, respectively, as shown in Figure 11.

**i)** $d_1 \geq d_2$ : Let us assume that $d_1 < d_2$. Then, $d_2 > 0$ since $d_1 \geq 0$. Because $d_2$ is non-zero, there must exist slot $\varphi_x$ ($< \varphi + T^{RR}$), where the most recent accumulation of bank accesses begins. Now let $n_x$ be the number of bank accesses initiated in $[\varphi_x, \varphi + T^{RR} - 1]$. Then,

$$d_2 = \varphi_x + L_B + n_x \cdot L_M - (\varphi + T^{RR} + L_B) = \varphi_x + n_x \cdot L_M - \varphi - T^{RR}.$$

Now let us consider slot $\varphi_x - T^{RR}$ and denote its slot delay by $d_y$. Then, $\varphi_x - T^{RR} + L_B + d_y + n_y \cdot L_M$ is the time instant when the last bank access initiated before $\varphi$ completes. Here, $n_y$ is the number of bank accesses initiated in $[\varphi_x - T^{RR}, \varphi - 1]$, which is equal to $n_x$ because of the periodicity of HRR schedule. Now, suppose that $d_1 > 0$. Then,

$$\begin{aligned} d_1 &= \varphi_x - T^{RR} + L_B + d_y + n_y \cdot L_M - (\varphi + L_B) \\ &= \varphi_x - T^{RR} + n_x \cdot L_M - \varphi + d_y = d_2 + d_y. \end{aligned}$$

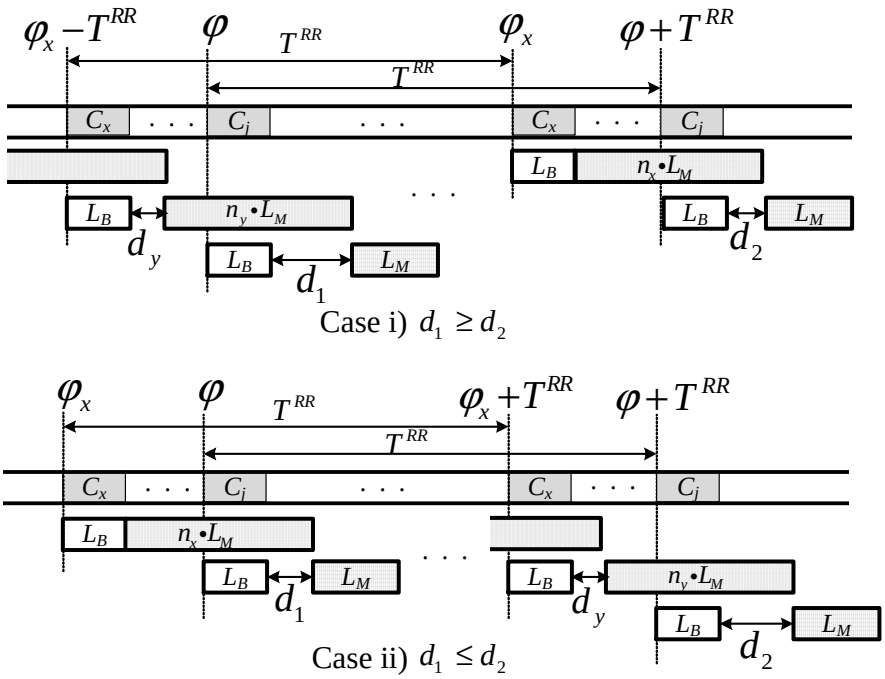



Fig. 11: Proof of Lemma 1. i) $d_1 \geq d_2$ and ii) $d_1 \leq d_2$.

The above equality results in $d_1 \geq d_2$ because $d_y \geq 0$, which contradicts the assumption that $d_1 < d_2$. Let us now consider the case where $d_1 = 0$. Then,

$$\begin{aligned} &\varphi_x - T^{RR} + L_B + d_y + n_y \cdot L_M - (\varphi + L_B) \leq 0 \\ \Rightarrow &\varphi_x - T^{RR} + n_x \cdot L_M - \varphi + d_y \leq 0 \Rightarrow d_2 \leq -d_y, \end{aligned}$$

which results in $d_2 = 0$ since $d_y \geq 0$. This contradicts the assumption that $d_1 < d_2$. Therefore, $d_1 < d_2$ never holds, concluding that $d_1 \geq d_2$.

**ii)** $d_1 \leq d_2$ : This can be proved similar to the previous one. Let us assume that $d_1 > d_2$. Then, there must exist $\varphi_x (< \varphi)$ that satisfies the following:

$$d_1 = \varphi_x + L_B + n_x \cdot L_M - (\varphi + L_B) = \varphi_x + n_x \cdot L_M - \varphi > 0.$$

Now let us consider slot $\varphi_x + T^{RR}$ and denote its slot delay by $d_y$. In this case also $n_y = n_x$. Then,

$$\begin{aligned} d_2 &= \varphi_x + T^{RR} + L_B + d_y + n_y \cdot L_M - (\varphi + T^{RR} + L_B) \\ &= \varphi_x + n_x \cdot L_M - \varphi + d_y = d_1 + d_y > 0 \end{aligned}$$

because $d_1 > 0$ and $d_y \geq 0$. Accordingly, $d_1 \leq d_2$, which contradicts our assumption that $d_1 > d_2$. Thus, $d_1 \leq d_2$.

By both i) $d_1 \geq d_2$ and ii) $d_1 \leq d_2$, we can therefore conclude that $d_1 = d_2$, i.e., $d^s_{j,\varphi} = d^s_{j,\varphi+T^{RR}}$, always holds. However, Case i) may not hold for the case where $\varphi = \phi_j$ since slot $\varphi_x - T^{RR}$ may not exist. In Case ii), on the other hand, $\varphi_x$ $(< \phi_j)$ always exists if $d_1 > 0$, and $d_1 \leq d_2$ always holds if $d_1 = 0$. Thus, we can conclude that only $d^s_{j,\varphi} \leq d^s_{j,\varphi+T^{RR}}$ holds for $\varphi = \phi_j$. ■

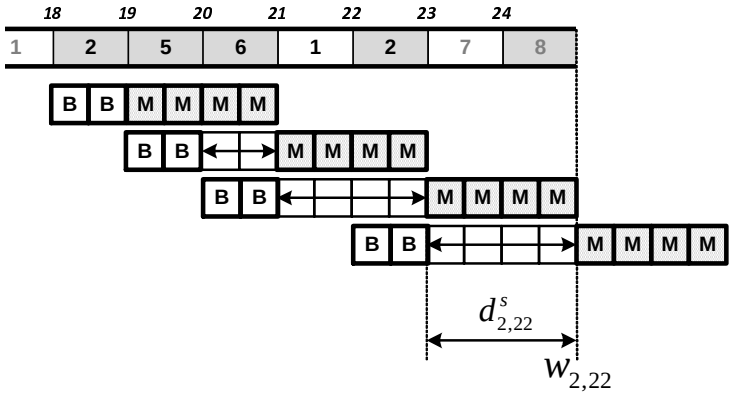



Fig. 12: An example busy period $w_{2,22}$.

By Lemma 1, we can therefore find $D_k^j$ by considering only the slots in the second round. That is,

$$D_k^j = \max(d_{j,\varphi}^s), \tag{8}$$

for $\varphi = \phi_j + T^{RR}, \phi_j + T^{RR} + T_j, \ldots, \phi_j + 2 \cdot T^{RR} - T_j$. However, $d_{j,\varphi}^s$ for the first round also need to be calculated since $d_{j,\varphi - T_j}^s$ is used when computing $d_{j,\varphi}^s$, as will be described shortly. Note that Lemma 1 does not hold in the presence of unbounded bank conflict delays.

*2) Computation of $d_{j,\varphi}^s$:* Now let us describe how to calculate each slot delay, $d_{j,\varphi}^s$. From Figure 10, we can see that $d_{j,\varphi}^s$ is affected by how much the workload, i.e., unfinished bank accesses, has been accumulated until to the point when core $j$ tries to access the shared bank. Here it is important to take into account that $d_{j,\varphi}^s$ could be delayed by not only the accesses initiated between $\varphi - T_j$ and $\varphi$, but also the ones preceding $\varphi - T_j$. For example, in Figure 10, core 2 at slot 14 would not experience any delay if one of core 5 at slot 7 and core 6 at slot 8 did not share, and thus did not access, the same bank. To model such backlog, let us define $w_{j,\varphi}$ as the time instant at which the most recent bank access completes. In finding $w_{j,\varphi}$, we need to compute the longest *busy period* of bank accesses that could delay the slot under consideration.

To help to understand how to find $w_{j,\varphi}$, let us first consider an example illustrated in Figure 12. Suppose in this example that we want to compute $w_{2,22}$. The busy period begins from core 2 at slot 18 since there is no backlog of bank access at the slot. Thus, the initial busy period of $w_{2,22}$ is

$$\begin{aligned} w_{2,22}^0 &= 18 + L_B + L_M \\ &= 21 \cdot L_B. \end{aligned}$$

Since the bank access of core 2 at 18 completes at 21, that of core 5, which tries to access the bank at 20 $(= 19 + L_B)$, is delayed by $w_{2,22}^0 - (19 + L_B)$, which results in

$$\begin{aligned} w_{2,22}^1 &= 19 + L_B + (w_{2,22}^0 - (19 + L_B)) + L_M \\ &= w_{2,22}^0 + L_M \\ &= 23 \cdot L_B. \end{aligned}$$

Likewise, the access of core 6 at the next slot is delayed by the accumulated delay, thus the busy period

grows to

$$\begin{aligned} w^2_{2,22} &= 20 + L_B + (w^1_{2,22} - (20 + L_B)) + L_M \\ &= w^1_{2,22} + L_M \\ &= 25 \cdot L_B. \end{aligned}$$

The busy period stops growing because core 1 at slot 21 does not access the shared bank, thus the final value of $w_{2,22}$ ends up being $25 \cdot L_B$. From this example, we can see that the busy period grows by $L_M$ as long as a new access has to be delayed by backlogs. However, it is possible for a busy period to be discontinued, especially if the busy period does not reach to the time instant of a new bank access. In this case, a new busy period continues from the new access.

