# Reducing Inter-Task Interference Delay by Optimizing Bank-to-Core Mapping*

Jizan Zhang, Zhimin Gu, Mingquan Zhang
School of Computer Science & Technology
Beijing Institute of Technology
Beijing, China
Email: zhangjizan@163.com, zmgu@x263.net, ncepu_zmq@163.com

***Abstract*—Inter-task interferences on the shared resources are the main difficulty in analyzing the timing behavior of multicores. In the timing predictable embedded multicore architecture MERASA, the inter-task interference delay suffered by a Hard Real-time Task (HRT) can be bounded by the number of cores. Although this method can simplify the Worst-Case Execution Time (WCET) estimation, the value of WCET is seriously overestimated. To obtain tighter WCET estimation, we propose a novel approach that reduces the inter-task interference delay by optimizing bank-to-core mapping in the multicore system with the interference-aware bus arbiter and the two-level partitioned cache. For this, we first analyze and compute the inter-task interference delays suffered by the HRTs running simultaneously, and then put forward the core-queue optimization method of bank-to-core mapping and design the optimizing algorithms with the minimum inter-task interference delay. Experimental results demonstrate that our approach can reduce inter-task interference delay and improve estimated WCET.**

***Keywords*—*multicore; hard real-time task; optimization; inter-task interference delay; bank-to-core mapping***

## I. Introduction

The usage of multicore processors, such as ARM11 MP-Core [1] and QorIQ P4080 [2], in hard real-time systems brings new challenges for WCET (Worst-Case Execution Time) analysis [3], [4]. Multicore processors often contain some shared resources, such as the on-chip shared cache and shared bus. Hard Real-time Tasks (HRTs) running simultaneously can interfere with each other when they try to access these shared resources at the same time. Since these interferences can bring unpredictably extra execution time for HRTs, their effects should be taken into account to obtain safe WCET. Unfortunately, well-developed timing analysis techniques for single-core systems are not able to estimate the effects of these inter-task interferences [4], [5].

Shared cache (L2 cache) and shared bus are two important resources of multicore architectures. Although cache locking and partitioning can eliminate storage interference [6], [7] and TDMA-based bus arbitration can be analyzed [8]–[11], most existing literatures ignored the effect of bank conflict delay on WCET estimation.

This work is supported by the national Natural Science Foundation of China under Grant No.61370062
Corresponding Author: Zhimin Gu

The combination of cache partitioning and bus arbitration designing can improve the predictability of hard real-time multicore systems [12]–[15]. For example, MERASA multicore architecture [13] applying the Interference-Aware Bus Arbiter (IABA) and L2 cache dynamic partitioning [14] to bound the Upper Bound Delay (UBD) by the number of cores. Yoon et al. [15] proposed a harmonic round-robin bus arbitration and two-level cache partitioning to bound the upper bound of inter-task interference delay. Although these methods took the effect of bank conflict delay into consideration, a potential maximum delay is added to the time that a request accesses L2 cache and the WCET is seriously overestimated. In fact, not all requests can suffer from inter-task interference delay. Even though inter-task interferences occur among a group of requests, the inter-task interference delay suffered by every request is different.

The goal of this paper is to reduce inter-task interference delay by optimizing bank-to-core mapping, and obtain tighter WCET estimations on the multicore systems with the IABA bus arbiter and two-level partitioned cache. The major contributions of this paper are as follows:

1) Analyze the inter-task interference delay, and then propose the method to compute inter-task interference delay according to the timing sequence of requests asking the bus.
2) Optimize bank-to-core mapping to reduce inter-task interference delay. We propose the method to make bank-to-core mapping according to the queue of cores, and design the algorithm for the optimization problem that the inter-task interference delay is computed according to the timing sequence of requests asking the bus.
3) Propose the method to estimate WCET using the optimized results, and analyze the waiting time of a request in the Intra-Core Bus Arbiter (ICBA).

This paper is organized as follows. Section II introduces related work. Section III describes the background and motivation. In Section IV, we analyze and compute the inter-task interference delay suffered by a HRT. In Section V, we optimize bank-to-core Mapping and design algorithms for the optimization problem. Section VI describes WCET estimation. Section VII describes experimental results. At last, we conclude this paper in Section VIII.

## II. Related Work

Existing literatures proposed various methods to deal with inter-task interference, which can be mainly classified into three categories.

Some put shared bus designing and shared cache partitioning together to bound the upper bound of inter-task interference delay that a request suffers. Paolieri et al. [14] proposed IABA bus arbiter and a dynamic cache partitioning. When more than one request tries to access the same bank, IABA employs a round robin policy to select one request that will access the bus, and delays the other requests to avoid any interference. Dynamic cache partitioning employs bankization to eliminate storage and bank interferences. Yoon et al. [15] proposed harmonic round robin bus arbitration and two-level cache partitioning scheme. Harmonic round robin bus arbitration allocates bus slots to cores and guarantees bank conflict delays are limited in one bus round. In the two-level cache partitioning, banks are mapped to cores and columns to HRTs in a way that bank access conflicts are minimized with the help of the harmonic round robin bus arbitration.

Some put storage interference and bus access interference together to carry on WCET analysis, but did not take bank conflict into consideration. Rosén et al [8] proposed a TDMA-based bus arbitration which allocates bus slots to cores by static scheduling. They analyzed the bus access delay and storage interference delay, and then optimized the bus arbitration policy. Chattopadhyay et al. [11] improved the treatment of loop structures in [9], the authors employed the maximum bus access delay to estimate WCET according to execution contexts instead of aligning each loop head execution to the first TDMA slot. Kelter et al. [10] improved further analysis efficiency via bounding the upper bound of TDMA offsets.

Some only focused on the analysis of storage interference, but did not take bank conflict and bus access interference into consideration. Yan et al. [16] computed the worst-case instruction access interferences between different threads based on the program control flow information of each thread. Chen et al. [17] analyzed the worst-case cache interferences based on instruction fetching timing, while judging the interferences status through instruction fetching timing relations. Ding et al. [6] proposed a flexible dynamic cache locking approach to eliminate storage interference.

Unlike the existing methods, we compute inter-task interference delay according to the request timing sequence, make bank-to-core mapping according to the queue of cores and optimize bank-to-core mapping to reduce inter-task interference delay.

## III. Background and Motivation

In this section, we describe the embedded multicore architecture, multi-task application model and motivation of our optimization approach.

### A. Embedded Multicore Architecture

An embedded multicore architecture consists of $N_{core}$ homogeneous in-order cores $C = \{c_1, c_2, \cdots, c_{N_{core}}\}$. Each core has its own private data and instruction cache. The unified L2 cache shared by all cores is two-level partitioned [15], which is partitioned evenly into $N_{bank}$ banks, $B = \{b_1, b_2, \cdots, b_{N_{bank}}\}$. Bank access latency is 4 cycles, labeled as $L_M$. Each bank is subdivided evenly into $N_{column}$ columns. The real-time bus connecting the cores and the L2 cache applies the IABA bus arbiter [13], [14]. Bus access latency is 2 cycles, labeled as $L_B$. The penalty of L2 cache miss is 30 cycles fixedly.

