Abstract—The unpredictable worst-case timing behavior of multicores architectures has been the biggest stumbling block for a widespread use of multicore in hard real-time systems. A great deal of research effort has been devoted to address the issue. Among others, the development of a new multicore architecture has emerged as an attractive solution because it can eliminate the unpredictable interference sources in the first place. This opens a new possibility of system-level optimizations with multicore-based hard real-time systems. To address this issue, we propose a new perspective of WCET model called tunable WCET, in which the WCETs of tasks are elastically adjusted according to the optimal shared resource allocation and arbitration methods. For this, we propose novel WCET-aware harmonic round-robin bus scheduling and two-level cache partitioning method. We present a mixed integer linear programming formulation as the solution to the optimization of tunable WCETs. Our experimental results show that the proposed methods can significantly lower overall system utilizations.

I. INTRODUCTION

Multicore processors are receiving wide attention from avionic and automotive industries as the demand for high-end real-time applications is rapidly growing [1], [2]. However, the major obstacle in applying multicore processors to such domains is that the execution time of applications can vary noticeably depending on how physical resources, such as shared cache [3]–[5] and shared bus [6]–[8], are contended between tasks on multiple cores. This unpredictable inter-core interferences in current multicore architecture are a huge barrier, especially for safety-critical systems in which the predictability of the worst-case temporal behavior is of primary importance. One possible solution to this kind of problem lies in analytic methods that can precisely estimate the worst-case execution time (WCETs) of applications in the presence of shared resource contentions [9]–[11]. The assumptions made in the existing analyses are commonly 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 inter-core interferences sources becomes much higher than before.

Due to such fundamental limitations of the analytic methods, hardware modification on multicore architectures has emerged as an attractive and viable solution [12]–[14]. While analytic methods try to analyze inter-core resource contentions, the new multicore architectures focus on eliminating such interferences in the first place for higher timing predictability. This line of research opens a new possibility of system-level optimizations with multicore-based hard real-time systems, as we will explore throughout this paper.

A. Motivating Hard Real-Time Multicore Architecture

In [14], Paolieri et al. proposed a new hard real-time multicore architecture in which accesses to shared bus and cache are controlled by hierarchical arbiters. The architecture employs round-robin as the shared bus arbitration policy, thus the maximum bus access delay is bounded by the number of cores in a system. They also analyzed shared cache interference with regard to two factors - bank conflict and storage conflict. The maximum bank conflict delay is similarly bounded as the bus access conflict is. They addressed cache partitioning techniques that can eliminate storage conflicts by splitting a cache space into private banks or columns.

This architecture provides a good architectural foundation for future hard real-time multicore systems. First of all, since resource contentions are resolved by hardware arbiters, it does not require any modification on applications’ source code and also does not impose any restrictions on programming language or OS. Furthermore, the WCET of a task can be obtained without the knowledge of other tasks.

B. Tunable WCET and Its Optimization

Paolieri’s multicore architecture provides a high degree of temporal predictability of the applications’ WCET; each core or task has its exclusive spatial and temporal partitions for shared resource accesses. While this makes the WCET analysis much easier by eliminating the potential sources of resource contentions, one possible limitation is that the resources may have limited capacities to accommodate a given workload. Recall that every task is assigned to private cache banks or columns in order to avoid storage conflicts. 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 bank conflict delays when accessing the banks. Therefore a proper partitioning method which can efficiently utilize the shared cache space while minimizing such 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 access times, we can achieve an enhanced overall efficiency, e.g., 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 execution branches and whose cache footprints rarely change from period to period.

In order to address the above problems, we propose a new perspective of WCET model called tunable WCET, in which the WCET of a task is partitioned into two components - fixed execution time and tunable delay, as illustrated in Fig. 1. In this model, the tunable delay of a task is a function of system configuration, and thus it enables system-level optimization for certain purposes by elastically adjusting shared resource configurations. In particular, we focus on the two major inter-core interference sources - shared bus and shared cache. In this paper, we investigate how different configurations of bus arbitration and cache partition would affect tasks’ tunable delay. In order to achieve this goal, we adopt Paoliieri’s multicore architecture [14] and propose novel bus arbitration and cache partitioning methods called harmonic round-robin and two-level cache partitioning, respectively. Our harmonic round-robin arbitration realizes application-aware bus scheduling by varying the worst-case bus access delays of different cores. Similarly, our two-level cache partitioning scheme maps banks to cores and columns to tasks in a way that bank access conflicts are minimized with the help of a harmonic round-robin bus schedule. As the solution to the optimization problem of tunable WCET’s, we present a mixed integer linear programming (MILP) formulation. As will be discussed later, our experimental results show that the proposed methods can significantly lower overall system utilization.

The rest of this paper is organized as follows: Sec. II introduces our HRR bus arbitration and two-level cache partitioning method and defines the tunable WCET optimization problem. Then, Sec. III describes in detail the tunable WCET model and its analysis. In Sec. IV, we present the MILP formulation for the tunable WCET optimization problem. The experimental results are given in Sec. V. Sec. VI summarizes the related work, and then Sec. VII concludes this paper.

II. Optimization of Tunable WCET

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

A. System Model

We consider a multicore system that consists of \( N^C \) homogeneous cores, \( C = \{ C_1, C_2, \ldots, C_{N^C} \} \). The system has a shared cache \( B \) which is partitioned into \( N^B \) banks \( \{ B_1, B_2, \ldots, B_{N^B} \} \), each of which is subdivided into \( N^W \) columns. That is, the cache has total \( N^X = N^B \cdot N^W \) columns, i.e., \( X = \{ X_1, X_2, \ldots, X_{N^X} \} \). On that system, we assume a set of \( N^F \) real-time tasks \( \Gamma = \{ \tau_1, \tau_2, \ldots, \tau_{N^F} \} \), each of which is represented by \( \tau_i = (e_i, p_i, N^X_i, N^M_i) \); \( \tau_i \) executes on \( C_j \) with the execution time of \( e_i \) and the period of \( p_i \), and it accesses \( N^X_i \) cache columns \( N^M_i \) times to store/load its instructions and data. The cache is partitioned by the two-level partitioning method, and the bus access is arbitrated by an HRR schedule, both of which are introduced in the following subsections. With these constraints, we further assume that both \( N^X_i \) and \( N^M_i \) are pre-profiled by a static analysis.