**Lemma 2.** *The worst-case busy period that can delay core $j$ at slot $\varphi$ accessing bank $k$, $w_{j,\varphi}$, can be found by the following iterative procedure:*

$$w^0_{j,\varphi} = \begin{cases} 0 & \text{if } \varphi = \phi_j, \\ \varphi - T_j + L_B + d^s_{j,\varphi - T_j} + L_M & \text{otherwise.} \end{cases} \tag{9}$$

*If core $j$ at slot $\psi$ does not use bank $k$, $w^{i+1}_{j,\varphi} = w^i_{j,\varphi}$. Otherwise,*

$$w^{i+1}_{j,\varphi} = \begin{cases} w^i_{j,\varphi} + L_M & \text{if } \psi + L_B < w^i_{j,\varphi}, \\ \psi + L_B + L_M & \text{otherwise.} \end{cases} \tag{10}$$

*The procedure loops from $\psi = \varphi - T_j$ to $\varphi - 1$, and thus $w_{j,\varphi} = w^{T_j - 1}_{j,\varphi}$. If $\varphi = \phi_j$, the procedure loops from $\psi = 1$ to $\phi_j - 1$, and $w_{j,\varphi} = w^{\phi_j - 1}_{j,\varphi}$.*

*Proof:* We first show that the sequence of $w^i_{j,\varphi}$ is non-decreasing. Let us consider slot $\psi$. If core $j$ at $\psi$ does not access bank $k$, the busy period remains unchanged, i.e., $w^{i+1}_{j,\varphi} = w^i_{j,\varphi}$. Otherwise, $w^{i+1}_{j,\varphi} \geq w^i_{j,\varphi} + L_M$ holds due to the following:

**i)** If the new access from $\psi$ is initiated before the busy period $w^i_{j,\varphi}$ ends, it grows by $L_M$. Thus,

$$w^{i+1}_{j,\varphi} = w^i_{j,\varphi} + L_M.$$

**ii)** If the new access from $\psi$ is initiated at or after the end of busy period $w^i_{j,\varphi}$, i.e., $\psi + L_B \geq w^i_{j,\varphi}$, then,

$$w^{i+1}_{j,\varphi} = \psi + L_B + L_M \geq w^i_{j,\varphi} + L_M.$$

Therefore, $w^{i+1}_{j,\varphi} \geq w^i_{j,\varphi}$ always holds.

Now we will show that $w^i_{j,\varphi} \leq w_{j,\varphi}$ holds for all $i$. To find the exact value of $w_{j,\varphi}$, one may begin from the very first slot of scheduling table. This is an inefficient way, however, in that the same procedure is unnecessarily repeated. Instead, we can take $d^s_{j,\varphi - T_j}$ into account in deriving $w_{j,\varphi}$, as illustrated in

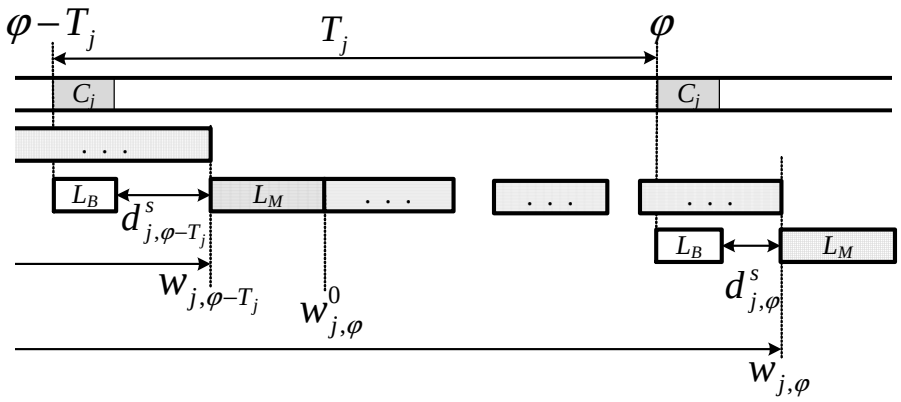



Fig. 13: Calculation of $w^i_{j,\varphi}$ and $d^s_{j,\varphi}$.

Figure 13, due to the following equality:

$$w^0_{j,\varphi} = w_{j,\varphi-T_j} + L_M = \varphi - T_j + L_B + d^s_{j,\varphi-T_j} + L_M.$$

That is, it is sufficient to consider only the slots in $[\varphi - T_j, \varphi - 1]$. Accordingly, the iterative procedure loops $T_j$ times, computing $(w^0_{j,\varphi}, w^1_{j,\varphi}, \ldots, w^{T_j-1}_{j,\varphi})$. Since the sequence of $w^i_{j,\varphi}$ is non-decreasing,

$$w^0_{j,\varphi} \le w^1_{j,\varphi} \le \cdots \le w^{T_j-2}_{j,\varphi} \le w^{T_j-1}_{j,\varphi} = w_{j,\varphi}.$$

Thus, $w^i_{j,\varphi} \le w_{j,\varphi}$ holds for all $i = 0, 1, \ldots, T_j - 1$. Similarly, if $\varphi = \phi_j$,

$$w^0_{j,\varphi} \le w^1_{j,\varphi} \le \cdots \le w^{\phi_j-2}_{j,\varphi} \le w^{\phi_j-1}_{j,\varphi} = w_{j,\varphi}.$$

Therefore, $w_{j,\varphi}$ calculated by the above procedure is the upper-bound of the busy period that can delay core $j$ at slot $\varphi$. ■

Algorithm 1 summarizes the computation process of $w_{j,\varphi}$.

Finally, slot delay $d^s_{j,\varphi}$ is non-zero if the obtained busy period, $w_{j,\varphi}$, delays the bank access of $\varphi^{th}$ slot and zero otherwise, as illustrated in Figure 13. Thus, for a given slot $\varphi$ of core $j$ that shares bank $k$ with others, $d^s_{j,\varphi}$ can be computed as follows:

$$d^s_{j,\varphi} = \max(w_{j,\varphi} - (\varphi + L_B), 0). \tag{11}$$

Note that $d^s_{j,\varphi}$ is lower-bounded by 0.

**Theorem 1.** *$d^M_i$ computed by Equation (7)–(11) is the the worst-case bank conflict delay of task $\tau_i$.*

*Proof:* The theorem is an immediate application of Lemma 1 and Lemma 2. In summary, the slot delays of core $j$ for each bank $k$ in $\mathbf{B_j}$ are first computed by Equation (9)–(11). Then, by Equation (8), we can find the worst-case bank conflict delay $D^j_k$ that any task in core $j$ could suffer due to using bank $k$. Finally, $d^M_i$ can be found by Equation (7), of which value depends on whether task $\tau_i$ uses any shared bank of core $j$, i.e., $B^j_s$ and/or $B^j_e$. ■

**Algorithm 1** CALC $w_{j,\varphi}$ $(j, \varphi, k, HRR)$

$HRR[\psi]$ : index of core using slot $\psi$ in the given HRR schedule.
$B_k[C]$ : *true* if core $C$ uses bank $k$ and *false* otherwise.
$\phi_j$ : the index of the first slot of core $j$ in the given HRR schedule.

**if** $\varphi = \phi_j$ **then**
    $w^0_{j,\varphi} := 0$
    $\psi := 1$
**else**
    $w^0_{j,\varphi} := \varphi - T_j + L_B + d^s_{j,\varphi - T_j} + L_M$
    $\psi := \varphi - T_j$
**end if**

$i := 0$
**while** $\psi < \varphi$ **do**
    **if** $B_k[HRR[\psi]]$ =*true* **then**
        **if** $\psi + L_B < w^i_{j,\varphi}$ **then**
            $w^{i+1}_{j,\varphi} \leftarrow w^i_{j,\varphi} + L_M$
        **else**
            $w^{i+1}_{j,\varphi} \leftarrow \psi + L_B + L_M$
        **end if**
    **else**
        $w^{i+1}_{j,\varphi} \leftarrow w^i_{j,\varphi}$
    **end if**
    $i \leftarrow i + 1$
    $\psi \leftarrow \psi + 1$
**end while**
$w_{j,\varphi} \leftarrow w^{i-1}_{j,\varphi}$

## IV. MILP FORMULATION

In this section, we present a *mixed integer linear programming* (MILP) formulation for our tunable WCET optimization problem described in Section II. Our optimization model takes as input a set of real-time tasks, $\mathbf{\Gamma}$, and a set of system parameters such as the number of cores, $N^{\mathbb{C}}$, and the shared cache model, $N^{\mathbb{B}}$ and $N^{\mathbb{X}}$. For the given input, it finds the optimal task assignments, harmonic round-robin schedule, and mappings of core-to-banks and task-to-columns that minimize the overall system utilization. In the rest of this section, we will describe how to formulate our proposed harmonic round-robin arbitration and two-level cache partitioning schemes (Section II), and our tunable WCET model (Section III), as a set of MILP constraints.

### A. Parameters and Variables

*1) System parameters:* Table I shows the list of system parameters given as input.

*2) Decision variables:* The followings are *zero-one* variables indicating to which task or core a shared resource is mapped. These and other decision variables not shown here will be explained in detail in the rest of this section when needed.