The IABA bus arbiter comprises one Inter-Core Bus Arbiter (XCBA) and several ICBAs [14]. Each core has one ICBA to schedule among requests from the same core. All the ICBAs apply FIFO arbitration for requests from HRTs and First Ready First Serviced arbitration for requests from Non Hard Real-time Tasks (NHRTs). The XCBA schedules among requests from different cores, first, the requests from HRTs have priority over requests from NHRTs, second, between different requests from HRTs, a round robin policy is applied as well as between different requests from NHRTs [14]. If more than one request tries to access the same bank at same time, the XCBA selects one request to access the bus and delays other requests to avoid inter-task interferences, including bus access interference and bank access conflict.

### B. Multi-task Application Model

A real-time application comprises HRTs and NHRTs, and the HRT set is denoted as $T_{hrt}$. All tasks are partitioned to $N_{core}$ cores based on a non-preemptive partitioned scheduling approach [18]. The HRTs allocated to the same core are executed in turn, and no migration is permitted. Let $C_{hrt}(\subseteq C)$ be the set of cores with HRTs, and the number of the cores in $C_{hrt}$ is $N_{hrt}(\leq N_{core})$. Only one core is only allocated with NHRTs, labeled as $c_{nhrt}(\in C)$. In the worst case, all $N_{hrt}$ cores in $C_{hrt}$ are executing HRTs, i.e., at most $N_{hrt}$ HRTs are running simultaneously.

We first allocate L2 cache for the cores in $C_{hrt}$. The number of columns allocated one core is determined by the HRT with the maximum number of demanded columns. In the HRT set $\Gamma_i(\subseteq T_{hrt})$ allocated to $c_i(\in C_{hrt})$ , $HRT_i(\in \Gamma_i)$ needs $Size_j$ columns. The columns allocated to $c_i$ (denoted as $Sz_{c_i}$) can be expressed as $Sz_{c_i} = max(Size_j | \forall HRT_j \in \Gamma_i)$. The HRTs allocated to one core exclusively use the columns allocated to this core. If $c_{nhrt} \neq \phi$, we allocate $(N_{bank} \cdot N_{column} - \sum_{\forall c_i \in C_{hrt}} Sz_{c_i})$ columns to $c_{nhrt}$. This cache allocation has the following characteristics: (1) no storage interference exists since each HRT exclusively uses the allocated columns, (2) one bank can be shared by more than one core. Bank conflict can occur when the HRTs running simultaneously try to access the same bank, and (3) the different distributions of $N_{bank} \cdot N_{column}$ columns among $N_{core}$ cores correspond to different bank-to-core mappings.

Additionally, we apply the methods mentioned in [19] to deal with the code shared among the HRTs and inter-task communication. If more than one HRT shares a function, we create one separate copy of this function for each HRT. The tasks communicate with each other through message passing via mailboxes.

### C. Motivation

The WCETs estimated by bounding the upper bound of inter-task interference delay, such as Paolieri [14] and Yoon

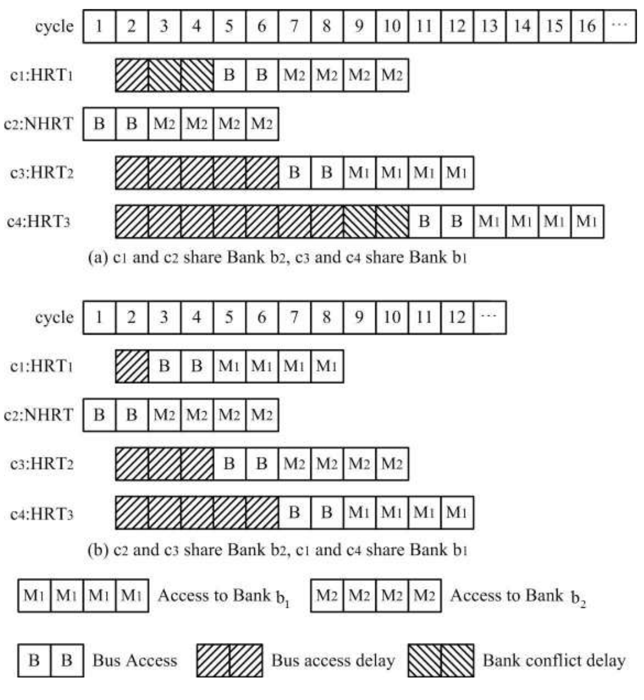


Fig. 1. An example that the inter-task interference delay suffered by HRTs in different bank-to-core mappings

[15], are seriously overestimated which can produce negative effect on the schedulability of HRTs, performance and energy dissipation [18], [20], [21]. Although the bankization of L2 cache can reduce the estimated WCET by eliminating storage and bank conflicts, the bankization may cause bank capacity's waste, what is more, it is unable to meet the needs of many real-time applications.

The two-level cache partitioning scheme can improve the utilization of shared cache, in which banks can be flexibly allocated between cores. If $\sum_{\forall c_i \in C_{hrt}} \lceil Sz_{c_i}/N_{column} \rceil \leq N_{bank}$, $\lceil Sz_{c_i}/N_{column} \rceil$ banks are exclusively allocated to $c_i$, the caching requirement of each HRT is satisfied and no bank conflict exists. If $\sum_{\forall c_i \in C_{hrt}} \lceil Sz_{c_i}/N_{column} \rceil > N_{bank}$, one bank can be shared by more than one core, and bank conflicts occur when the HRTs allocated to these cores try to access this shared bank at the same time. In different bank-to-core mappings, bank access conflict is different since banks can be shared by different cores. For example, in the bank-to-core mapping shown in Fig.1(a), $HRT_1(c_1)$ and $NHRT(c_2)$ share Bank $b_2$, $HRT_2(c_3)$ and $HRT_3(c_4)$ share Bank $b_1$. The requests from $HRT_1(c_1)$ and $HRT_3(c_4)$ can suffer from bank conflict, the inter-task interference delays suffered by the requests from $HRT_1$, $HRT_2$ and $HRT_3$ are 3, 5 and 9 cycles, respectively. In another bank-to-core mapping shown in Fig.1(b), $HRT_2(c_3)$ and $NHRT(c_2)$ share Bank $b_2$, $HRT_1(c_1)$ and $HRT_3(c_4)$ share Bank $b_1$, the inter-task interference delays suffered by the requests from $HRT_1$, $HRT_2$ and $HRT_3$ are 1, 3 and 5 cycles.