B. Harmonic Round-Robin Bus Arbitration

In pure round-robin bus scheduling, the bus access delays of every task are upper-bounded by \( N^C \cdot L_B \), where \( N^C \) is the number of cores and \( L_B \) is the bus access latency \(^1\). As mentioned before, this is inefficient in that the same amount of bus access delay of different tasks affects a certain performance metric, e.g., overall system utilization, differently. For example, suppose that \( \tau_A \) in \( C_1 \) and \( \tau_B \) in \( C_3 \) have the same worst-case bus access delay of \( 4 \cdot L_B \) as shown in Fig. 2(a). If their period is 50 but the total numbers of cache accesses, \( N^M_A \) and \( N^M_B \), are 500 and 100, respectively, then the contributions of their bus access delays to the system utilization are \( u^A_{bus} = (500 \cdot 4 \cdot L_B)/50 \) and \( u^B_{bus} = (100 \cdot 4 \cdot L_B)/50 \), respectively. Similarly, suppose now that \( p_A \) is 10 and \( N^M_A \) is 100 in both cases, \( u^A_{bus} / u^B_{bus} \) is 5, meaning that \( \tau_A \) affects the system utilization five times more than \( \tau_B \). Now let us suppose that \( C_1 \) and \( C_3 \) are guaranteed to be able to access the bus every 2 and 8 slots, respectively, as shown in Fig. 2(b). Then, \( u^A_{bus} \) and \( u^B_{bus} \) become \( (100 \cdot 2 \cdot L_B)/10 \) and \( (100 \cdot 8 \cdot L_B)/50 \), respectively. Accordingly, the net contribution, i.e., \( u^A_{bus} + u^B_{bus} \), is reduced from \( (2400 \cdot L_B)/50 \) to \( (1800 \cdot L_B)/50 \). 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 util-

\(^1\)We assume that a bus request should arrive at the bus before each designated time slot to be granted to send the request at the bus slot.
such tasks and assigning them to the more pre-assigned to cores, we can further reduce it by grouping we propose Harmonic Round-Robin arbitration, we can also harmonize with each other if and only if they satisfy the following conditions: 

\[ T_j + 1 \leq \sum_{j=1}^{N_C} \frac{1}{T_j} = 1. \]  

(1)

Because \( \{T_j\} \) are bounded within \([T^{\min}, T^{\max}]\), only a finite number of harmonic sets can be made within the given range. Once a set of HRR periods \( \{T_1, T_2, \ldots, T_{N_C}\} \) satisfying Condition (1) is obtained, we can create the unique corresponding HRR table of length \( T^{RR} = T_{N_C} \) by constraints C9–C11 in Sec. IV. For example, Fig. 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 [14] that the worst-case bus access delay of a task can be obtained without the knowledge of other real-time tasks. Furthermore, more importantly, it helps our two-level cache partitioning method reduce bank conflicts, as will be described in the following subsection.

C. Two-Level Cache Partitioning

In [14], the authors consider a column-level cache partitioning method called columnization [15] as a way to eliminate storage conflict delays. In columnization, however, a task may suffer a bank conflict delay if another task is already accessing the same bank. To avoid such interference, we may allocate a set of private banks to each task, which is called bankization [14]. However, this is restrictive in that the shared cache has to have at least as many banks as 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, as illustrated in Fig. 3(a) and (b). In the example, \( C_2 \) and \( C_4 \) can share bank \( B_k \) without suffering any bank conflict delays since the request from \( C_4 \) begins after \( C_2 \) completes its request. On the other hand, \( C_3 \) may suffer a bank conflict delay since its requests can be overlapped with the one from \( C_2 \). For the same reason, \( C_3 \) and \( C_4 \) cannot share any bank without suffering delays. However, as shown in (b), if we assign another bank \( B_{k'} \) to \( C_3 \), none of the cores experiences bank conflict delays.

However one may encounter a situation where no more banks are available for \( C_3 \) in Fig. 3(a). In that case, we can further reduce or eliminate the bank conflict delays with the help of a harmonic round-robin schedule, as illustrated in Fig 3(c). The HRR schedule in the example enables the bank access requests from \( C_2, C_3 \), and \( C_4 \) to be serial, that is to not overlap.

Another important factor that influences on the worst-case 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 Fig. 3(c) again. As mentioned before, the bank conflicts among $C_2$, $C_3$, and $C_4$ could be avoided since their slots are far enough apart from each other. However, once $L_M$ becomes $2.5 \cdot L_B$, then the cores start experiencing bank conflict delays, as shown in Fig. 3(d). Furthermore, the delays can be accumulated beyond one round, which we call unbounded bank conflict delay problem (see Fig. 4). In order to prevent it, the busy period of bank accesses should be bounded by the length of one round. Since each $C_j$ can generate at most $\frac{1}{T_{RR}}$ bank requests within $T_{RR}$, the constraint that prevents such unbounded bank conflict delays for a shared bank, $B_k$, can be expressed as follows:

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

Thus, in Fig. 4, one of the cores should use a separate bank.

As has been described above, arbitrary bank allocation may introduce unnecessary bank conflicts, and the problem can be aggravated by insufficient banks. One efficient way to partition a 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. 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 $C_A$ and $C_B$ share two banks, e.g., $B_k$ and $B_{k+1}$, each of which has 10 columns. Suppose that $C_A$ uses 7 columns of $B_k$ and 5 columns of $B_{k+1}$, and $C_B$ uses the rest. This is, however, equivalent to giving all columns of $B_k$ and 2 columns of $B_{k+1}$ to $C_A$ and then letting only $B_{k+1}$ be shared between $C_A$ and $C_B$. 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 Fig. 5. By this task-level subpartitioning, we can eliminate storage conflict interferences among tasks even in the same core. Moreover, we can prioritize the tasks in allocating their columns. Let us consider an example shown in Fig. 5, where $B_k$ is shared between $C_i$ and $C_j$. Since task $\tau_B$ shares $B_k$ with $\tau_A$, it may suffer bank conflict delays. However, $B_{k+1}$ of $C_j$ cannot be shared with any other cores by our core-level partitioning, and hence the bank accesses from $\tau_C$ are free from any bank conflicts. We call such banks BCD-free banks - a set of banks in which tasks mapped to a subset of their columns cannot experience any bank conflict delays. Accordingly, a more memory-intensive or high-utilization task can benefit from using such BCD-free banks.