TABLE I: List of system parameters.

| Parameter | Description |
|---|---|
| $N^{\mathbf{\Gamma}}$ | number of tasks |
| $N^{\mathbb{C}}$ | number of cores |
| $N^{\mathbb{B}}$ | number of banks of the cache |
| $N^{W}$ | number of columns in a bank |
| $N^{\mathbb{X}}$ | number of columns of the cache |
| $N_i^{\mathbb{X}}$ | number of columns required for task $i$ |
| $N_i^{M}$ | number of cache accesses of task $i$ |
| $T^{max}$ | upper-bound of HRR periods |
| $T^{min}$ | lower-bound of HRR periods |
| $L_B$ | bus access latency |
| $L_M$ | bank access latency |

*a) Task to Core Mapping:*

$$\alpha_{i,j} = \begin{cases} 1 & \text{if task } i \text{ is allocated to core } j, \\ 0 & \text{otherwise.} \end{cases}$$

*b) Core to Cache Bank Mapping:*

$$\beta_{j,k} = \begin{cases} 1 & \text{if core } j \text{ uses bank } k, \\ 0 & \text{otherwise.} \end{cases}$$

*c) Core to HRR Slot Mapping:*

$$\sigma_{j,s} = \begin{cases} 1 & \text{if core } j \text{ uses slot } s \text{ of a harmonic round-robin table,} \\ 0 & \text{otherwise.} \end{cases}$$

*d) HRR Period Selection:*

$$\lambda_{j,p} = \begin{cases} 1 & \text{if } T_j \text{ has the value of } p + T^{min} - 1, \\ 0 & \text{otherwise.} \end{cases}$$

*e) Ratio of $T_{j+1}$ to $T_j$ :*

$$\lambda'_{j,q} = \begin{cases} 1 & \text{if } T_{j+1}/T_j \text{ has the value of } q, \\ 0 & \text{otherwise.} \end{cases}$$

*3) Range variables:* The following integer variables are used for representing ranges of banks and cache columns assigned to each core and each task, respectively (see Figure 17).

$$\text{Bank} \begin{cases} b_s^j & : \text{the index of the leftmost bank allocated to core } j, \\ b_e^j & : \text{the index of the rightmost bank allocated to core } j. \end{cases}$$

$$\text{Column} \begin{cases} x_s^i & : \text{the index of the leftmost column allocated to task } i, \\ x_e^i & : \text{the index of the rightmost column allocated to task } i. \end{cases}$$

### B. Objective Function

As presented in Section II-E, the optimization objective we consider in this study is to minimize the overall system utilization, that is,

$$\text{Minimize} \sum_{i=1}^{N^{\Gamma}} \frac{wcet_i}{p_i}.$$

Note that our optimization model is not restricted to a specific objective. We can, for example, minimize the WCET of a specific task or the number of required banks with the function of $wcet_i$ or $\sum_{j=1}^{N^{\mathbb{C}}} b_e^j - b_s^j + 1$, respectively.

### C. Constraints

*1) Harmonic round-robin:* Recall that a bus scheduling is a *harmonic round-robin* if and only if the HRR periods of the table satisfy Condition (1), which can be redefined as the following three conditions:

- $T^{min} \leq T_1 \leq T_2 \leq \cdots \leq T_{N^{\mathbb{C}}} \leq T^{max}$,
- $\forall_{j \leq N^{\mathbb{C}}-1} \ \frac{T_{j+1}}{T_j} = m_j \in \mathbb{N}$, and
- $\sum_{j=1}^{N^{\mathbb{C}}} \frac{1}{T_j} = 1$,

where $T_j$ is a positive integer variable representing the HRR period of core $j$. Note that the above conditions are non-linear. However these can be converted into *separable functions* with the help of *piecewise linear approximation* [22], and this does not compromise the accuracy of our optimization model since the constraints consist of only bounded integer values.

By the first condition above, each $T_j$ has to be an integer in the range $[T^{min}, T^{max}]$. This is a selection of a discrete point in the range, thus $T_j$ can be expressed by the following piecewise linear function:

$$T_j = T^{min} \cdot \lambda_{j,1} + (T^{min} + 1) \cdot \lambda_{j,2} + \cdots + T^{max} \cdot \lambda_{j,(T^{max} - T^{min} + 1)}, \tag{12}$$

where $\lambda_{j,p}$ is an indicator variable that is 1 if $T_j$ has the value $p + T^{min} - 1$ and 0 otherwise. Equation (12) therefore can be formulated as the following constraint:

**C1.** *For each core $j$,*

$$T_j = \sum_{p=T^{min}}^{T^{max}} p \cdot \lambda_{j,(p - T^{min} + 1)}.$$

Here only one term of the above equation has to end up having a positive value. Thus the sum of all $\lambda_{j,p}$ should be equal to 1, that is

**C2.** *For each core $j$,*

$$\sum_{p=1}^{T^{max} - T^{min} + 1} \lambda_{j,p} = 1.$$

The second condition requires that each HRR period, $T_{j+1}$, is a positive integer multiple of $T_j$ in order for the periods to be a harmonic set. Note that, by the first condition, the value of $m_j$ cannot be greater than $\frac{T^{max}}{T^{min}}$. Thus, each integer multiple $m_j$ is chosen within the range of 1 to $\frac{T^{max}}{T^{min}}$, and this can be expressed similar to the above approximation method as follows:

$$m_j = 1 \cdot \lambda'_{j,1} + 2 \cdot \lambda'_{j,2} + \cdots + m^{max} \cdot \lambda'_{j,m^{max}}, \tag{13}$$

where $m^{max}$ is a constant whose value is $\frac{T^{max}}{T^{min}}$, and $\lambda'_{j,q}$ is an indicator variable that is 1 if $m_j$ has the value of $q$ and 0 otherwise, and the sum of all $\lambda'_j$ is equal to 1. This equation therefore can be formulated as the following two constraints:

**C3.** *For each core* $j$,

$$m_j = \sum_{q=1}^{m^{max}} q \cdot \lambda'_{j,q}.$$

**C4.** *For each core* $j$,

$$\sum_{q=1}^{m^{max}} \lambda'_{j,q} = 1.$$

It should be noted that the above constraints are not directly related to HRR periods; it just determines the ratio of every adjacent HRR periods. Thus, we now need an additional constraint that relates $m_j$ to $\frac{T_{j+1}}{T_j}$. First of all, from **C1** we know that if $\lambda_{j,p}$ is 1, the value of $T_j$ is $p + T^{min} - 1$. Also, by **C3**, $m_j = q$ if $\lambda'_{j,q}$ is 1. Therefore, if both $\lambda_{j,p}$ and $\lambda'_{j,q}$ are 1, the value of $T_{j+1}$ has to be equal to

$$T_{j+1} = T_j \cdot m_j = (p + T^{min} - 1) \cdot q,$$

which can be represented as

**C5.** *For each core* $j$, $1 \le p \le T^{max} - T^{min} + 1$, *and* $1 \le q \le m^{max}$,

$$\text{if } \lambda_{j,p} = 1 \text{ and } \lambda'_{j,q} = 1 \Longrightarrow \lambda_{j+1,(p+T^{min}-1)\cdot q - T^{min}+1} = 1.$$

Depending on both values of $T_j$ and $m_j$, $T_{j+1}$ may exceed the upper limit of HRR periods, i.e., $T^{max}$. Hence we need another constraint that

**C6.** *For each core* $j$, $1 \le p \le T^{max} - T^{min} + 1$, *and* $1 \le q \le m^{max}$,

$$\text{if } (p + T^{min} - 1) \cdot q \ge T^{max} + 1 \Longrightarrow \lambda'_{j,q} = 0.$$

For the formulation for conditional constraints, please refer to [22].

The third condition enforces the HRR periods to be a complete round-robin as explained in Section II-A. In the condition, the value of each term of the summation is not in a linear relation with each HRR period. However, in this case also we can transform it into a separable form by the aforementioned piecewise linear approximation as each $T_j$ has a discrete integer value. From Equation (12) we already know that each $T_j$ is bounded between $T_{min}$ and $T_{max}$, and the indicator variable $\lambda_{j,p}$ has the value of 1 if $T_j$ is $T^{min} + p - 1$. Now let $O_j$ be the reciprocal of $T_j$, i.e., $\frac{1}{T^{min}+p-1}$. Then, by substituting $O_j$ for $T_j$, and

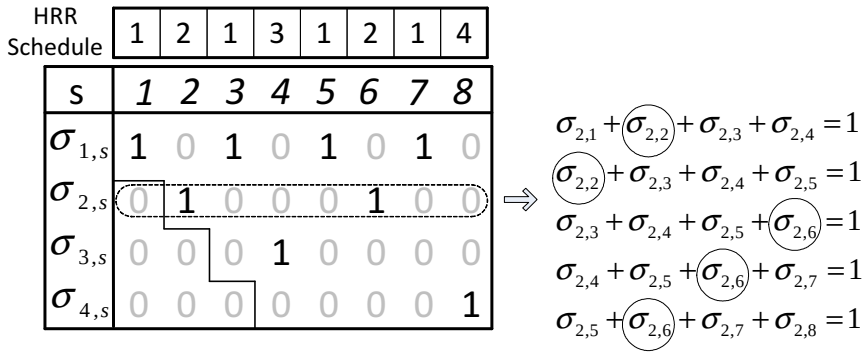



Fig. 14: The assignments of $\sigma_{j,s}$ for HRR of $(2, 4, 8, 8)$.

$\frac{1}{T^{min}+x}$ for $T^{min} + x$ from Equation (12), $O_j$ can be expressed as follows:

**C7.** *For each core* $j$,

$$O_j = (\frac{1}{T^{min}}) \cdot \lambda_{j,1} + (\frac{1}{T^{min}+1}) \cdot \lambda_{j,2} + \cdots + (\frac{1}{T^{max}}) \cdot \lambda_{j,(T^{max}-T^{min}+1)}.$$

We can therefore simply substitute $O_j$ for $\frac{1}{T_j}$ in the original condition, which results in

**C8.**

$$\sum_{j=1}^{N^{\mathbb{C}}} O_j = 1.$$

In summary, once $\lambda_{j,p}$ is chosen, $T_j$ and its corresponding $O_j$ are determined by **C1** and **C7**.