A HRT has a timing sequence of requests asking the bus in its worst case execution path. The inter-task interference delay suffered by the HRT can be computed according to the timing sequence and the XCBA's arbitration policy. In this paper, we compute inter-task interference delay suffered by a HRT according to the timing sequence of requests asking the bus in its worst case execution path, and reduce inter-task interference delay by optimizing bank-to-core mapping.

## IV. Analysis and Computation of Inter-Task Interference Delay

Let $QHRT_i$ be the timing sequence of requests asking the bus in the worst case execution path of $HRT_i$. $q_j(\in QHRT_i)$ is a request tries to access Bank $b_k$, the time of $q_j$ asking the bus is $tarr_j$. $Txcba_{sta}$ is the start time of the current scheduling of the XCBA. If $tarr_j < Txcba_{sta}$, $q_j$ reaches the ICBA before $Txcba_{sta}$, and stay in the ICBA until it is sent to XCBA at $Txcba_{sta}$. In this case, we update $tarr_j = Txcba_{sta}$. Let $Q_{cur}$ be the request set in the current schedule of XCBA. We divide two cases to discuss the interference delay suffered by the request $q_j$.

Case (1): No NHRT exists. Let $q_{(j-1)}(\in Q_{cur})$ be the previous request of $q_j(\in Q_{cur})$. The bus access delay suffered by $q_j$ can be expressed as $dbus_j = tacc_{(j-1)} + L_B - Txcba_{sta}$, where $tacc_{(j-1)}$ is the time that $q_{(j-1)}$ is granted to access the bus. $q_r(\in Q_{cur})$ is the previous request of $q_j$ to access Bank $b_k$, the time that $q_r$ is granted to access the bus can be expressed as $tacc_r = Txcba_{sta} + dbus_r + dbank_r$, where $dbank_r$ is the bank conflict delay suffered by $q_r$. if $tacc_r + L_B + L_M > Txcba_{sta} + dbus_j + L_B$, $q_j$ suffers from bank access conflict, and the bank conflict delay suffered by $q_j$ can be expressed as $tacc_r + L_B + L_M - (Txcba_{sta} + dbus_j + L_B)$, i.e., $dbank_j = dbus_r + dbank_r + L_M - dbus_j$. Otherwise, $q_j$ does not suffers from bank access conflict, $dbank_j = 0$.

Case (2): NHRTs exist. The requests from HRTs can suffer from inter-task interference delay caused by the requests from NHRTs, e.g., the inter-task interference delay shown in Fig.1. Let $tacc_{nhrt}$ be the time that the request from a NHRT is granted to access the bus. If $tarr_j < tacc_{nhrt} + L_B$, $q_j$ suffers from bus access interference caused by the request from the NHRT, the bus access delay can be expressed as $dbus_j = tacc_{nhrt} + L_B - tarr_j$. If $HRT_i$ share the same bank with the NHRT and $tacc_{nhrt} + L_B + L_M > tarr_j + dbus_j + L_B$, $q_j$ suffers from bank access conflict caused by the request from the NHRT, and the bank conflict delay can be expressed as $dbank_j = tacc_{nhrt} + L_M - (tarr_j + dbus_j)$.

The total inter-task interference delay suffered by $HRT_i$ can be expressed as $\sum_{\forall q_j \in QHRT_i}(dbus_j + dbank_j)$. Let $Qc_i$ be the timing sequence of the requests from the tasks allocated to $c_i$. If $c_i \in C_{hrt}$, let $Qc'_i(\subseteq Qc_i)$ be the timing sequence of the requests from the HRTs allocated to $c_i$. Algorithm 1 is to compute the interference delay in one schedule of the XCBA. $CurrentQ[i]$ is the current request in $Qc_i(c_i \in C)$. $BtoCMapping[N_{core}][N_{bank}]$ is a bank-to-core mapping. $used[i]$ denotes whether $CurrentQ[i]$ is deal with or not. If $CurrentQ[i]$ has been dealt with, $used[i]$=TRUE. In line 1$\sim$11, a function $Compute\_Bankdelay$ is defined to compute the potential bank conflict delay. Line 12$\sim$18 determine which kind of request should be scheduled. The XCBA schedules the requests from HRTs in line 20$\sim$29. Bus access delay is computed in line 23, and the finish time of the current scheduling is computed in line 29. The requests from NHRTs are dealt with in line 31$\sim$53. In the process of dealing with

**Algorithm 1** Compute the interference delay in a scheduling of the XCBA

```
Require: $C$, $N_{core}$, $B$, $N_{bank}$, $C_{hrt}$, $CurrentQ[i]$, $QC'_i(c_i \in C)$ , $used[i]$, $Txcba_{sta}$, $BtoCMapping[N_{core}][N_{bank}]$
Ensure: bus access delay suffered by a request ($dbus[i]$), bank conflict delay suffered by a request ($dbank[i]$), the bus access delay caused by the requests from NHRTs ($dbusbyNHRT[i]$), the finish time that a request accesses the bus ($FinishQ[i]$), the finish time of the current scheduling of XCBA ($Txcba_{fin}$), $used[i]$
1: function $Compute\_Bankdelay(curr)$ do
2: for $j = 1; j \le N_{bank}; j++$ do
3:     for $k = N_{core}; k \ge 1; k--$ do
4:         if $used[k]$ ==TRUE , and $c_k$ and $c_{curr}$ share Bank $b_j$ in BtoCMapping then
5:             if $dbus[k] + dbank[k] + L_M - dbus[curr] > dbank[curr]$ then
6:                 $dbank[curr] = dbus[k] + dbank[k] + L_M - dbus[curr]$;
7:             end if
8:         end if
9:     end for
10: end for
11: end function
12: $CheckHRT$=FALSE; $CheckNHRT$=FALSE;
13: for $i = 1; i \le N_{core}; i++$ do
14:     if $CurrentQ[i] \le Txcba_{sta}$ then
15:         if$CurrentQ[i] \in Qc'_i$ then $CheckHRT$ =TRUE;
16:         if$CurrentQ[i] \notin Qc'_i$ then $CheckNHRT$ =TRUE;
17:     end if
18: end for
19: $FinTime = Txcba_{sta}$;
20: if $CheckHRT$==TRUE then
21:     for $i = 1; i \le N_{core}; i++$ do
22:         if $CurrentQ[i] \in Qc'_i$ and $CurrentQ[i] == Txcba_{sta}$ then
23:             $dbus[i] = FinTime - Txcba_{sta}$;
24:             CALL $Compute\_Bankdelay(i)$;
25:             $FinTime = FinTime + L_B + dbank[i]$;
26:             $FinishQ[i] = FinTime$; $used[i]$= TRUE;
27:         end if
28:     end for
29:     $Txcba_{fin} = FinTime + L_M$;
30: else
31:     for $i = 1; iNcore; i++$ do
32:         if $CurrentQ[i] \notin Qc'_i$ then
33:             if $CurrentQ[i] \le Txcba_{sta}$ then
34:                 CALL $Compute\_Bankdelay(i)$;
35:                 $FinTime = FinTime + L_B + dbank[i]$;
36:                 $FinishQ[i] = FinTime$; $used[i]$= TRUE;
37:             end if
38:             $MinValue = min(CurrentQ[i] | \forall CurrentQ[i] \in Qc'_i)$;
39:             if $MinValue \le FinTime + L_M$ then
40:                 $CheckHRT$=TRUE;
41:                 for $j = 1; j \le N_{core}; j++$ do
42:                     if $CurrentQ[j] = MinValue$ and $CurrentQ[j] \in Qc'_j$ then
43:                         if $CurrentQ[j] < FinTime$ then $dbusbyNHRT[j] = FinTime - CurrentQ[j]$;
44:                         CALL $Compute\_Bankdelay(j)$;
45:                         $FinTime = FinTime + L_B + dbank[j]$;
46:                         $FinishQ[j] = FinTime; used[j] = TRUE$;
47:                     end if
48:                 end for
49:             end if
50:             if $CheckHRT$==TRUE then break;
51:         end if
52:     end for
53:     $Txcba_{fin} = FinTime + L_M$;
54: end if
55: return $dbus[], dbank[], dbusbyNHRT[], FinishQ[], Txcba_{fin}, used[]$;
```