D. Tunable WCET

Fig. 6 illustrates the rationale behind the tunable WCET model proposed in this paper; the WCET of a task is partitioned into fixed execution time and tunable delay. While the former is the maximum time duration that a task could take to execute the instructions over its critical path, the latter is the sum of the delays incurred for all of its cache accesses over the same path. In particular, we model the variable delays as the function of bus and cache configurations. Accordingly, the tunable part is subdivided into bus access delays and bank access delays. Now let $d^B_i$ and $d^M_i$ are the upper-bounds of bus access and bank conflict delays for each cache access, as shown in Fig. 7. Then, the WCET of $\tau_i$ can be defined as follows:

$$\text{wcet}_i = c_i + N^M_i \cdot \{L + (d^B_i + d^M_i)\}, \quad (3)$$

where $N^M_i$ is the number of $\tau_i$’s cache accesses, $L = 2 \cdot L_B + L_M$ is the sum of fixed latencies $^2$.

One may argue that the critical path of, and thus $N^M_i$ 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, $c_i$, is so long enough that other paths cannot be longer than the obtained critical path even if they would experience maximum possible delays.

$^2$We assume the bus is full-duplex as was assumed by [14]. Thus, only the core-to-cache requests can be delayed. For the simplicity of illustrations, cache-to-core ones are not shown in any figure of this paper.


Figure 8: The worst-case bank access scenario of $C_2$, $C_5$, $C_6$, and $C_8$ which share $B_k$. HRR : (4, 4, 12, 12, 12, 12, 12, 12).

E. Problem Description

For a given multicore system, 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.,

$$\text{Minimize } \sum_{i=1}^{N} \frac{w_{cel_i}}{p_i}. \quad (4)$$

Low system utilization is generally preferred in system development since 1) a lower-utilized system can be more utilized by accommodating additional tasks or 2) the same task set can be implemented with lower-speed cores, which can reduce the unit cost of production. In Sec. IV, we will present the MILP formulation for this optimization problem.

III. Tunable WCET Analysis

In this section, we explain in detail how to find the worst-case bus access, $d_i^B$, and bank conflict delays, $d_i^M$, in Eq. (3).

A. Bus Access Delay $d_i^B$

The worst-case bus access delay is defined as the maximum length of time that a bus request should wait until it is granted. A pure round-robin schedule can bound it by $N^C \cdot L_B$, and thus $d_i^B$ is independent of which core $\tau_i$ is allocated to. In an HRR schedule, on the other hand, task allocation is a delay factor since cores may have different HRR periods. Thus, if $\tau_i$ runs on $C_j$ whose HRR period is $T_j$, a bus access from $\tau_i$ can be delayed at most $T_j$ bus slots in the worst-case. Accordingly,

$$d_i^B = T_j \cdot L_B. \quad (5)$$

B. Bank Conflict Delay $d_i^M$

Let us suppose that $\tau_i$ runs on $C_j$. By the system model assumed in Sec. II-A, $C_j$ uses a contiguous subset of banks, $B_j = \{B_j^1, B_j^{s+1}, \cdots, B_j^{s+n_j-1}\}$, where $n_j^B$ is the number of banks required by $C_j$, which depends on the total number of columns required by all tasks in $C_j$. Recall that only the leftmost and rightmost banks, i.e., $B_j^1$ and $B_j^{s+n_j-1}$, can be shared with others as illustrated in Fig. 5. For simplicity of notation, let us denote $B_j^{s+n_j-1}$ by $B_j^e$.

To identify which banks $\tau_i$ uses, let us denote its cache columns as $X_i = \{X_i^k, X_i^{k+1}, \cdots, X_i^{k+N_i^2-1}\}$, where $N_i^2$ is the number of columns required by $\tau_i$. Then, one of the following cases holds:

- Case 1. a subset or all of $X_i$ reside in $B_j^1$, but not in $B_j^e$.
- Case 2. a subset or all of $X_i$ reside in $B_j^e$, but not in $B_j^1$.
- Case 3. none of $X_i$ reside in either $B_j^1$ or $B_j^e$.
- Case 4. $X_i$ stretch from $B_j^1$ to $B_j^e$.

The upper-bound of bank conflict delays can vary in different cases. For example, in Fig. 5, $\tau_C$ is free from bank conflict interferences since all of its columns are in $C_j$’s BCD-free banks (Case 3). On the other hand, $\tau_A$ and $\tau_B$ use the shared bank $k$ (Case 1–2), and thus could experience bank conflict delays. Meanwhile, the columns of $\tau_i$ may stretch from $B_j^1$ to $B_j^e$ (Case 4). In this case, all accesses of $\tau_i$ are assumed to experience the worst-case delay, which will depend on the bank conflict delays of $B_j^1$ and $B_j^e$.

Now let $D_j^1$ and $D_j^e$ be the upper-bounds of bank conflict delays that a task on $C_j$ could experience when accessing $B_j^1$ and $B_j^e$, respectively. If Case 1 holds for $\tau_i$, the worst-case bank conflict delay that $\tau_i$ could suffer, i.e., $d_i^M$, is $D_j^1$. Similarly, $d_i^M$ for Case 2 is $D_j^e$. For Case 3, $d_i^M$ is always 0. Lastly, $d_i^M$ for Case 4 is the maximum of $D_j^1$ and $D_j^e$.