*2) Bus scheduling table:* As explained in Section II-A, once a set of HRR periods $\{T_1, T_2, \ldots, T_{N^{\mathbb{C}}}\}$ is given, the corresponding harmonic round-robin schedule can be uniquely determined by the following constraints:

- a slot can be assigned to only one core,
- the first slot of core $j$ can appear only after at least one slot is assigned to core $1, \ldots, j-1$, and
- slot $s$ is assigned to core $j$ if slot $s - T_j$ is assigned to core $j$.

Recall that $\sigma_{j,s}$ is a decision variable indicating whether core $j$ occupies $s^{th}$ slot of the scheduling table. Thus, building a harmonic round-robin scheduling table is equivalent to assigning a set of $\sigma_{j,s}$ to each core $j$. An example of $\sigma_{j,s}$ assignment is shown in Figure 14.

First of all, the first constraint above can be formulated straightforwardly as follows:

**C9.** *For each slot* $s$,

$$\sum_{j=1}^{N^{\mathbb{C}}} \sigma_{j,s} = 1.$$

The second constraint requires that the first slot of core $j$ cannot appear before $j^{th}$ slot of the table:

**C10.** *For each core* $j$ *and each slot* $s$,

$$\text{if } s \leq j-1 \Longrightarrow \sigma_{j,s} = 0.$$

By this constraint the sequence is uniquely determined for a given set of HRR periods. In other words, without this constraint, $< 1, 3, 1, 2, 1, 4, 1, 2 >$ is a possible schedule table for HRR of $(2, 4, 8, 8)$ instead of the one shown in Figure 14.

The periodicity of the slots of core $j$ can be ensured by checking the sum of every $T_j$ consecutive $\sigma_{j,s}$ of the core. In the example of Figure 14, the sum of four consecutive $\sigma_{2,s}$ should be equal to 1 as $T_2$ is 4. Accordingly, we need the following constraint:

**C11.** *For each core* $j$, $T^{min} \leq p \leq T^{max}$, *and* $1 \leq s \leq T^{max} - p + 1$,

$$T_j = p \Rightarrow \sum_{t=s}^{s+p-1} \sigma_{j,t} = 1.$$

*3) Task to core mapping:* Every task should be allocated to one of $N^{\mathbb{C}}$ cores, thus,

**C12.** *For each task* $i$,

$$\sum_{j=1}^{N^{\mathbb{C}}} \alpha_{i,j} = 1.$$

*4) Core to bank mapping:* The minimum number of cache banks required by core $j$, i.e., $n_j^B$, depends on the sum of the columns required by all tasks in the core, which is expressed as follows:

$$n_j^B = \sum_{i=1}^{N^{\mathbb{T}}} \alpha_{i,j} \cdot \frac{N_i^{\mathbb{X}}}{N^W}, \tag{14}$$

where $N_i^{\mathbb{X}}$ and $N^W$ are the number of cache columns that task $i$ requires and that a bank has, respectively. Note that $n_j^B$ may not be integer divisible by $N^W$, which means that for example at least two banks should be allocated to the core that requires 1.2 banks. However, this does not necessarily imply that two banks are sufficient for that core. It may require three banks. Let us consider the following cases which are illustrated in Figure 15:

(a) $n_j^B = \frac{1}{N^W}$ : The core requires only one column. In this case, one bank is sufficient.
(b) $n_j^B = \frac{2}{N^W}$ : The core requires two columns. These can belong to a bank, however it is possible that the columns may stretch across two banks.

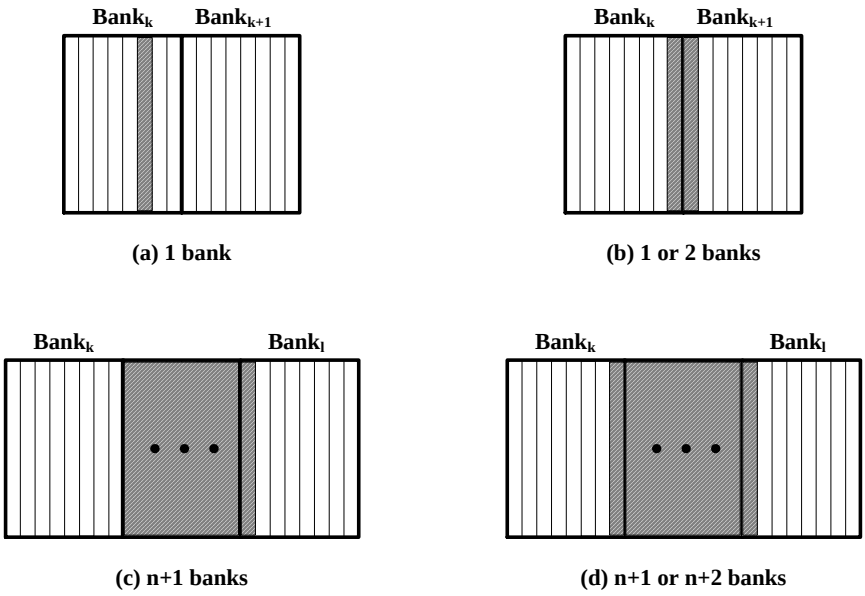



Fig. 15: The number of required banks for the different number of required columns.

(c) $n_j^B = n + \frac{1}{N^W}$ : The core requires $n * N^W + 1$ columns. In this case, $n + 1$ is the necessary and sufficient number of banks. It never happens that the columns stretch across $n + 2$ banks.
(d) $n_j^B = n + \frac{2}{N^W}$ : The core requires $n * N^W + 2$ columns. These can fit in $n + 1$ banks, however the columns may stretch across $n + 2$ banks.

Recall that the range of banks are expressed by $b_s^j$ and $b_e^j$; the indices of the first and last banks allocated to core $j$, and thus the number of banks that core $j$ will use is $b_e^j - b_s^j + 1$ banks. We can therefore formulate the following constraint:

**C13.** *For each core* $j$,

$$n_j^B \leq (b_e^j - b_s^j + 1) \leq n_j^B + 2 - \frac{2}{N^W},$$

where $n_j^B$ is defined by Equation (14). If $n_j^B$ is 0, however, no banks should be allocated to the core. Thus we need another constraint as follows:

**C14.** *For each core* $j$,

$$\text{if } n_j^B = 0 \Longrightarrow b_e^j = b_s^j = N^{\mathbb{B}} + 1.$$

For the purpose of bank conflict delay calculation, we should relate $b_s^j$ and $b_e^j$ with $\beta_{j,k}$ indicating whether core $j$ uses bank $k$ or not:

**C15.** *For each core* $j$ *and each bank* $k$,

$$\text{if } b_s^j \leq k \leq b_e^j \Longrightarrow \beta_{j,k} = 1.$$

In our proposed system model we restrict any two cores to share at most one bank. In addition to this, if two cores share a bank, it should be either the first or the last bank of each core (see Figure 16). This

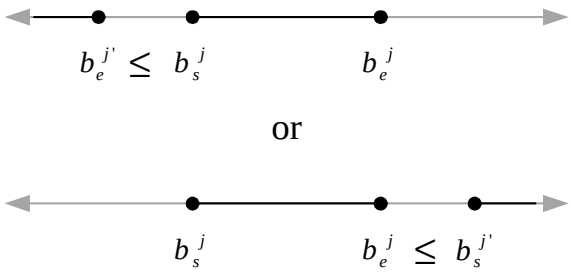



Fig. 16: Bank-sharing constraint.

constraint can be expressed as follows:

**C16.** *For each core* $1 \leq j \leq N^{\mathbb{C}-1}$*, and each* $j+1 \leq j' \leq N^{\mathbb{C}}$,

$$b_e^{j'} \leq b_s^j \quad \text{or} \quad b_e^j \leq b_s^{j'}.$$

For the formulation for logical constraints, please refer to [22].

Finally, as described in Section II-B, unbounded bank conflict delay may occur when a bank is shared among too many cores, or a few cores whose HRR periods are short. The condition that prevents such an unbounded delay is defined by Condition (2), and can be restated as follows:

$$\text{For each bank } k, \qquad \sum_{j=1}^{N^{\mathbb{C}}} \beta_{j,k} \cdot \frac{L_M}{T_j \cdot L_B} \leq 1.$$

Recall that we can substitute $O_j$ for $\frac{1}{T_j}$. Thus the above condition results in the following constraint:

**C17.** *For each bank* $k$,

$$\frac{L_M}{L_B} \sum_{j=1}^{N^{\mathbb{C}}} \beta_{j,k} \cdot O_j \leq 1.$$

The product of $\beta_{j,k}$ and $O_j$ can be linearized by adding the following four constraints and then by replacing $\beta_{j,k} \cdot O_j$ in **C17** with a new variable, $f_{j,k}$ :

**C17**$^*$**.** $\forall$ *core* $j$ $\forall$ *bank* $k$,

$$\begin{aligned} f_{j,k} &\leq U_{O_j} \cdot \beta_{j,k}, & f_{j,k} &\leq O_j, \\ f_{j,k} &\geq O_j - U_{O_j} \cdot (1 - \beta_{j,k}), & f_{j,k} &\geq 0, \end{aligned}$$

where $U_{O_j}$ is the upper-bound of $O_j$, which is $\frac{1}{T_{min}}$. If $\beta_{j,k}$ is 1, $f_{j,k}$ has to have the value of $O_j$ to satisfy all the constraints in **C17**$^*$. For the detail, please refer to [23].