the requests from NHRTs, deal with the requests from HRTs in line 39∼50 if any, and the bus access delay caused by the requests from NHRTs is computed in line 43.

Algorithm 2 is to compute the inter-task interference delay suffered by HRTs. The first request is taken from $Qc_i$ to $CurrentQ[i]$ in line 6 if $used[i]$=TRUE. If more than one HRT is allocated to the same core, compute the inter-task interference delay suffered by the previous HRT in line 9. In line 15∼17, $CurrentQ[i]$ is updated as $Txcba_{sta}$ if

**Algorithm 2** Compute the inter-task interference delay suffered by HRTs

```
Require: $C$, $N_{core}$, $B$, $N_{bank}$, $C_{hrt}$, $Qc_i$, $Qc'_i(c_i \in C)$
Ensure: the interference delay suffered by each HRT ($TaskDelay[][]$) and the total inter-task interference delay suffered by the HRTs allocated to the same core ($Delay[]$)
1: $Delay[i] = 0$, $CurrentQ[i] = 0$, $FinishQ[i] = 0$, $used[i]$=TRUE, $BusDelay[i] = 0$, $1 \le i \le N_{core}$;
2: $Txcbasta = 0$, $DelayTime[j] = 0$, $BankDelay[i] = 0$, $k_i = 1, 1 \le i \le N_{core}$;
3: while $\exists Qc_i \ne$NULL, $c_i \in C$ do
4:     for $i = 1; i \le N_{core}; i++$ do
5:         if $used[i]$== TRUE and $Qc_i \ne$NULL then
6:             Fetch the first request of $Qc_i$ to $CurrentQ[i]$;
7:             Delete the first request from $Qc_i$;
8:             if $CurrentQ[i]$ and its previous request are from the different tasks then
9:                 $TaskDelay[i][k_i] = BusDelay[i] + DBusbyNHRT[i] + BankDelay[i]$;
10:                $BusDelay[i] = 0$, $DBusbyNHRT[i] = 0$, $BankDelay[i] = 0$, $k_i++$;
11:            end if
12:            $used[i]$=FALSE;
13:        end if
14:    end for
15:    for $i = 1; i \le N_{core}; i++$ do
16:        if $CurrentQ[i] < Txcba_{sta}$ then $CurrentQ[i] = Txcba_{sta}$;
17:    end for
18:    $MinValue = min(CurrentQ[i] | \forall c_i \in C)$;
19:    $Txcbasta = MinValue$;
20:    Call Algorithm 1 to compute the interference delays suffered by $CurrentQ[i]$;
21:    for $i = 1; i \le N_{core}; i++$ do
22:        if $CurrentQ[i] \in Qc'_i$ then
23:            $BusDelay[i] = BusDelay[i] + dbus[i]$;
24:            $BankDelay[i] = BankDelay[i] + dbank[i]$;
25:            $DBusbyNHRT[i] = DBusbyNHRT[i] + dbusbyNHRT[i]$;
26:        end if
27:    end for
28:    $Txcba_{sta} = Txcba_{fin}$;
29: end while
30: $TaskDelay[i][k_i] = BusDelay[i] + DBusbyNHRT[i] + BankDelay[i], c_i \in C_{hrt}$;
31: $Delay[i] = \sum_{j=1}^{k_i} TaskDelay[i][j], c_i \in C_{hrt}$;
32: return $TaskDelay[][], Delay[]$;
```

$CurrentQ[i] < Txcba_{sta}$. $Txcba_{sta}$ is updated in line 18 and 19. In line 20, Algorithm 1 is called to compute bus access delay $dbus[i]$, bank conflict delay $dbank[i]$, and the bus access delay $dbusbyNHRT[i]$ caused by NHRTs if any. Bus access delay and bank conflict delay are updated in line 21∼27. The total inter-task interference delay suffered by the HRTs allocated to the same core is computed in line 31.

## V. Optimizing Bank-to-Core Mapping to Reduce Inter-Task Interference Delay

In this section, we optimize bank-to-core mapping to reduce inter-task interference delay. We describe the optimization problem, and design algorithms for the optimization problem.

### A. Optimization Problem

Let $x_{ik}$ denote whether $b_k (\in B)$ has columns allocated to $c_i (\in C)$. If $b_k$ has columns allocated to $c_i$, $x_{ik} = 1$. Otherwise, $x_{ik} = 0$. The number of columns of $b_k$ allocated to $c_i$ is $ncol_{ik}$. If $x_{ik} = 1$, $ncol_{ik} > 0$. Otherwise, $ncol_{ik} = 0$. $Txcba_{sta}$ and $Txcba_{fin}$ are the start time and finish time of a schedule of XCBA, respectively. $x_{ik}$ and $ncol_{ik}$ are the decision variables.