Accordingly, $d_i^M$ can be expressed by the following equation:

$$d_i^M = \max(\delta_{j,s}^1, D_j^1, \delta_{j,e}^e, D_j^e), \quad (6)$$

where $\delta_{j,s}^1$ and $\delta_{j,e}^e$ are assumed in analyzing the WCET of a task. Thus, if $\tau_i$ uses both shared and non-shared banks, e.g., Task $B$ in Fig. 5, for the tractability of the analysis, we assume that every access of $\tau_i$ goes to the shared bank in the worst-case.
use slot $\phi$, $d_{j,\varphi}^n$ 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 $d_{j,\varphi+T_{RR}}^n$ is always equal to $d_{j,\varphi}^n$ except for $\varphi = \phi_j$, where $\phi_j$ is the first slot index of $C_j$ in a given HRR table. For $\varphi = \phi_j$, $d_{j,\varphi}^n \leq d_{j,\varphi+T_{RR}}^n$ always holds.

**Proof:** If $C_j$ does not use slot $\varphi$, then $d_{j,\varphi+T_{RR}}^n$ is always 0 for all $i = 1, 2, \ldots$. Thus, in what follows, let us consider the case when $C_j$ uses slot $\varphi$. We will prove i) $d_{j,\varphi}^n \geq d_{j,\varphi+T_{RR}}^n$ and ii) $d_{j,\varphi}^n \leq d_{j,\varphi+T_{RR}}^n$. For the simplicity of notations, let us denote $d_{j,\varphi}$ and $d_{j,\varphi+T_{RR}}$ by $d_1$ and $d_2$, respectively, as shown in Fig. 9.

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 $\varphi_x$ ($\varphi < \varphi_x < \varphi + T_{RR}$), where the most recent accumulation of bank accesses begins. Now let $n_y$ be the number of bank accesses initiated in $[\varphi_x, \varphi + T_{RR} - 1]$. Then,

$$d_2 = \varphi_x + L_B + n_y \cdot L_M - (\varphi + T_{RR} + L_B) = \varphi_x + n_y \cdot L_M - \varphi - T_{RR}.$$  

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,

$$d_1 = \varphi_x - T_{RR} + L_B + d_y + n_y \cdot L_M - (\varphi + L_B) = \varphi_x - T_{RR} + n_x \cdot L_M - \varphi + d_y = d_2 + d_y.$$  

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,

$$\varphi_x - T_{RR} + L_B + d_y + n_y \cdot L_M - (\varphi + L_B) \leq 0 \Rightarrow \varphi_x - T_{RR} + n_x \cdot L_M - \varphi + d_y \leq 0 \Rightarrow d_2 \leq -d_y,$$

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 similarly with the above arguments. 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,

$$d_2 = \varphi_x + T_{RR} + L_B + d_y + n_y \cdot L_M - (\varphi + T_{RR} + L_B) = \varphi_x + n_x \cdot L_M - \varphi + d_y = d_1 + d_y > 0$$

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., $d_{j,\varphi}^n = d_{j,\varphi+T_{RR}}^n$ 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, $\varphi_x (< \varphi_j)$ always exists if $d_1 > 0$, and $d_1 \leq d_2$ always holds if $d_1 = 0$. Thus, we can conclude that only $d_{j,\varphi}^n \leq d_{j,\varphi+T_{RR}}^n$ holds for $\varphi = \phi_j$.

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

$$D_j^k = \max(d_{j,\varphi}^n),$$

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

2) Computation of $d_{j,\varphi}^n$: Let us first denote the $\varphi^{th}$ bus slot as $\sigma_{\varphi}$. Each slot delay $d_{j,\varphi}^n$ is affected by how much unfinished bank accesses have been accumulated until $\sigma_{\varphi}$. To model such busy period, we define $w_{j,\varphi}$ as the time instant at which the most recent access to the shared bank completes before $\sigma_{\varphi}$. To help to understand how to find $w_{j,\varphi}$, let us consider the example in Fig. 8 again, and suppose that we want to compute $w_{2,22}$. The busy period begins from the bank access of $C_2$ at $\sigma_{18}$ where there is no backlog of accesses to $B_k$. Thus, the initial busy period, $w_{0,22}^0$, is $18 + L_B + L_M = 21 \cdot L_B$. It delays the access from $\sigma_{19}$ by $w_{2,22}^0 = (19 + L_B)$, and which results in

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

Likewise, the access from $C_0$ at the next slot is delayed by the accumulated delay, which makes the busy period grow to

$$w_{2,22}^2 = 20 + L_B + (w_{2,22}^1 - (20 + L_B)) + L_M = w_{2,22}^1 + L_M = 25 \cdot L_B.$$  

The busy period stops growing at $\sigma_{21}$ because $C_1$ does not use $B_k$, thus the final value of $w_{2,22}$ ends up being $25 \cdot L_B$. As has been seen, the busy period grows by $L_M$ for every new access. However, when it is discontinued, a new busy period continues from a new access.

**Lemma 2.** The worst-case busy period that can delay $C_j$ at $\sigma_{\varphi}$ accessing $B_k$, $w_{j,\varphi}$, can be found by the following iterative
If the new access from \( \phi \) does not use \( B_k \), \( w_{j, \phi}^{i+1} = w_{j, \phi}^i \). Otherwise, \( w_{j, \phi}^{i+1} = \begin{cases} w_{j, \phi}^i + L_M & \text{if } \psi + L_B < w_{j, \phi}^i, \\ \psi + L_B + L_M & \text{otherwise.} \end{cases} \) (9)

The procedure loops from \( \psi = \varphi - T_j \) to \( \varphi - 1 \), and thus \( w_{j, \phi} = w_{j, \phi}^{T_j-1} \). If \( \varphi = \phi_j \), the procedure loops from \( \psi = 1 \) to \( \phi_j - 1 \), and \( w_{j, \phi} = w_{j, \phi}^{\phi_j-1} \).

Proof: We first show that the sequence of \( w_{j, \phi}^i \) is non-decreasing. Let us consider slot \( \sigma_\psi \). If \( C_j \) at \( \sigma_\psi \) does not access \( B_k \), the busy period remains unchanged, i.e., \( w_{j, \sigma_\psi}^{i+1} = w_{j, \sigma_\psi}^i \). Otherwise, \( w_{j, \sigma_\psi}^{i+1} \geq w_{j, \sigma_\psi}^i + L_M \) holds due to the following:

i) If the new access from \( \sigma_\psi \) is initiated before the busy period \( w_{j, \sigma_\psi}^i \) ends, it grows by \( L_M \). Thus, \( w_{j, \sigma_\psi}^{i+1} = w_{j, \sigma_\psi}^i + L_M \).

ii) If the new access from \( \sigma_\psi \) is initiated at or after the end of busy period \( w_{j, \sigma_\psi}^i \), i.e., \( \psi + L_B \geq w_{j, \sigma_\psi}^i \), then,

\[
w_{j, \sigma_\psi}^{i+1} = \psi + L_B + L_M \geq w_{j, \sigma_\psi}^i + L_M.
\]

Therefore, \( w_{j, \sigma_\psi}^{i+1} \geq w_{j, \sigma_\psi}^i \) always holds.