*5) Task to column mapping:* Recall that the range of columns occupied by task $i$ is bounded by $x_s^i$ and $x_e^i$, and the required number of columns, $N_i^{\mathbb{X}}$, is given as an input. The mapping of task to columns can be simply expressed by the following constraint:

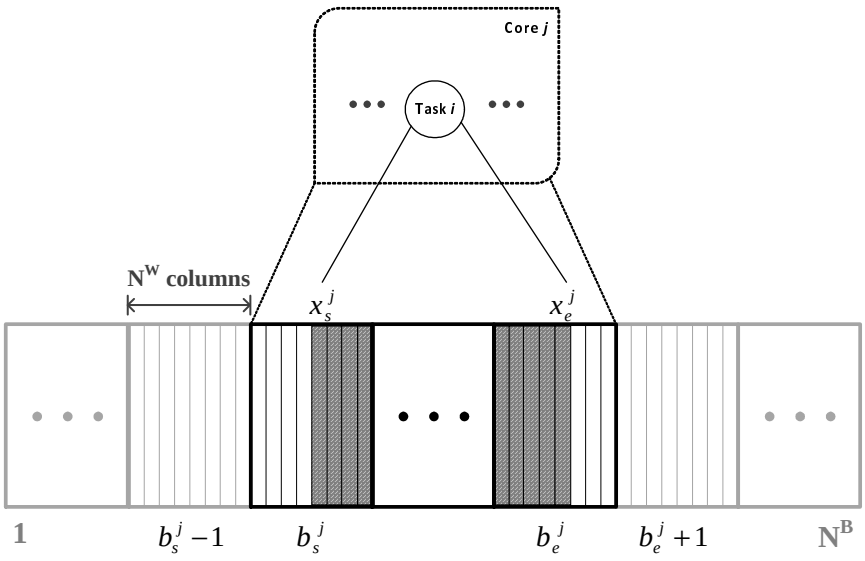



Fig. 17: The range constraint of mapping from a task to columns.

**C18.** *For each task* $i$,

$$x_e^i - x_s^i + 1 = N_i^{\mathbb{X}}.$$

Furthermore, no column is shared between tasks:

**C19.** *For each task* $1 \leq i \leq N^{\mathbf{\Gamma}-1}$, *and each* $i+1 \leq i' \leq N^{\mathbf{\Gamma}}$,

$$x_e^{i'} \leq x_s^i - 1 \quad \text{or} \quad x_e^i + 1 \leq x_s^{i'}.$$

Now an important constraint is that task $i$ residing in core $j$ can only occupy a contiguous subset of columns belonging to the banks allocated to the core. As can be seen in Figure 17, the indices of the first column of bank $b_s^j$ and the last column of bank $b_e^j$ are $(b_s^j - 1) \cdot N^W + 1$ and $b_e^j \cdot N^W$, respectively. The following constraint requires that $[x_s^i, x_e^i]$ should be in the range of $[(b_s^j - 1) \cdot N^W + 1, b_e^j \cdot N^W]$:

**C20.** *For each task* $i$ *and each core* $j$,

$$\text{if } \alpha_{i,j} = 1 \Longrightarrow (b_s^j - 1) \cdot N^W + 1 \leq x_s^i \quad \text{and} \quad x_e^i \leq b_e^j \cdot N^W,$$

where $N^W$ is the number of columns in a bank. This constraint can be transformed into the following two inequalities:

$$(b_s^j - 1) \cdot N^W + 1 - x_s^i \leq N^{\mathbb{X}}(1 - \alpha_{i,j}),$$
$$b_e^j \cdot N^W - x_e^i \geq (N^W - N^{\mathbb{X}})(1 - \alpha_{i,j}).$$

*6) WCET calculation:* By Equation (5), the worst-case execution time of task $i$ is formulated as follows:

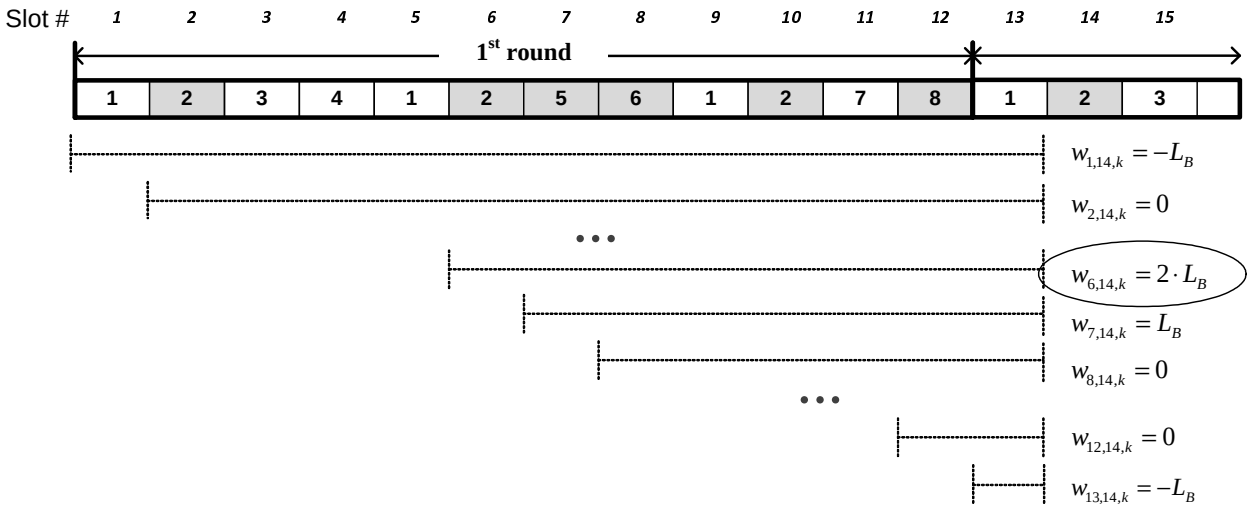



Fig. 18: An illustration of $w_{s',s,k}$ for slot $s = 14$ with $L_M = 2 \cdot L_B$.

**C21.** *For each task* $i$,

$$wcet_i = e_i + N_i^M \cdot \{(2 \cdot L_B + L_M) + (d_i^B + d_i^M)\},$$

where $d_i^B$ and $d_i^M$ are the upper-bounds of bus access delay and that of bank conflict delay, respectively.

*a) Bus Access Delay* $d_i^B$*:* As explained in Section III-A, the upper-bound of a bus access delay depends on to which core the task is allocated. Since the decision variable $\alpha_{i,j}$ is 1 if task $i$ is allocated to core $j$, the following constraint can be used to represent Equation (6):

**C22.** *For each task* $i$,

$$d_i^B = L_B(\sum_{j=1}^{N^{\mathrm{C}}} \alpha_{i,j} \cdot T_j).$$

Note that **C22** can be similarly linearized as in **C17**$^*$, but now $U_{T_j}$ is $T_{max}$.

*b) Bank Conflict Delay* $d_i^M$*:* The most crucial part in finding upper-bound bank conflict delay is the calculation of slot delays. If it is given that which core uses which banks, how the banks are shared with others, and how the cores are scheduled by a harmonic round-robin schedule, it is straightforward to compute slot delays with the information, as described in Section III-B. However, those are variable during optimization, thus the steps presented here are slightly different from the analysis in Section III-B.

Let us first define $w_{s',s,k}$ as the residual workload generated in the range of $s'$ to $s-1$ that could delay a bank access of slot $s$ to bank $k$, and which can be represented by the following expression:

$$w_{s',s,k} = L_M \cdot \left( \sum_{t=s'}^{s-1} \sum_{j=1}^{N^{\mathrm{C}}} \sigma_{j,t} \cdot \beta_{j,k} \right) - L_B(s - s'). \tag{15}$$

Recall that $\sigma_{j,t}$ is the decision variable introduced in Section IV-C2 representing whether core $j$ is scheduled at slot $t$ or not. Thus, $\sigma_{j,t} \cdot \beta_{j,k}$ is 1 if and only if core $j$ at slot $t$ uses bank $k$. The product

of two decision variables, $\sigma_{j,t} \cdot \beta_{j,k}$, can be linearized by adding the following three constraints and then by replacing $\sigma_{j,t} \cdot \beta_{j,k}$ with a new binary variable, $g_{j,k,t}$:

**C23*.** $\forall$ *core* $j$, $\forall$ *bank* $k$, $\forall_{1 \le t \le 2 \cdot T^{RR}-1}$,

$$g_{j,k,t} \le \sigma_{j,t}, \quad g_{j,k,t} \le \beta_{j,k}, \quad g_{j,k,t} \ge \sigma_{j,t} + \beta_{j,k} - 1.$$

$g_{j,k,t}$ is 1 only when both $\sigma_{j,t}$ and $\beta_{j,k}$ have the value of 1. The rationale behind this constraint, **C23**, is based on the fact that every slot accessing to bank $k$ generates $L_M$ workload and $L_B$ is consumed by each slot afterward. It is important to know here that not all $w_{s',s,k}$ are exact because either 1) a busy period may be broken at some slots during $[s', s-1]$ or 2) a busy period that counts from slot $s'$ may ignore some preceding workload generated at slots $s'' < s'$. However, the exact longest busy period is guaranteed to be taken into account by the above equation if it exists. To help to understand the underlying idea of $w_{s',s,k}$, let us consider Figure 18 as an example. Suppose here that core $2, 5, 6,$ and $8$ are sharing core $k$ and that we want to find $w_{s',14,k}$ for all $1 \le s' \le 13$. As mentioned before, some of the values are incorrect. For example, $w_{2,14,k}$ does not capture that the busy period starting from slot $2$ discontinues at slot $5$. Also, $w_{7,14,k}$ is calculated assuming there is no backlog at slot 7. The values of $w_{s',14,k}$ for both cases are therefore meaningless. On the other hand, $w_{6,14,k}$ is the exact residual workload that could maximally delay slot $14$ of core $2$, because it counts from a slot with no backlog and also there is no discontinuity in its busy period. In fact, such a busy period is uniquely determined if it exists.