*1) Objective Function:* Since inter-task interference delay is non-negative, reducing the inter-task interference delay suffered by each HRT is equivalent to reducing the total inter-task interference delay suffered by all HRTs. The objective function can be expressed as follows:

$$min(\sum_{\forall c_i \in C_{hrt}} \sum_{\forall q_j \in Qc'_i} (dbus_j + dbank_j)) \quad (1)$$

*2) Constraints:*

- The start time of the current schedule of XCBA is the finish time of the previous schedule or the earliest time that the current requests ask the bus, which is expressed as follows: if $Txcba_{fin} \geq min(tarr_j | \forall q_j \in Q_{cur})$, $Txcba_{sta} = Txcba_{fin}$; otherwise, $Txcba_{sta} = min(tarr_j | \forall q_j \in Q_{cur})$.
- If the time of a request asking the bus is earlier than the start time of the current schedule of XCBA, update the time of the request asking the bus as the start time of the current schedule, which is expressed as follows:

$$tarr_j = Txcba_{sta}, \text{if } tarr_j < Txcba_{sta}, \forall q_j \in Q_{cur}$$

- The finish time of the current schedule of XCBA is the finish time of the last request in this schedule, which can be expressed as follows:

$$Txcba_{fin} = max(tacc_j + L_B + L_M | \forall q_j \in Q_{cur})$$

- The time that a request is granted to access the bus can be expressed as follows:

$$tacc_j = Txcba_{sta} + dbus_j + dbank_j, \forall q_j \in Q_{cur}$$

- The bus access delay suffered by a request can be expressed as follows: $\forall q_j \in Q_{cur}$, $dbus_j = tacc_{(j-1)} + L_B - tarr_j$ if existing the effect of NHRTs; otherwise, $dbus_j = tacc_{(j-1)} + L_B - Txcba_{sta}$.
- If bank access conflict occurs between two requests ($q_r$ and $q_j (r < j)$) from HRTs, the bank conflict delay suffered by $q_j$ can be expressed as follows: if $dbus_r + dbank_r + L_M - dbus_j > 0$, $dbank_j = dbus_r + dbank_r + L_M - dbus_j$; otherwise, $dbank_j = 0$.
- If bank access conflict occurs between the request $q_j$ from HRT and the requests from NHRTs, the bank conflict delay suffered by $q_j$ can be expressed as follows: if $tacc_{nhrt} + L_M - (tarr_j + dbus_j)$, $dbank_j = tacc_{nhrt} + L_M - (tarr_j + dbus_j)$; otherwise, $dbank_j = 0$.
- The capacity of L2 cache should meet the demand of all cores, that is, $N_{bank} \cdot N_{column} \geq \sum_{i=1}^{N_{core}} Sz_{c_i}$.
- If $c_{nhrt} \neq \phi$, the remainder columns are allocated to it, that is,

$$Sz_{c_{nhrt}} = N_{bank} \cdot N_{column} - \sum_{\forall c_i \in C_{hrt}} Sz_{c_i}$$

- The total columns allocated to a core needs to meet the demand of the core, that is,

$$Sz_{c_i} \leq \sum_{k=1}^{N_{bank}} ncol_{ik} \cdot x_{ik}, \forall c_i \in C_{hrt}$$

- The columns of any bank allocated to all cores are less than or equal to the capacity of the bank, which can be expressed as follows:

$$\sum_{i=1}^{N_{core}} ncol_{ik} \cdot x_{ik} \leq N_{column}, \forall b_k \in B$$

- The demand columns of any core and the columns of any bank allocated to any core are nonnegative, that is, $Sz_{c_i} \geq 0$, $ncol_{ik} \geq 0$, $\forall c_i \in C$, $\forall b_k \in B$.

## B. The Way of Making Bank-to-Core Mapping

There are some different ways of bank-to-core mapping. Different ways can produce different bank-to-core mappings. In this paper, we make bank-to-core mapping according to the queue of cores. Considering the demanded columns of cores, the way of making bank-to-core mapping can be divided into three cases as follows:

Case (1):$\sum_{\forall c_i \in C_{hrt}} \lceil Sz_{c_i} / N_{column} \rceil \leq N_{bank}$. In this case, we exclusively allocate $\lceil Sz_{c_i} / N_{column} \rceil$ banks to the core $c_i (\in C_{hrt})$. If $c_{nhrt} \neq \phi$, we allocate $(N_{bank} - \sum_{\forall c_i \in C_{hrt}} \lceil Sz_{c_i} / N_{column} \rceil)$ banks to $c_{nhrt}$. Since the cores in $C_{hrt}$ exclusively use the allocated banks, all HRTs do not suffer from bank conflict.

Case (2): $\sum_{\forall c_i \in C_{hrt}} \lceil Sz_{c_i} / N_{column} \rceil) > N_{bank}$ and $\sum_{\forall c_i \in C_{hrt}} Sz_{c_i} > (N_{bank} - 1) \cdot N_{column}$. If $c_{nhrt} \neq \phi$, we allocate $(N_{bank} \cdot N_{column} - \sum_{\forall c_i \in C_{hrt}} Sz_{c_i})$ columns to $c_{nhrt}$. The process of making bank-to-core mapping can be described as follows: (1) make bank-to-core mapping according to a core queue. We first allocate columns for the first core in the core queue. Next, we allocate columns for the second core, and so on, and (2) we first allocate the columns of Bank $b_1$ to cores. Next, we allocate the columns of Bank $b_2$, and so on.

Case (3): $\sum_{\forall c_i \in C_{hrt}} \lceil Sz_{c_i} / N_{column} \rceil) > N_{bank}$ and $\sum_{\forall c_i \in C_{hrt}} Sz_{c_i} \leq (N_{bank} - 1) \cdot N_{column}$. If $c_{nhrt} \neq \phi$, we allocate $(N_{bank} \cdot N_{column} - \sum_{\forall c_i \in C_{hrt}} Sz_{c_i})$ columns to $c_{nhrt}$. The process of making bank-to-core mapping is same as that of Case (2). Otherwise, we first reserve $\lfloor (N_{bank} \cdot N_{column} - \sum_{\forall c_i \in C} Sz_{c_i}) / N_{bank} \rfloor$ columns for $((N_{bank} \cdot N_{column} - \sum_{\forall c_i \in C} Sz_{c_i}) \ mod \ N_{bank})$ banks and $\lceil (N_{bank} \cdot N_{column} - \sum_{\forall c_i \in C} Sz_{c_i}) / N_{bank} \rceil$ columns for the rest banks that do not take part in allocation, and then apply the method of Case (2) to make bank-to-core mapping.

## C. The Optimizing Algorithm

It is necessary to know the request timing sequences to compute inter-task interference delay according to the objective function (i.e, Equation (1)). Chronos [22], one static WCET analysis tool for single-core systems, can obtain the latest time that requests ask the bus in each basic block of CFG (Control Flow Graph). With the help of Integer Linear Programming (ILP) solvers, such as lp_solve [23], we can obtain the request timing sequence in the worst case execution path of one HRT. In this paper, we use the request timing sequences obtained by Chronos and lp_solve to compute inter-task interference delay. In order to simplify computation and guarantee that the estimated WCET is safe, we assume that the requests from one core always try to access the shared banks in the worst case.