Now we will show that \( w_{j, \phi}^i \leq w_{j, \phi}^i \) holds for all \( i \). First of all, as described in Fig. 10, the initial busy period, \( w_{j, \phi}^0 \), can be derived from \( d_{j, \phi, T_j}^0 \) by the following equality:

\[
w_{j, \phi}^0 = w_{j, \phi, T_j}^0 + L_M = \varphi - T_j + L_B + d_{j, \phi, T_j}^0 + L_M.
\]

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 \((w_{j, \phi}^0, w_{j, \phi}^1, \ldots, w_{j, \phi}^{T_j-1})\). Since the sequence of \( w_{j, \phi}^i \) is non-decreasing,

\[
w_{j, \phi}^0 \leq w_{j, \phi}^1 \leq \cdots \leq w_{j, \phi}^{T_j-2} \leq w_{j, \phi}^{T_j-1} = w_{j, \phi}^i.
\]

Thus, \( w_{j, \phi}^i \leq w_{j, \phi}^i \) holds for all \( i = 0, 1, \ldots, T_j - 1 \). Similarly, if \( \varphi = \phi_j \),

\[
w_{j, \phi}^0 \leq w_{j, \phi}^1 \leq \cdots \leq w_{j, \phi}^{\phi_j-2} \leq w_{j, \phi}^{\phi_j-1} = w_{j, \phi}^i.
\]

Therefore, \( w_{j, \phi} \) calculated by the above procedure is the upper-bound of the busy period that can delay \( C_j \) at \( \sigma_\phi \).

Algorithm 1 calculates \( w_{j, \phi} \).

Once \( w_{j, \phi} \) is obtained, \( d_{j, \phi}^p \) can be computed by

\[
d_{j, \phi}^p = \max(w_{j, \phi} - (\varphi + L_B), 0),
\]

as illustrated in Fig. 10.

**Theorem 1.** \( d_{j, \phi}^M \) computed by Eq. (6)–(10) is 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 \( C_j \) for each bank \( B_k \) in \( B_3 \) are first computed by Eq. (8)–(10). Then, by Eq. (7), we can find the worst-case bank conflict delay \( D_j^k \) that any task in \( C_j \) could suffer due to using \( B_k \). Finally, \( d_{j, \phi}^M \) can be found by Eq. (6), of which value depends on whether \( \tau_i \) uses any shared bank of \( C_j \), \( B_{j_1} \) and/or \( B_{j_2} \).
Table I: List of system parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N^T)</td>
<td>number of tasks</td>
</tr>
<tr>
<td>(N^C)</td>
<td>number of cores</td>
</tr>
<tr>
<td>(N^B)</td>
<td>number of banks of the cache</td>
</tr>
<tr>
<td>(N^W)</td>
<td>number of columns in a bank</td>
</tr>
<tr>
<td>(N^X)</td>
<td>number of columns of the cache</td>
</tr>
<tr>
<td>(N^M)</td>
<td>number of columns required for task (i)</td>
</tr>
<tr>
<td>(T^\text{min}, T^\text{max})</td>
<td>lower- and upper-limit of HRR periods</td>
</tr>
<tr>
<td>(L_B, L_M)</td>
<td>bus and bank access latency</td>
</tr>
</tbody>
</table>

B. Objective Function

The optimization objective we consider in this paper is to minimize the overall system utilization, that is,

\[
\text{Minimize } \sum_{i=1}^{N^T} \frac{\text{wcet}_i}{p_i}.
\]

C. Constraints

1) Harmonic round-robin: Firstly, \(T_j\) has to be an integer in the range of \([T^\text{min}, T^\text{max}]\), thus it is expressed as follows:

\[
\forall \text{ core } j, \quad T_j = \sum_{p=T^\text{min}}^{T^\text{max}} p \cdot \lambda_{j,p}, \quad 0 \leq \lambda_{j,p} \leq 1.
\]

where \(\lambda_{j,p}\) is an indicator variable that is 1 if \(T_j\) has the value of \(p + T^\text{min} - 1\) and 0 otherwise. Meanwhile, the sum of all \(\lambda_{j,p}\) should be equal to 1, that is,

\[
\sum_{p=T^\text{min}}^{T^\text{max}} \lambda_{j,p} = 1, \quad \forall \text{ core } j.
\]

Secondly, each \(T_{j+1}\) is a positive integer multiple of \(T_j\), i.e., \(m_j = \frac{T_{j+1}}{T_j} \in \mathbb{N}\), where \(1 \leq m_j \leq \frac{T^\text{max}}{T^\text{min}}\). Accordingly,

\[
\forall \text{ core } j, \quad m_j = \sum_{q=1}^{m^\text{max}} q \cdot \lambda_{j,q}, \quad \sum_{q=1}^{m^\text{max}} \lambda_{j,q} = 1,
\]

where \(m^\text{max}\) is \(\frac{T^\text{max}}{T^\text{min}}\), and \(\lambda_{j,q}\) is an indicator variable that is 1 if \(m_j\) has the value of \(q\) and 0 otherwise. Now if both \(\lambda_{j,p}\) and \(\lambda_{j,q}\) are 1, the value of \(T_{j+1}\) should be \(T_j \cdot m_j = (p + T^\text{min} - 1) \cdot q\), but not exceed \(T^\text{max}\). Thus, the following conditional constraints are needed:

\[
\forall \text{ core } j, \forall 1 \leq p \leq \frac{T^\text{max} - T^\text{min} + 1}{T^\text{min}}, \forall 1 \leq q \leq m^\text{max}, \lambda_{j,q} = 1 \Rightarrow \lambda_{j+1,(p+T^\text{min} - 1) \cdot q - T^\text{min} + 1} = 1, \quad (p + T^\text{min} - 1) \cdot q \geq T^\text{max} + 1 \Rightarrow \lambda_{j,q} = 0.
\]