As explained in Subsection III-B1, different slots of a core may experience different bank conflict delays, thus we need to calculate $w_{s',s,k}$ for each $s$. Because $\phi_j$ is variable with HRR schedules, $w_{s',s,k}$ should be computed from $s' = 1$. However, we do not need to compute the slot delays in the first round since the delay of each slot in the second round is always greater than or equal to the same slot in the first round, as mentioned before. Also, $w_{s',s,k}$ may vary depending on which bank the core at slot $s$ shares with others. Accordingly, each $w_{s',s,k}$ can be computed by the following constraint:

**C23.** *For each bank* $k$, $T^{RR} + 1 \le s \le 2 \cdot T^{RR}$, *and* $1 \le s' \le s-1$,

$$w_{s',s,k} = L_M \cdot \left( \sum_{t=s'}^{s-1} \sum_{j=1}^{N^{\mathbb{C}}} \sigma_{j,t} \cdot \beta_{j,k} \right) - L_B(s - s').$$

Now let us define by $u_{s,k}$ the worst-case delay that slot $s$ could experience when accessing bank $k$. Since it is the maximum of $w_{s',s,k}$ for all $1 \le s' \le s-1$, we can represent it by the following constraint:

**C24.** *For each bank* $k$, $T^{RR} + 1 \le s \le 2 \cdot T^{RR}$, *and* $1 \le s' \le s-1$,

$$u_{s,k} \ge w_{s',s,k}.$$

Note that $u_{s,k}$ should be lower-bounded by 0 because the core at slot $s$ may suffer no delay, especially if it does not use or share bank $k$. For example, in Figure 18, while $u_{14,k}$ is $2 \cdot L_B$, $u_{15,k}$ should be $0$ since there is no access from slot $15$ to bank $k$. Note also that if an access from slot $s$ to bank $k$ could experience any delay, $u_{s,k}$ always results in a positive value.

One can easily think of $u_{s,k}$ as a slot delay, i.e., $d^s_{j,\varphi}$, analyzed in Section III-B2 [7]. Now let us define by $z_{j,k}$ the upper-bound of slot delays of core $j$ using bank $k$, i.e., $D^j_k$ in Section III-B1. The following constraint is equivalent to Equation (8):

[7]The subscript $s$ of $u_{s,k}$ is different from the superscript $s$ of $d^s_{j,\varphi}$. While the former is the index of a slot, the latter is just a notational symbol representing that $d^s$ is a slot delay. The subscript $\varphi$ of $d^s_{j,\varphi}$ is the slot index.

**C25.** *For each core $j$ and each bank $k$, and $T^{RR}+1 \leq s \leq 2 \cdot T^{RR}$,*

$$z_{j,k} \geq \sigma_{j,s} \cdot u_{s,k}.$$

$z_{j,k}$ ends up being a positive value if and only if 1) core $j$ occupies slot $s$ of the scheduling table , 2) core $j$ uses bank $k$, and 3) core $j$ suffers bank conflict delays when accessing the bank. If at least one of the three conditions does not hold, then $z_{j,k}$ is always 0. Note that **C25** can be similarly linearized as in **C17**$^*$, but now $U_{u_{s,k}}$ is $(L_M - L_B) \cdot (s-1)$.

As explained in Section III-B, in order to find $d_i^M$, we need to compute $D_s^j$ and $D_e^j$ first. Recall that the variables $b_s^j$ and $b_e^j$ have the index of the leftmost and the rightmost bank of core $j$ respectively, thus the following constraints can be used to find $D_s^j$ and $D_e^j$ from slot delays $z_{j,k}$:

**C26.** *For each core $j$ and each bank $k$,*

$$\text{if } b_s^j = k \Longrightarrow D_s^j = z_{j,k}.$$

Similarly,
**C27.** *For each core $j$ and each bank $k$,*

$$\text{if } b_e^j = k \Longrightarrow D_e^j = z_{j,k}.$$

A task running on core $j$ could experience bank conflict delays if some of its columns are in shared banks as explained in Section. III. That is, if the columns are in the leftmost (the rightmost) bank of core $j$, the delay could be at most $D_s^j$ ($D_e^j$). However, if the columns stretch across both ends, the upper-bound delay is the maximum of $D_s^j$ and $D_e^j$. Recall now that the column range of a task, say $i$, is bounded by $x_s^i$ and $x_e^i$ variables (see Figure 17). Accordingly, the following constraints can finally find the bank conflict delay that an access from task $i$ on core $j$ can suffer in the worst-case:

**C28.** *For each task $i$, each core $j$, and each bank $k$,*

$$\text{if } \alpha_{i,j} = 1 \text{ and } x_s^i \leq b_s^j \cdot N^W \implies d_i^M \geq D_s^j.$$

**C29.** *For each task $i$, each core $j$, and each bank $k$,*

$$\text{if } \alpha_{i,j} = 1 \text{ and } (b_e^j - 1) \cdot N^W + 1 \leq x_e^i \implies d_i^M \geq D_e^j.$$

Note here that if $[x_s^i, x_e^i]$ stretch across $[b_s^j, b_e^j]$, $d_i^M$ results in $\max(D_s^j, D_e^j)$ by the above two constraints.

*7) Task and core utilization:* If the worst-case execution time of a task exceeds its period, it can never be schedulable. Thus we need to limit each task's worst-case execution time to be bounded by its period, which can be simply formulated as follows:

**C30.** *For each task $i$,*

$$wcet_i \leq p_i.$$

Likewise, we need to limit each core utilization to 1, or a specific bound, e.g., Liu and Layland's bound [24]:

TABLE II: Experimental parameters.

| Parameter | Value |
|---|---|
| $N^{\mathbb{C}}$ | $\{4, 6, 8\}$ cores |
| $N^{\mathbb{B}}$ | $\{4, 8\}$ banks |
| $N^{W}$ | $\{16, 32\}$ columns |
| $L_B$ | 2.5 ns |
| $L_M$ | $2 \cdot L_B$ |
| $N^{\mathbf{\Gamma}}$ | $\{20, 30, 40\}$ tasks |
| $e_i$ | uniform from $[10, 250]$ ms |
| $p_i$ | uniform from $[500, 10000]$ ms |
| $N_i^{\mathbb{X}}$ | uniform from $[1, 5]$ columns |
| $N_i^{M}$ | uniform from $[10^5, 10^6 \cdot \{1, 3, 5, 7, 10\}]$ times |

**C31.** *For each core* $j$,

$$\sum_{i=1}^{N^{\mathbf{\Gamma}}} \frac{wcet_i}{p_i} \cdot \alpha_{i,j} \leq 1.$$

Similar to **C17**$^*$, the above constraint also can be linearized with $U_{wcet_i} = p_i$ (by **C30**).

## V. Evaluation

In this section, we evaluate the proposed tunable WCET optimization model in terms of the minimum achievable system utilization. For this, we used IBM ILOG CPLEX 12.1 [25] to solve the optimization problem formulated in Section IV.

### A. Evaluation Method

*1) Experimental parameters:* Table II summarizes the experimental parameters used for the experiments. We consider a multicore system following the model described in Section II-C. Each system consists of 4, 6, or 8 cores and has a shared cache partitioned into 4 or 8 banks. Each bank is divided into 16 or 32 columns, however the total number of columns is maintained at 128. The bus access latency is assumed to be 2.5 nanosecond, and the latency of bank access is twice of bus.

With these system parameters, we generate synthetic task sets, each of which is randomly generated as follows: Each set consists of 20, 30, or 40 tasks. The fixed execution time and the period of a task are randomly selected from the ranges of [10 ms, 250 ms] and [500 ms, 10000 ms], respectively. Also, each task can occupy a set of at most 5 cache columns, and accesses the shared cache at least $10^5$ times but at most $10^7$ times.

*2) Experimental groups:* With these experimental parameters, we compare the following three methods, which are also summarized in Table III:

- *PureRR* : In this method, the bus is scheduled by pure round-robin arbitration method. That is, every core has the same slot period, i.e., $N^{\mathbb{C}}$, as in [20]. However, task allocations are flexible and the cache is partitioned by our proposed two-level partitioning method.

TABLE III: Experimental groups.

| | *PureRR* | *BFD* | *Proposed* |
|---|---|---|---|
| Task allocation | flexible | pre-allocated | flexible |
| Bus schedule | pure round-robin | harmonic round-robin | harmonic round-robin |
| Cache partition | two-level and flexible | | |

- *BFD* : In order to see how well our proposed method works for the cases in which each task is allocated to a fixed core, we employ *Best-Fit Decreasing* heuristic for task allocation. In this method, we first sort the task in decreasing order by *estimated task utilization*, which is defined as

$$\frac{\widehat{wcet_i}}{p_i} = \frac{e_i + N_i^M \cdot \{2 \cdot L_B + L_M + (N^{\mathbb{C}} \cdot L_B)\}}{p_i}.$$

  Each task is then allocated in the sorted order to the core which after accommodating the task will have the least remaining utilization. One may notice that $\widehat{wcet_i}$ above is similar to Equation (3). In fact, we compute the estimated WCET of task $i$ assuming that it does not experience any bank conflict delay and that the bus is scheduled by pure round-robin. It should be noted that, however, the bus schedule may change to a harmonic round-robin during an optimization. Also, the cache is partitioned by our two-level partitioning method.

- *Proposed* : This is the proposed method in this study. That is, all of task allocation, bus schedule, and cache partitioning are flexible and thus are optimized by our proposed methods.

It should be noted that in all the methods, the shared cache is not pre-partitioned because the unbounded bank conflict delay problem may arise with a random or fixed pre-partitioning.