**Algorithm 3** Optimize bank-to-core mapping to minimize interference delay

```
Require: $C$, $N_{core}$, $C_{hrt}$, $N_{hrt}$, $c_{nhrt}$, $B$, $N_{bank}$, $N_{column}$,$Sz_{c_i}(c_i \in C)$,
    $Qc_i(c_i \in C)$, $Qc'_i(c_i \in C_{hrt})$
Ensure: The minimum interference delay suffered each HRT ($MinTaskDelay[][]$),
    the sum of the minimum interference delay suffered each HRT ($MinTotalDelay$)
    and the corresponding bank-to-core mapping ($MinDelayMap[][]$)
1: $MinTotalDelay = Infinity$, $used[i]$=FALSE, $1 \le i \le N_{core}$;
2: function $FindMinMapping(n)$ do
3: if $n > N_{core}$ then
4:     Make bank-to-core mapping $BtoCMapping[][]$ according to the core queue
       in $c\_seq[]$;
5:     Call Algorithm 2 to compute the inter-task interference delay $TaskDelay[i][]$
       suffered by each HRT and the total inter-task interference delay $Delay[i]$
       suffered by the HRTs allocated to $c_i$;
6:     $TotalDelay = \sum_{\forall c_i \in C_{hrt}} Delay[i]$ ;
7:     if $TotalDelay < MinTotalDelay$ then
8:         $MinTaskDelay[i][] = TaskDelay[i][]$, $MinTotalDelay =$
           $TotalDelay$;
9:         $MinDelayMap[][] = BtoCMapping[][]$;
10:    end if
11:    return
12: end if
13: for $i = 1; i \le N_{core}; i++$ do
14:    if $!used[i]$ then
15:        $c\_seq[n] = c_i$; $used[i]$=TRUE;
16:        $FindMinMapping(n+1)$; $used[i]$=FALSE;
17:    end if
18: end for
19: end function
20: Call $FindMinMapping(1)$;
21: return $MinTaskDelay[][], MinTotalDelay, MinDelayMap[][]$;
```

Algorithm 3 is to optimize bank-to-core mapping to minimize the inter-task interference delay suffered by HRTs. A function $FindMinMapping(n)$ is defined in line 2$\sim$19. $c\_seq[]$ stores a core queue. The total interference delay is computed in line 6. In line 7$\sim$10, the minimum interference delay and the corresponding bank-to-core mapping are updated. The backtracking and recursion of the algorithm is practiced in line 13$\sim$18.

## VI. WCET Estimation

The worst case execution time of $HRT_i(\in T_{hrt})$ comprises three parts: (1) the inter-task interference delay, which can be expressed as $\sum_{\forall q_i \in QHRT_i}(dbus_j + dbank_j)$. This part is estimated by the optimized result of Algorithm 3, (2) the waiting time in the ICBA, which can be expressed as $\sum_{\forall q_i \in QHRT_i} wicba_j$, where $wicba_j$ is the waiting time of $q_j$ in the ICBA, and (3) the rest part which does not be disturbed by other HRTs and can be measured by WCET analysis tools designed for single-core systems, such as the total execution time in the pipeline, the total latency of the shared L2 cache and bus, the total penalty of L2 cache miss (Note that we have assumed that each L2 cache miss fixedly needs 30 cycles to access the memory), and so on.

Let $q_j(\in QHRT_i)$ be the request in the current schedule of the XCBA, $q_{(j-1)}(\in QHRT_i)$ be the request in the previous schedule of the XCBA. When no NHRT, the waiting time of $q_j$ in the ICBA can be expressed as:

$$wicba_j = \begin{cases} Txcba_{sta} - tarr_j, & tarr_j > tacc_{(j-1)} \\ Txcba_{sta} - tacc_{(j-1)}, & \text{otherwise} \end{cases} \quad (2)$$

where $Txcba_{sta}$ is the start time of the current schedule. When existing NHRTs, Equation (2) also holds.

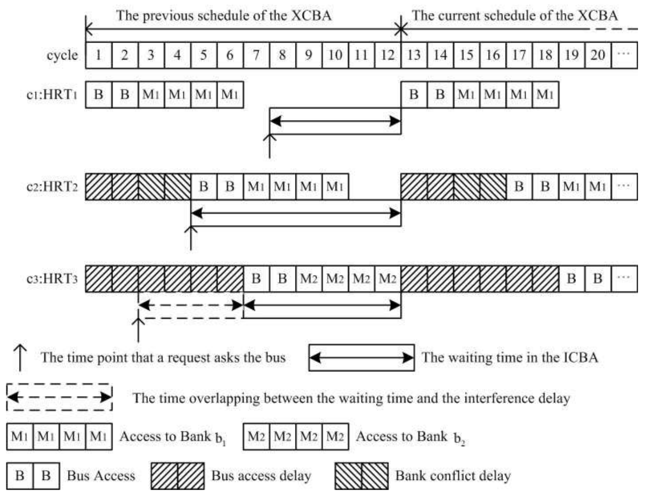

The previous schedule of the XCBA
The current schedule of the XCBA
cycle
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ···
c1:HRT1
c2:HRT2
c3:HRT3
B
M1
M2
The time point that a request asks the bus
The waiting time in the ICBA
The time overlapping between the waiting time and the interference delay
Access to Bank b1
Access to Bank b2
Bus Access
Bus access delay
Bank conflict delay

Fig. 2. An example of the waiting time in ICBAs when $c_{nhrt} = \phi$

An example of the waiting time in ICBAs when $c_{nhrt} = \phi$ is shown in Fig. 2. The start time of the current schedule of the XCBA is at the 13th cycle. In the current schedule of the XCBA, the request from $HRT_1(c_1)$ asks the bus at the 8th cycle which is later than the 1st cycle which is the time that the previous request from the same HRT accesses the bus. The waiting time of the request are 5 cycles. For the request from $HRT_3(c_3)$ in the current schedule of the XCBA, it asks the bus at the 3rd cycle which is earlier than the 7th cycle which is the time that the previous request from the same HRT accesses the bus. The valid waiting time of the request are 6 cycles because the time overlapping exists between its waiting time and the interference delay suffered by the previous request from the same HRT.

## VII. Experimental Evaluation

In this section, we evaluate our approaches by Mälardalen wcet benchmarks [24], which is maintained by the Mälardalen WCET research group situated at Mälardalen Real-Time Research Center (MRTC, http://www.es.mdh.se/).

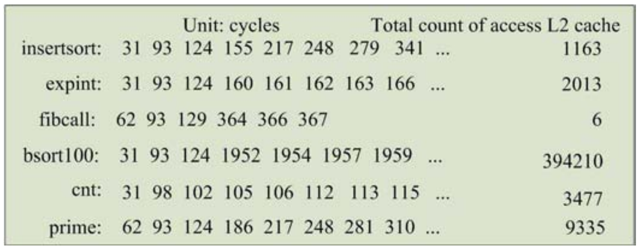
### A. Experimental Environment and Criterion Measurement

The embedded multicore architecture consists of 6 homogeneous cores, $c_1, c_2, \cdots, c_6$, each of which implements an in-order 5-stages pipeline without branch prediction. The instruction fetch queue size is 4, fetch width is 2 and the instruction window size is 8. Each core has 64B private instruction and data L1 cache (1-bank, 2-way associated, 8-byte per line, 1 cycle access and LRU replacement policy). The L2 cache is shared among all cores and it is 4KB (4 banks, 4-way associated, 32-byte per line, 4 cycles' access, LRU replacement policy). Each bank is 1KB and comprises 8 columns, each of which is 128B. The real-time bus, connecting cores and L2 cache, applies the IABA bus arbiter [14]. Bus access latency is 2 cycles. The penalty of L2 cache miss is 30 cycles.