Lastly, the sum of the reciprocals of HRR periods should be 1. Now let \(O_j\) be the reciprocal of \(T_j\). Then, by substituting \(O_j\) for \(T_j\) in C1, \(O_j\) can be expressed as follows:

\[
\forall \text{ core } j, \quad O_j = \sum_{p=T^\text{min}}^{T^\text{max}} \frac{1}{p} \cdot \lambda_{j,(p-T^\text{min}+1)}.
\]

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

\[
\sum_{j=1}^{N^C} O_j = 1.
\]

2) Bus scheduling table: Building an HRR table is equivalent to assigning a set of \(\sigma_{j,s}\) to each \(C_j\). An example of \(\sigma_{j,s}\) assignment is shown in Fig. 11.

\[
\forall \text{ slot } s, \sum_{j=1}^{N^C} \sigma_{j,s} = 1.
\]

The first slot of \(C_j\) can appear only after at least one slot is assigned to each core \(C_1, \ldots, C_{j-1}\). Thus,

\[
\forall \text{ core } j \forall \text{ slot } s, \quad s \leq j - 1 \Rightarrow \sigma_{j,s} = 0.
\]

Lastly, the periodicity of the slots of \(C_j\) can be ensured by checking the sum of every \(p=(T_j)\) consecutive \(\sigma_{s,p}\) of \(C_j\). For example, in Fig. 11, the sum of four consecutive \(\sigma_{s,p}\) should be 1 as \(T_j = 4\) is four. Accordingly,

\[
\forall \text{ core } j, \forall T^\text{min} \leq p \leq T^\text{max}, \text{ and } \forall 1 \leq s \leq T^\text{max} - p + 1, \sum_{t=s}^{s+p-1} \sigma_{j,t} = 1.
\]

3) Task to core mapping: Every task should be allocated to one of \(N^C\) cores, thus,

\[
\forall \text{ task } i, \sum_{j=1}^{N^C} \alpha_{i,j} = 1.
\]

4) Core to bank mapping: The minimum number of cache banks required by \(C_j\) is \(n^B_j = \sum_{t=1}^{N^C} (\alpha_{i,t} \cdot (N^T C^\sigma / N^W))\). If \(n^B_j = n + \frac{1}{N^W}, n + 1\) banks are sufficient. However, if \(n^B_j \geq n + \frac{2}{N^W}, n + 2\) banks can be required by \(C_j\):

\[
\forall \text{ core } j, \quad n^B_j \leq (b^j_l + b^j_r + 1) \leq n^B_j + 2 - \frac{2}{N^W}.
\]

If \(n^B_j = 0\), however, no banks should be allocated to \(C_j\):

\[
\forall \text{ core } j, \quad n^B_j = 0 \Rightarrow b^j_l = b^j_r = N^B + 1.
\]

If \(C_j\) uses \(B_k, \beta_{j,k}\) should be set to 1:

\[
\forall \text{ core } j \forall \text{ bank } k, \quad b^j_k \leq k \leq b^j_r \Rightarrow \beta_{j,k} = 1.
\]

Any two cores can share at most one bank, which is either the first or the last bank of each core (see Fig. 5):
Finally, the unbounded bank conflict delay problem defined by Condition (2) in Sec. II-C should be prevented:

C17. ∀ bank \( k \), \( \frac{L_M}{L_B} \sum_{j=1}^{N^C} \beta_{j,k} \cdot O_j \leq 1 \).

Here, 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, \( f_{j,k} \):

\[
\begin{align*}
C17^* & : \forall \text{ core } j \forall \text{ bank } k, \\
& \text{ (a) } f_{j,k} \leq U_{O_j} \cdot \beta_{j,k}, \\
& \text{ (b) } f_{j,k} \leq O_j, \\
& \text{ (c) } f_{j,k} \geq O_j - U_{O_j} \cdot (1 - \beta_{j,k}), \\
& \text{ (d) } f_{j,k} \geq 0,
\end{align*}
\]

where \( U_{O_j} \) is the upper-bound of \( O_j \), which is \( \frac{1}{\Gamma_{\text{min}}} \). 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 [17].

5) Task to column mapping: \( \tau_i \) requires \( N^X \) columns,

C18. ∀ task \( i \), \( x^i_e - x^i_s + 1 = N^X \).

No column can be shared between any two tasks:

C19. For each task \( i \leq i \leq N^T \), and \( i + 1 \leq i' \leq N^T \), \[ x^i_{e'} \leq x^i_e - 1 \] or \( x^i_{e'} + 1 \leq x^i_{e'} \).

\( \tau_i \) on \( C_j \) can occupy only a contiguous subset of the columns belonging to \( C_j \)'s banks, i.e., \( [b^i_{s,k}, b^i_{t,k}] \). Thus,

C20. ∀ task \( i \) ∀ core \( j \), \[ \alpha_{i,j} = 1 \Rightarrow (b^i_{s,k} - 1) \cdot N^W + 1 \leq x^i_s \leq b^i_{t,k} \cdot N^W. \]

6) WCET calculation: The worst-case execution time of \( \tau_i \), i.e., Eq. (3) in Sec. II-D, is formulated as follows:

C21. ∀ task \( i \), \[ \text{wcet}_i = e_i + \sum_{j=1}^{N^C} \alpha_{i,j} \cdot T_j. \]

7) Bus Access Delay \( d^B_i \): Eq. (5) in Sec. III-A can be formulated as follows:

C22. ∀ task \( i \), \[ d^B_i = L_B \sum_{j=1}^{N^C} \alpha_{i,j} \cdot T_j. \]

Note that C22 can be similarly linearized as in C17*, but now \( U_{T_j} \) is \( T_{\text{max}} \).