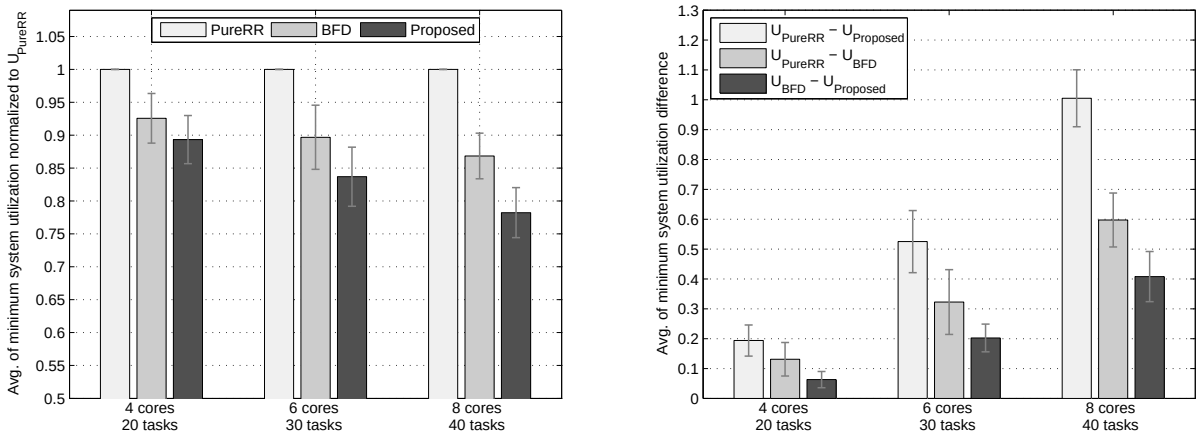
The optimization for the above methods is solved with the MILP formulation in Section IV. The only difference is that all $T_j$ have the same value $N^{\mathbb{C}}$ in ***PureRR***, and all $\alpha_{i,j}$ values are pre-assigned in ***BFD***. Thus one can easily expect that the minimum utilization obtained with our proposed method is always less than, or at least equal to, those that can be achieved with ***PureRR*** and ***BFD*** since these are more constrained optimization problem.

*3) Evaluation metric:* We compare the aforementioned three methods in terms of minimum achievable system utilization, , $U_{PureRR}$, $U_{BFD}$, and $U_{Proposed}$. Note that different task sets may have different baseline system utilizations. Thus, for fair comparison, we normalize $U_{BFD}$ and $U_{Proposed}$ to $U_{PureRR}$ for each task set and then take the average of 20 random sets for each configuration. Also, we compare and present $U_{PureRR} - U_{Proposed}$, $U_{PureRR} - U_{BFD}$, and $U_{BFD} - U_{Proposed}$ in order to illustrate the magnitude of the differences between the methods. Each error bar in each graph indicates the standard deviation of the normals.

### *B. Evaluation Result*

In this section, we will present the evaluation results for the aforementioned methods obtained with different 1) numbers of cores 2) intensities of cache accesses, and 3) cache configuration.

*1) Impact of core count:* Figure 19 compares the minimum system utilization as increasing the number of cores. In this experiment, the number of banks and that of columns in a bank are 8 and 16 respectively, and the number of cache accesses of tasks are randomly chosen from the range of $[10^5, 10^7]$. Also, in order to maintain average load for the cores, we increase the number of tasks as the core count increases.




(a) Minimum utilizations normalized to $U_{PureRR}$.

(b) Differences of minimum utilizations.

Fig. 19: Minimum system utilization with different core counts.

TABLE IV: Average minimum system utilizations with different core counts.

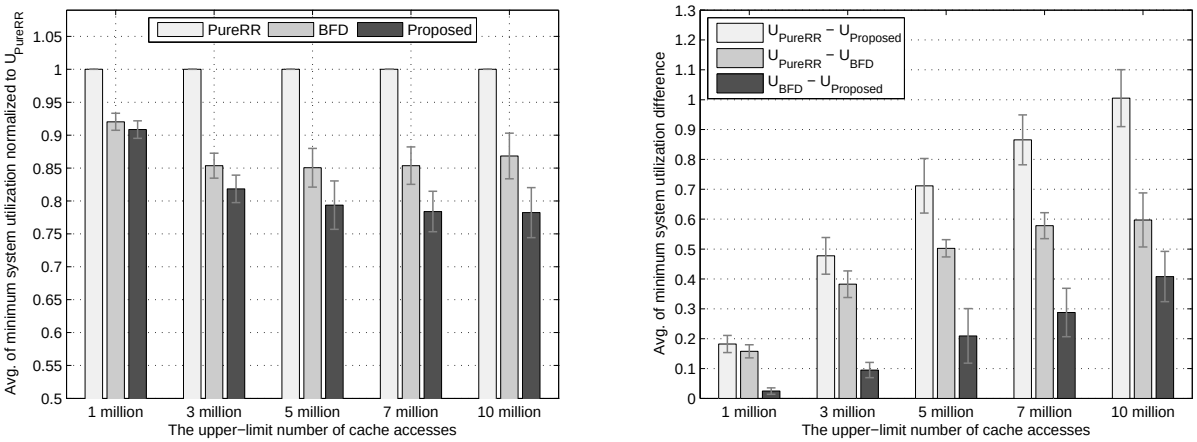
| | 4 cores 20 tasks | 6 cores 30 tasks | 8 cores 40 tasks |
|---|---|---|---|
| *PureRR* | 1.986237 | 3.409642 | 4.704161 |
| *BFD* | 1.855228 | 3.087044 | 4.106718 |
| *Proposed* | 1.792319 | 2.884526 | 3.698793 |

As the result in Figure 19(a) shows, our proposed methods can achieve lower system utilization than *PureRR* by $10\%$ at least to $20\%$ at most in average. We can see the improvement of *proposed* method compared to *PureRR* increases with the number of cores and thus of tasks. This can be backed up by the fact that the number of bank shared among cores increases with the core and task counts due to the fixed capacity of the shared cache, i.e., the number of banks and columns. Another important factor that can be attributed to this trend is that with more cores a harmonic round-robin schedule is able to be more flexible in prioritizing the cores so that high-utilization tasks can benefit from being allocated to the cores whose HRR periods are short. Meanwhile, the improvement gap between *Proposed* and *BFD* can be explained in a similar manner, that is, pre-assigning tasks to cores prevent further optimization by tightening the constraints on the bus schedule and bank-sharing.

One interesting observation from this experiment is that *BFD* outperforms *PureRR*. This implies that even if there is no flexibility in assigning tasks to cores, it is likely that the system utilization can significantly be lowered by employing our proposed harmonic round-robin bus scheduling, which in turn enhances the efficiency of two-level cache partitioning as described in Section II-B.

*2) Impact of cache accesses intensity:* In order to see how the different cache access intensities affect the proposed optimization model, we perform another experiment by increasing the upper-limit number of cache accesses. In this experiment, the numbers of cores, tasks, and banks are fixed to $8$, $40$, and $8$ respectively, and each bank consists of $16$ columns. We vary the upper-limit number of cache accesses from $1$ million to $10$ million while the lower-limit is fixed to $0.1$ million. With these parameters, $N_i^M$ for each task is chosen randomly between the limits.

As can be seen from Figure 20, the utilization improvement of *Proposed* over *PureRR* increases with the upper-limit of cache accesses. This can be explained by the underlying rationale behind our Tunable




(a) Minimum utilizations normalized to $U_{PureRR}$.

(b) Differences of minimum utilizations.

Fig. 20: Minimum system utilization with different cache access intensities.

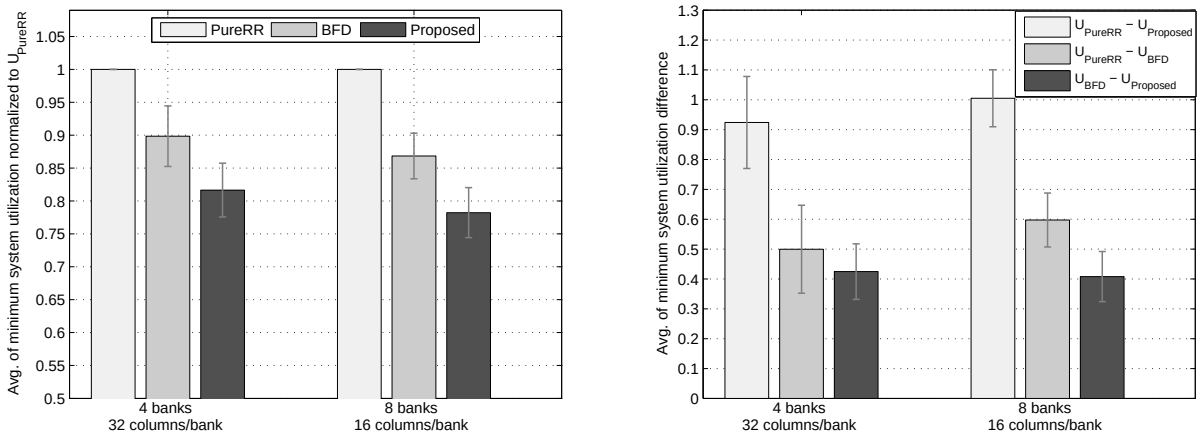
TABLE V: Average minimum system utilizations with different cache intensities.

| | 1 million | 3 million | 5 million | 7 million | 10 million |
|---|---|---|---|---|---|
| *PureRR* | 2.013391 | 2.654945 | 3.556437 | 4.073253 | 4.704161 |
| *BFD* | 1.855803 | 2.272641 | 3.053946 | 3.495127 | 4.106718 |
| *Proposed* | 1.831165 | 2.177714 | 2.844715 | 3.207637 | 3.698793 |

WCET model (Section II-D). That is, the possibility of further optimization grows with the ratio of tunable delay to the fixed execution time. Thus if a task accesses the bus and banks more intensively than others, it is more likely for the task to enable the overall system utilization to be further reduced by a similar argument explained in the previous discussion. However, this does not necessarily imply that higher intensity of cache accesses would always lead to less system utilization, as can be observed from the above result. We can see from Figure 20(a) that the improvements of *Proposed* and *BFD* over *PureRR* converge to certain levels (around $20\%$ and $15\%$, respectively) as the tasks become more memory-intensive. This is due to the fact that as the proportion of tunable delay grows, the *sensitivity* of WCET variation to changes in bus schedule and cache partitioning also increases. Recall that, by our tunable WCET model, a decrease in one's delay naturally leads to increases in the delays of the rest of the tasks. Therefore, if most tasks are sensitive to WCET variation, the overall improvement may not be possible because it is more likely for some tasks or cores to exceed the utilization bound constraints.