To get the demanded L2 caching of all tasks, we use Chronos to measure their WCET when the L2 cache capacity is 128B, 256B, 512B, 1KB, 2KB and 4KB, respectively.

TABLE I. THE TYPES, DEMANDED COLUMNS AND ALLOCATION ON CORES

| Benchmark | Type | Demanded columns | Core |
|---|---|---|---|
| insertsort | HRT | 4 | $c_1$ |
| expint | HRT | 2 | $c_2$ |
| fibcall | HRT | 1 | $c_3$ |
| bsort100 | HRT | 16 | $c_4$ |
| cnt | NHRT | 8 | $c_5$ |
| prime | HRT | 1 | $c_6$ |



| | Unit: cycles | Total count of access L2 cache |
|---|---|---|
| insertsort: | 31 93 124 155 217 248 279 341 ... | 1163 |
| expint: | 31 93 124 160 161 162 163 166 ... | 2013 |
| fibcall: | 62 93 129 364 366 367 | 6 |
| bsort100: | 31 93 124 1952 1954 1957 1959 ... | 394210 |
| cnt: | 31 98 102 105 106 112 113 115 ... | 3477 |
| prime: | 62 93 124 186 217 248 281 310 ... | 9335 |

Fig. 3. The request timing sequences of the tasks

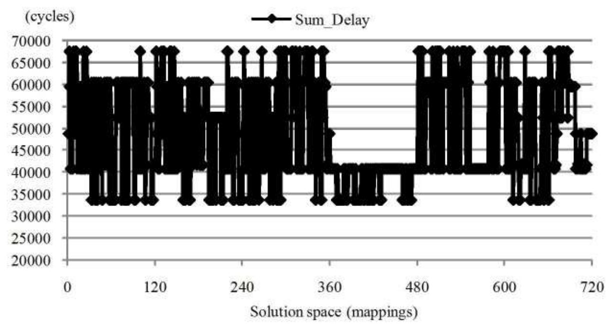


Fig. 4. The execution results of Algorithm 3 when existing NHRTs

According to the measured results, the demanded columns of each task are listed in TABLE I. The types of these tasks and the allocation on cores are listed in TABLE I as well. The bus request time sequences of these tasks measured by Chronos and lp_solve are shown in Fig. 3.

## B. Results

The measured results of Algorithm 3 are shown in Fig. 4. The solution space is 720, and the sum of the inter-task interference delay suffered by 5 HRTs is between 33774 and 67542 cycles. One of bank-to-core mappings with the minimum inter-task interference delay is shown in TABLE II. The waiting time and the inter-task interference delay suffered by 5 HRTs in this mapping are listed in TABLE III.

In order to demonstrate the effect of optimized bank-to-core mapping on WCET, we select one mapping in the solution space of Algorithm 3 as the un-optimized mapping which shown in TABLE IV. The corresponding waiting time and inter-task interference delay suffered by each task in this mapping are listed in TABLE V. We estimate the WCETs of all HRTs in these two mappings (shown in TABLE II and TABLE IV) respectively via computing the inter-task interference delay and waiting time in ICBAs. The results are shown in Fig. 5. All

TABLE II. A BANK-TO-CORE MAPPING THAT THE SUM OF THE INTER-TASK INTERFERENCE DELAY IS THE MINIMUM

| Benchmark | Core | Demanded columns | $b_1$ | $b_2$ | $b_3$ | $b_4$ |
|---|---|---|---|---|---|---|
| insertsort | $c_1$ | 4 | 4 | 0 | 0 | 0 |
| expint | $c_2$ | 2 | 0 | 2 | 0 | 0 |
| fibcall | $c_3$ | 1 | 1 | 0 | 0 | 0 |
| bsort100 | $c_4$ | 16 | 0 | 0 | 8 | 8 |
| cnt | $c_5$ | 8 | 3 | 5 | 0 | 0 |
| prime | $c_6$ | 1 | 0 | 1 | 0 | 0 |

TABLE III. THE WAITING TIME AND INTERFERENCE DELAY SUFFERED BY EACH HRT IN THE MAPPING SHOWN IN TABLE II

| Benchmark | Interference delay (cycles) | Waiting time (cycles) |
|---|---|---|
| insertsort | 5 | 11565 |
| expint | 2349 | 16039 |
| fibcall | 8 | 31 |
| bsort100 | 6387 | 1595516 |
| prime | 25025 | 37336 |

TABLE IV. AN UN-OPTIMIZED BANK-TO-CORE MAPPING

| Benchmark | Core | Demanded columns | $b_1$ | $b_2$ | $b_3$ | $b_4$ |
|---|---|---|---|---|---|---|
| insertsort | $c_1$ | 4 | 4 | 0 | 0 | 0 |
| expint | $c_2$ | 2 | 2 | 0 | 0 | 0 |
| fibcall | $c_3$ | 1 | 1 | 0 | 0 | 0 |
| bsort100 | $c_4$ | 16 | 1 | 8 | 7 | 0 |
| cnt | $c_5$ | 8 | 0 | 0 | 0 | 8 |
| prime | $c_6$ | 1 | 0 | 0 | 1 | 0 |

results are relative to the estimated WCETs of the tasks without inter-task interference. "Opt_Delay" and "UnOpt_Delay" denote the measured results in the optimized mapping and the un-optimized mapping, respectively. Compared with the measured results in the un-optimized mapping, the measured results in the optimized mapping have different degrees of improvement, improved by 13% in average.

The estimated WCET of $insertsort$ has the most improvement, about 23%. According to the arbitration of XCBA, the requests from HRTs have priority over requests from NHRTs. The interference delay suffered by $insertsort$ is only caused by the NHRT (i.e., $cnt$). In the mapping shown in TABLE II, $insertsort$ shares Bank $b_1$ with $cnt$, and $insertsort$ can suffer from bus access interference and bank access conflict caused by $cnt$. In the mapping shown in TABLE IV, however, $insertsort$ shares no bank with $cnt$, and $insertsort$ can only suffer from bus access interference caused by $cnt$. Thus, $insertsort$ suffers more interference delay in the bank-to-core mapping shown in TABLE II. On the other side, the chance that HRTs share the same banks is reduced in the mapping shown in TABLE II. This means that inter-task interference between HRTs is reduced and the length of a XCBA's schedule can also be reduced. For this reason, the waiting time of $insertsort$ in the mapping shown in TABLE II is less than that in the mapping shown in TABLE IV.