8) Bank Conflict Delay \( d^M_i \): Let \( w_{s',s,k} \) be the residual workload generated in \( [s', s-1] \) that could delay a bank access from slot \( s \) to \( B_k \), which can be represented as follows:

C23. ∀ bank \( k \), ∀ \( T_{RR} + 1 \leq s \leq 2 \cdot T_{RR} \), ∀ \( 1 \leq s' \leq s-1 \),

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

Note that \( \sigma_{j,t} \cdot \beta_{j,k} \) is 1 if and only if \( C_j \) at slot \( t \) uses \( B_k \). 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, \( g_{j,k,t} \):

\[
\begin{align*}
C23^* & : \forall \text{ core } j \forall \text{ bank } k, \forall 1 \leq t \leq 2 \cdot T_{RR} - 1, \\
& g_{j,k,t} \leq \sigma_{j,t}, \\
& g_{j,k,t} \leq \beta_{j,k}, \\
& g_{j,k,t} \geq \sigma_{j,t} \cdot \beta_{j,k} - 1.
\end{align*}
\]

\( g_{j,k,t} \) 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 that every slot accessing \( B_k \) generates \( L_M \) workload and \( L_B \) is consumed by each slot afterward. The example in Fig. 12 shows possible busy periods that could delay the bank access from \( C_j \) at slot 14. Among others, \( w_{8,14,k} \) is the unique and exact residual workload that maximally delays the slot, since it counts from a no-backlogged slot and also there is no discontinuity in its busy period. Also note that we do not need to compute the slot delays in the first round, as explained in Sec. III-B1.

Now let \( u_{s,k} \) be the worst-case slot delay that slot \( s \) could experience when accessing \( B_k \), which is simply the maximum of \( w_{s',s,k} \) for all \( 1 \leq s' \leq s - 1 \). Thus,

C24. ∀ bank \( k \), ∀ \( T_{RR} + 1 \leq s \leq 2 \cdot T_{RR} \), ∀ \( 1 \leq s' \leq s-1 \), \[ u_{s,k} \geq w_{s',s,k}. \]

\( u_{s,k} \) should be lower-bounded by 0 since the core at slot \( s \) may not use \( B_k \) or not share it with others.

Now let us define by \( z_{j,k} \) the maximum of slot delays of \( C_j \) using \( B_k \), i.e., \( D_{s,k} \). The following constraint is equivalent to Eq. (7) in Sec. III-B1:

C25. ∀ core \( j \) ∀ bank \( k \), ∀ \( T_{RR} + 1 \leq s \leq 2 \cdot T_{RR} \), \[ z_{j,k} \geq \max (D_{s,k}). \]

Note that C25 can be similarly linearized as in C17*, but now \( U_{w_{s,k}} \) is \( (L_M - L_B) \cdot (s - 1) \).

As explained in Sec. III-B, in order to find \( d^M_i \), we need to compute \( D^j_s \) and \( D^j_e \) first. The following constraints can be used to find \( D^j_s \) and \( D^j_e \) from \( z_{j,k} \):

C26–27. ∀ core \( j \) ∀ bank \( k \), \[ b^i_{s,k} = k \Rightarrow D^j_s = z_{j,k}, \quad b^i_{t,k} = k \Rightarrow D^j_e = z_{j,k}. \]

\( \tau_i \) on \( C_j \) could experience bank conflict delays if it uses \( C_j \)'s shared banks. The following constraints can finally find the worst-case bank conflict delay that \( \tau_i \) on \( C_j \) can suffer:

C28–29. ∀ task \( i \) ∀ core \( j \) ∀ bank \( k \), \[ \alpha_{i,j} = 1 \land x^i_s \leq b^i_{s,k} \cdot N^W \quad \Rightarrow \quad d^M_i \geq D^j_s, \]
[\[ \alpha_{i,j} = 1 \land (b^i_{e,k} - 1) \cdot N^W + 1 \leq x^i_{s'} \quad \Rightarrow \quad d^M_i \geq D^j_e. \]

Note here that if \( [x^i_s, x^i_{s'}] \) stretch across \( [b^i_{s,k}, b^i_{t,k}] \), \( d^M_i \) results in \( \max(D^j_s, D^j_e) \) by these constraints.

9) Task and core utilization: To bound the WCET of \( \tau_i \), \( \text{wcet}_i \) should be restricted within its period, i.e., \( p_i \):

C30. ∀ task \( i \), \[ \text{wcet}_i \leq p_i. \]
Table II: Experimental parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N^C_i$</td>
<td>${4, 6, 8}$ cores</td>
</tr>
<tr>
<td>$N^B \times N^W$</td>
<td>${4 \times 16, 8 \times 32}$ columns</td>
</tr>
<tr>
<td>$L_B, L_M$</td>
<td>2.5 ns, 5.0 ns</td>
</tr>
<tr>
<td>$N^F$</td>
<td>${20, 30, 40}$ tasks</td>
</tr>
<tr>
<td>$e_i$</td>
<td>uniform from $[10, 250]$ ms</td>
</tr>
<tr>
<td>$p_i$</td>
<td>uniform from $[500, 10000]$ ms</td>
</tr>
<tr>
<td>$N^X_i$</td>
<td>uniform from $[1, 5]$ columns</td>
</tr>
<tr>
<td>$N^M_i$</td>
<td>uniform from $[10^5, 10^8 \cdot {1, 3, 5, 7, 10}]$ times</td>
</tr>
</tbody>
</table>

Likewise, we need to limit each core utilization to 1, or to a specific bound, e.g., Liu and Layland’s bound [18].

$C31$, $\forall$ core $j$, $\sum_{i=1}^{N^F} \frac{\text{wce}t_i}{p_i} \cdot \alpha_{i,j} \leq 1$.

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

V. EVALUATION

In this section, we evaluate the proposed tunable WCET optimization problem formulated in Sec. IV in terms of the minimum achievable system utilization by using IBM ILOG CPLEX 12.1 [19]. The detailed results can be found in [16].