*3) Impact of cache configuration:* Figure 21 shows how different cache configurations affect our optimization model. In this experiment, the numbers of cores and tasks, and the upper-limit cache accesses counts are fixed to $8$, $40$, and $10$ million, respectively. With these parameters, we consider two different cache configurations: 4 banks $\times$ 32 columns and 8 banks $\times$ 16 columns.

It can be seen from the figure that the utilization improvements of *Proposed* and *BFD* over *PureRR* with $8 \times 16$ structure are slightly higher than those with $4 \times 32$ one. Note that the total number of columns required by all tasks are similar between two cases. Although it is small but the reason why such a difference nevertheless arises is mainly due to the different granularity of core-to-bank mappings. To put it simply, let us suppose that a core requires 35 columns. With the $8 \times 16$ cache, it is possible that the core is mapped to 3 out of 8 banks. With the $4 \times 32$ cache, on the other hand, the core needs at




(a) Minimum utilizations normalized to $U_{PureRR}$.

(b) Differences of minimum utilizations.

Fig. 21: Minimum system utilization with different cache configurations.

TABLE VI: Average minimum system utilizations with different cache configurations.

| | 4 banks× 32 columns | 8 banks× 16 columns |
|---|---|---|
| *PureRR* | 5.19825 | 4.704161 |
| *BFD* | 4.69875 | 4.106718 |
| *Proposed* | 4.27405 | 3.698793 |

least 2 but possibly 3 out of 4 banks, which is equivalent to 4 or 6 banks of the $8 \times 16$ cache. Because each core is likely to take more banks than it actually needs, the number of bank-sharing can increase with the $4 \times 32$ cache. As an extreme case, if each bank consists of only one column, then no core, and thus no task, suffers any bank conflict delay due to the same reason. Another factor that influences the difference is that, as already described in the previous discussions, with more cores bank conflict delays can further be reduced by the help of harmonic round-robin scheduling.

## VI. Related Work

A great deal of research efforts have been devoted to address the optimization of shared resource allocation and arbitration in multicore architectures. For on-chip memory partitioning, Suhendra *et al.* [26] proposed an ILP formulation that finds the optimal scratchpad memory partition and task allocation/scheduling which minimize tasks' execution times. In [27], the authors examined the impacts of different combinations of cache locking and partitioning schemes on the system utilization. In [28], Bui *et al.* proposed a genetic algorithm that can find a near optimal cache partition and task-to-partition assignments that minimizes the system utilization.

Another line of research has focused on shared bus arbitration methods. Rosén *et al.* [9] and Andrei *et al.* [29] addressed TDMA-based bus access policies that is tightly coupled with the worst-case execution paths of tasks. They proposed an optimization problem that finds the optimal TDMA schedule which minimizes the global delay of tasks, and extended it to deal with average-case delays [11]. Additionally, Schranzhofer *et al.* [15] analyzed the worst-case response time of real-time tasks under different cache access models for TDMA-based bus arbitration policies.

Although it is not addressed in this paper, the issue of shared memory contention is also receiving increasing attention [30], [31].

## VII. Conclusion and Future Work

In this paper, we have proposed a novel perspective of WCET model called *tunable WCET*, which enables system-level optimization for hard real-time multicore system. In this model, the WCETs of tasks are no longer dependent upon a system configuration, but rather decide how to configure the shared bus and cache of the system. As the WCET-aware shared resource arbitration and allocation methods, we have introduced harmonic round-robin bus scheduling and two-level cache partitioning method. We have formulated an MILP-based optimization problem, and the experimental results have shown that our proposed methods can significantly lower overall system utilizations.

In the future, we will investigate how to extend our resource allocation methods to support soft real-time tasks as well. One possible direction is to allow soft real-time tasks to share a few banks of the shared cache, and then take the storage interference due to column-sharing [32] into account in the tunable WCET model. Additionally, we plan to develop a heuristic algorithm that can efficiently solve our tunable WCET optimization problem. Also, we have assumed in this paper that the critical path does not change with the change in tunable delays. This is a clear limitation, thus, as in [9], we will investigate the possibility of combining control flow analysis with our WCET analysis, in order to evaluate the practical applicability of the proposed approach.

## References


[1] "Intel Xeon E7 Processors," http://www.intel.com/products/server/processor/xeonE7/index.htm.

[2] "Oracle Unveils SPARC T3 Processor and SPARC T3 Systems," http://www.oracle.com/us/corporate/press/173536.

[3] J. Held, J. Bautista, and S. Koehl, "From a Few Cores to Many: A Tera-scale Computing Research Overview," http://download.intel.com/research/platform/terascale/terascale_overview_paper.pdf.

[4] "Freescale QorIQ P4080 Processor," http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=P4080.

[5] "ARM11MPCore Processor," http://www.arm.com/products/processors/classic/arm11/arm11-mpcore.php.

[6] A. Fedorova, S. Blagodurov, and S. Zhuravlev, "Managing Contention for Shared Resources on Multicore Processors," *Communications of the ACM*, vol. 53, no. 2, pp. 49–57, 2010.

[7] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors Via Scheduling," in *Proc. of Int'l Conference on Architectural Support for Programming Languages and Operating Systems*, 2010.

[8] M. Kandemir, S. P. Muralidhara, S. H. K. Narayanan, Y. Zhang, and O. Ozturk, "Optimizing Shared Cache Behavior of Chip Multiprocessors," in *Proc. of IEEE/ACM Int'l Symposium on Microarchitecture*, 2009, pp. 505–516.

[9] J. Rosén, A. Andrei, P. Eles, and Z. Peng, "Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip," in *Proc. of IEEE Int'l Real-Time Systems Symposium*, 2007, pp. 49–60.

[10] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling Shared Cache and Bus in Multi-cores for Timing Analysis," in *Proc. of Int'l Workshop on Software and Compilers for Embedded Systems*, 2010, pp. 1–10.

[11] J. Rosén, C.-F. Neikter, P. Eles, Z. Peng, P. Burgio, and L. Benini, "Bus Access Design for Combined Worst and Average Case Execution Time Optimization of Predictable Real-Time Applications on Multiprocessor Systems-on-Chip," in *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011.

[12] J. Yan and W. Zhang, "WCET Analysis for Multi-Core Processors with Shared L2 Instruction Caches," in *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2008, pp. 80–89.

[13] W. Zhang and J. Yan, "Accurately Estimating Worst-Case Execution Time for Multi-core Processors with Shared Direct-Mapped Instruction Caches," in *Proc. of IEEE Int'l Conference on Embedded and Real-Time Computing Systems and Applications*, 2009, pp. 455–463.

[14] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury, "Timing Analysis of Concurrent Programs Running on Shared Cache Multi-Cores," in *Proc. of IEEE Int'l Real-Time Systems Symposium*, 2009, pp. 57–67.

[15] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Timing Analysis for TDMA Arbitration in Resource Sharing Systems," in *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 215–224.

[16] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, "Worst Case Delay Analysis for Memory Interference in Multicore Systems," in *Proc. of the Conference on Design, Automation and Test in Europe*, 2010, pp. 741 – 746.

[17] M. Schoeberl, "JOP: A Java Optimized Processor," in *Proc. of Workshop on Java Technologies for Real-time and Embedded Systems*, 2003.

[18] A. El-Haj-Mahmoud, A. S. AL-Zawawi, A. Anantaraman, and E. Rotenberg, "Virtual Multiprocessor: An Analyzable, High-performance Architecture for Real-Time Computing," in *Proc. of Int'l Conference on Compilers, Architecture, and Synthesis from Embedded Systems*, 2005.

[19] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee, "Predictable Programming on a Precision Timed Architecture," in *Proc. of Int'l Conference on Compilers, Architecture, and Synthesis from Embedded Systems*, 2008.

[20] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware Support for WCET Analysis of Hard Real-Time Multicore Systems," in *Proc. of IEEE/ACM Int'l Symposium on Computer Architecture*, 2009, pp. 57–68.

[21] D. Chiou, P. Jain, S. Devadas, and L. Rudolph, "Dynamic Cache Partitioning via Columnization," in *Proc. of Design Automation Conference*, 2000.

[22] H. P. Williams, *Model Building in Mathematical Programming*, 4th ed. Wiley, 1999.

[23] J. Bisschop, *AIMMS Optimization Modeling*, Paragon Decision Technology, March 2011.

[24] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[25] "IBM ILOG CPLEX Optimizer," http://www-01.ibm.com/software/integration/optimization/cplex-optimizer.

[26] V. Suhendra, C. Raghavan, and T. Mitra, "Integrated Scratchpad Memory Optimization and Task Scheduling for MPSoC Architectures," in *Proc. of Int'l Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2006, pp. 401–410.

[27] V. Suhendra and T. Mitra, "Exploring Locking & Partitioning for Predictable Shared Caches on Multi-Cores," in *Proc. of Design Automation Conference*, 2008.

[28] B. D. Bui, M. Caccamo, L. Sha, and J. Martinez, "Impact of Cache Partitioning on Multi-tasking Real Time Embedded Systems," in *Proc. of IEEE Int'l Conference on Embedded and Real-Time Computing Systems and Applications*, 2008.

[29] A. Andrei, P. Eles, Z. Peng, and J. Rosén, "Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip," in *Proc. of Int'l Conference on VLSI Design*, 2008, pp. 103–110.

[30] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *Proc. of IEEE/ACM Int'l Symposium on Microarchitecture*, 2007.

[31] M. Paolieri, E. Quiñones, F. J. Cazorla, and M. Valero, "An Analyzable Memory Controller for Hard Real-Time CMPs," *IEEE Embedded Systems Letters*, vol. 1, no. 4, DEC 2009.

[32] H. Ramaprasad and F. Mueller, "Bounding Preemption Delay within Data Cache Reference Patterns for Real-Time Tasks," in *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.