Although $bsort100$ suffers from less interference delay in the mapping shown in TABLE II, the improvement for $bsort100$ is not obvious (about 1%) for its huge program scale . For $fibcall$, the improvement also is about 1% for the less L2 cache access and the small program scale.

TABLE V. THE WAITING TIME AND INTERFERENCE DELAY SUFFERED BY EACH HRT IN THE MAPPING SHOWN IN TABLE IV

| Benchmark | Interference delay (cycles) | Waiting time (cycles) |
|---|---|---|
| insertsort | 1 | 18463 |
| expint | 4701 | 24043 |
| fibcall | 24 | 43 |
| bsort100 | 12765 | 1614178 |
| prime | 50051 | 37344 |

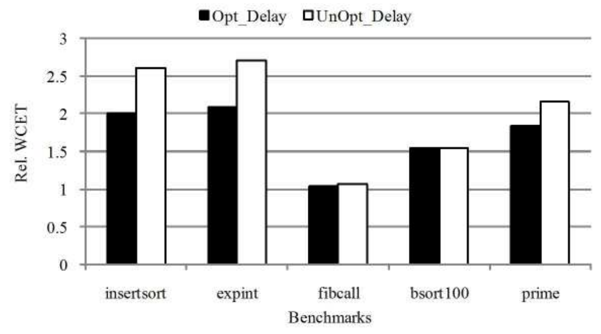


Fig. 5. The estimated results via computing interference delay and waiting time

## VIII. CONCLUSION

The WCETs estimated by bounding the upper bound of interference delay is seriously overestimated which can produce negative effect on the schedulability of HRTs, performance and energy dissipation of hard real-time multicore systems. In this paper, we reduce the inter-task interference delay by optimizing bank-to-core mapping on the multicore systems with IABA and the two-level partitioned cache.

We analyze and compute inter-task interference delay, and then put forward a core-queue optimization method of bank-to-core mapping and design the optimizing algorithms with the minimum inter-task interference delay. In this method, we use the request timing sequence in the worst case execution path to compute inter-task interference delay.

We use the Mälardalen WCET benchmarks to evaluate our proposal, and experimental results demonstrate that our approaches can reduce the inter-task interference delay and obtain tighter WCET estimations.

## ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No.61370062. Special thanks to the anonymous reviewers for their helpful comments and suggestions.

## REFERENCES

[1] ARM11 MPCore Processor, http://www.arm.com/products/processors/classic/arm11/arm11-mpcore.php.

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

[3] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza and et al., Predictability considerations in the design of multi-core embedded systems, in Proc. of Embedded Real Time Software and Systems, pp. 36-42, 2010.

[4] V.Suhendra and T. Mitra, Exploring locking & partitioning for predictable shared caches on multi-cores, in Proc. of the 45th annual Design Automation Conference, 2008, pp. 300-303.

[5] N. Guan, M. Stigge, W. Yi, and G.Yu, Cache-aware scheduling and analysis for multicores, in Proc. of the 7th ACM Int'l Conference on Embedded Software, 2009, pp. 245-254.

[6] H. Ding, Y. Liang, and T. Mitra, WCET-centric dynamic instruction cache locking, in Proc. of the conference on Design, Automation & Test in Europe, 2014, pp. 1-6.

[7] T. Liu, M. Li, and C. J. Xue, Instruction cache locking for multi-task real-time embedded systems, Real-Time Syst, vol. 48, no. 2, pp. 166-197, 2012.

[8] 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 the 28th IEEE Real-Time Systems Symposium, 2007, pp. 49-60.

[9] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, Modeling shared cache and bus in multi-cores for timing analysis, in Proc. of the 13th International Workshop on Software & Compilers for Embedded Systems, 2010, pp. 1-10.

[10] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, Bus-aware multicore WCET analysis through TDMA offset bounds, in Proc. of the 2011 Euromicro Conference on Real-Time Systems, 2011, pp. 3-12.

[11] S. Chattopadhyay, L. K. Chong, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk, A unified WCET analysis framework for multi-core platforms, ACM Trans. Embedd. Comput. Syst., vol. 13, no. 4s, pp. 1-29, 2014.

[12] P. Axer, R. Ernst, H. Falk, A. Girault, D. Grund, and et al., Building timing predictable embedded systems, ACM Trans. Embed. Comput. Syst., vol. 13, no. 4, pp.1-37, 2014.

[13] T. Ungerer, F. J. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, et al., Merasa: Multicore execution of hard real-time applications supporting analyzability, Micro, IEEE, vol. 30, no. 5, pp.66-75, 2010.

[14] 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 the 36th IEEE/ACM International Symposium on Computer Architecture, 2009, pp. 57-68.

[15] M. K. Yoon, J.E. Kim, and L. Sha, Optimizing tunable WCET with shared resource allocation and arbitration in hard real-time multicore systems, in Proc. of the 32th IEEE Real-Time Systems Symposium, 2011, pp. 227-238.

[16] J. Yan and W. Zhang, WCET analysis for multi-core processors with shared L2 instruction caches, in Proc. of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium,2008, pp. 80-89.

[17] F. Chen, D. Zhang and Z. Wang, Static analysis of run-time inter-thread interferences in shared cache multi-core architectures based on instruction fetching timing, in Proc. of 2011 IEEE International Conference on Computer Science and Automation Engineering, 2011, pp. 208-212.

[18] R. I. Davis and A. Burns, A survey of hard real-time scheduling for multiprocessor systems, ACM Computing Surveys, vol. 43, no. 4, pp. 1-44, 2011.

[19] 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 the 32th IEEE Real-Time Systems Symposium, 2009, pp. 57-67.

[20] H. F. Sheikh, H. Tan, I. Ahmad, S. Ranka, and P. Bv, Energy- and performance-aware scheduling of tasks on parallel and distributed systems, J. Emerg. Technol. Comput. Syst., vol. 8, no. 4, pp. 1-37, 2012.

[21] L. W. Yeon, Energy-Efficient Scheduling of Periodic Real-Time Tasks on Lightly Loaded Multicore Processors, IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 3, pp. 530-537, 2012.

[22] X. Li, Y. Liang, T. Mitra, and A. Roychoudhury, Chronos: A timing analyzer for embedded software, Science of Computer Programming, vol. 69, no.13, pp. 56-67, 2007.

[23] M. Berkelaar, K. Eikland, and P. Notebaert, lp solve version 5.5, http://lpsolve.sourceforge.net/5.5/, 2012.

[24] J. Gustafsson, A. Betts, A. Ermedahl, B. Lisper, The Mälardalen WCET benchmarks: past, present and future, in Proc. of the 10th International Workshop on Worst-Case Execution Time Analysis, 2010, pp.137-147.