A. Evaluation Method

Table II summarizes the experimental parameters used for the experiments. With these parameters, we compare the following three methods:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>PureRR</th>
<th>BFD</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task allocation</td>
<td>flexible</td>
<td>pre-allocated</td>
<td>flexible</td>
</tr>
<tr>
<td>Bus schedule</td>
<td>PRR</td>
<td>HRR</td>
<td>HRR</td>
</tr>
<tr>
<td>Cache partition</td>
<td>two-level and flexible</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **PureRR**: The bus is scheduled by a pure round-robin; every core has the same slot period, i.e., $N^C$, as in [14].
- **BFD**: Each task is pre-assigned to a core by Best-Fit Decreasing heuristic. For this, tasks are first sorted in decreasing order by estimated task utilization, which is defined as

$$\text{wce}t_i/p_i = \left[ e_i + N^M_i \cdot \left\{ 2 \cdot L_B + L_M + (N^C \cdot L_B) \right\} \right]/p_i.$$  

Each task is then allocated to the most crowded core from which the task will have the least remaining utilization. Note that $\text{wce}t_i$ is computed by assuming $\tau_i$ does not experience any bank conflict delay and the bus is scheduled by a pure round-robin. However, the bus schedule may change to a harmonic one during optimization.

- **Proposed**: This is the proposed method in this paper.

Note that in all the methods, the shared cache is not pre-partitioned since the unbounded bank conflict delay problem may arise with a random or fixed pre-partitioning.

1) Evaluation metric: We compare the above methods in terms of minimum achievable system utilization, $U_{\text{PureRR}}$, $U_{\text{BFD}}$, and $U_{\text{Proposed}}$. Note that different task sets may have different baseline system utilizations. Thus, for fair comparisons, we normalize $U_{\text{BFD}}$ and $U_{\text{Proposed}}$ to $U_{\text{PureRR}}$ for each task set and then take the average of 20 random sets. Each error bar indicates the standard deviation of the normals.

B. Evaluation Result

In this section, we present the evaluation results for the above methods obtained with different 1) core counts, 2) cache access intensities, and 3) cache configurations.

1) Impact of core count: Fig. 13 compares the minimum system utilization as increasing the number of cores. In this experiment, the cache is configured as 8 banks $\times$ 16 columns, and $N^M_i$ of tasks are randomly chosen within $[10^5, 10^7]$. Also, in order to maintain average load for the cores, we proportionally increase $N^F$ as the core count increases.

As the result shows, our proposed method can achieve lower system utilization than PureRR by 10%–20% in average. We can see the improvement of proposed method compared to PureRR increases with core count. This is because the number of bank-sharing among cores increases with the core and task counts due to the fixed capacity of the shared cache. Another important factor is that with more cores an HRR schedule can be more flexible in prioritizing the cores. Thus, high-utilization tasks can benefit from being allocated to such cores whose HRR periods are short. Meanwhile, the improvement gap between Proposed and BFD can be explained in a similar manner, that is, pre-task assignments prevent further optimization by tightening the constraints on the bus schedule and bank-sharing. Another interesting observation in the result 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 HRR bus scheduling, which in turn helps reduce bank conflict delays as described in Sec. II-C.

2) Impact of cache access intensity: In order to see the impact of cache access intensity on the utilization improvement, we perform another experiment by increasing the upper-limit number of cache accesses, $N^M_i$. In this experiment, we fix $N^C = 8$, $N^F = 40$, and $N^B \times N^W = 8 \times 16$, and vary $N^M_i$ from 1 million to 10 million. With these parameters, $N^M_i$ for each task is chosen randomly between 0.1 million and $N^M_i$.

Fig. 14 shows that the utilization improvement of Proposed over PureRR increases with the cache access intensity. This can be explained by the underlying rationale behind our tunable WCET model. That is, the possibility of further
optimization grows with the ratio of tunable delay to the fixed execution time. Thus if tasks access the bus and cache more intensively, it is more likely that the system utilization can be further reduced by a similar argument explained in the previous discussion. However, this does not necessarily imply that higher cache access intensity would always lead to lower system utilization; \( \frac{U_{\text{Proposed}}}{U_{\text{PureRR}}} \) and \( \frac{U_{\text{BFD}}}{U_{\text{PureRR}}} \) converge to certain levels (around 0.8 and 0.85, 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 partition 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 can be limited because it is more likely for some tasks or cores to exceed the utilization bound, i.e., C30–C31 in Sec. IV.

3) Impact of cache configuration: Fig. 15 shows the impact of different cache configurations on the utilization improvement. For this, we fix \( N^C = 8 \), \( N^T = 40 \), and \( N^M = 10 \) million, and vary the shared cache configurations: 4 banks \( \times \) 32 columns and 8 banks \( \times \) 16 columns.

The result shows that the utilization improvements of Proposed and BFD over PureRR with 8 \( \times \) 16 cache are slightly higher than those with 4 \( \times \) 32 one, even though the total number of columns required by all tasks is similar between two cases. This difference arises mainly due to the different granularity of core-to-bank mappings. To put it clearly, let us consider a core that requires 35 columns. With the 8 \( \times \) 16 cache, the core can fit into 3 out of 8 banks. With the 4 \( \times \) 32 cache, on the other hand, the core needs at least 2 but possibly 3 out of 4 banks, which is equivalent to 4 or 6 banks of the 8 \( \times \) 16 cache. Accordingly, each core is likely to take more banks than it actually needs with a fewer-banks cache, which results in an increased number of bank-sharing. Another factor that influences the difference is that with more cores bank conflict delays can further be reduced by the help of a harmonic round-robin schedule, as previously discussed.

VI. Related Work

A great deal of research effort has been devoted to address the optimization of shared resource allocation and arbitration in multicore architectures. For on-chip memory partitioning, Suhendra et al. [20] proposed an ILP formulation that finds the optimal scratchpad memory partition and task allocation/scheduling which minimize tasks’ execution times. In [21], the authors examined the impacts of different combinations of cache locking and partitioning schemes on the system utilization. In [22], Bui et al. proposed a genetic algorithm that can find near optimal cache partition and task-to-partition assignments that minimize the system utilization.

Another line of research has focused on shared bus arbitration methods. Rosén et al. [6] and Andrei et al. [23] 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 [8]. Additionally, Schranzhofer et al. [11] 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 [24], [25].

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 [26] 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 [6], 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.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments and suggestions and Sungjin Im for his help in the MILP formulation. This work is supported in part by a grant from Rockwell Collins, by a grant from Lockheed Martin, and by ONR N00014-08-1-0896. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of sponsors.

REFERENCES