

Minimizing Bank Conflict Delay for Real-Time Embedded Multicore Systems via Bank Mapping

Zhihua Gan^{1,2(✉)}, Mingquan Zhang¹, Zhimin Gu¹, Jizan Zhang¹, and Hai Tan¹

¹ School of Computer Science and Technology, Beijing Institute of Technology,
Beijing, China
800521@bit.edu.cn

² School of Software, Henan University, Kaifeng, China

Abstract. Multi-core architectures may meet the increasing performance requirement of real-time systems. However, it is harder to compute the WCET estimation in multi-core platforms due to inter-task interference that tasks suffer when accessing shared hardware resources. In this paper, we propose a finer grained approach to analyze the inter-task interference for multi-core platforms with the TDMA policy and bank-column cache partitioning, and our approach can reasonably estimate inter-task interference delays. Moreover, we make bank-to-core mapping to optimize the interference delays, and develop an algorithm for finding the best bank-to-core mapping. The experimental results show that our interference analysis approach can improve the tightness of interference delays by 14.68% on average compared to Upper Bound Delay (UBD) approach, and the optimized bank-to-core mapping can achieve the WCET improvement by 9.27% on average.

Keywords: Bank conflict · Multicore system · Worst case execution time

1 Introduction

The increasing demand for new functionality in real-time embedded system is driving an increment in the performance requirement of embedded processors. Multi-core processors are believed to be one of major solutions for real-time embedded systems [1–3], since they can provide high performance with low cost and relatively simple design [17]. However, applying multi-cores for real-time system is difficult due to inter-task interference accessing hardware shared resources [4]. These interferences complicate the behavior of a system, resulting in difficulties for performing a tight worst case execution time(WCET) analysis for a given task.

Previous solutions [5–9] have been focusing on the effect of inter-task interference caused by on-chip shared resources. For example Chattopadhyay et al. [5] 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. In [8], the authors improved the treatment of loop structures. Kelter et al. [9] improved analysis efficiency via bounding the upper bound of TDMA offsets. However, these researches mainly concentrated on the bus access interference and cache storage interference, and ignored the effect of bank conflict on the WCET estimation. In multi-core architecture, the shared cache usually consists of multiple banks that can be accessed in parallel, i.e., different cache requests can access different banks simultaneously. A bank can only handle one cache request at a time, when two or more cache requests try to access the same bank at the same time, the bank conflict takes place. The bank conflict brings extra execution time for the task, and its influence on WCET estimation has to be taken into account for ensuring safety of WCET. To the best of our knowledge, only Paolieri et al's work [12] and Yoon et al's work [13] considered bank conflict for WCET estimation. But they all employed the Upper Bound Delay(UBD) method to estimate the interference delay, in which a potential maximum delay (i.e., upper bound delay) that each cache request suffers is bounded, then, this delay is added to each request during WCET analysis. However, not all requests can suffer from bank conflict, even though bank conflicts occur among a group of requests, the delay of bank conflict suffered by each request is different. Therefore, this method causes pessimistic WCET.

The goal of this paper is to minimize the bank conflict delay and obtain the tighter WCET estimation for embedded multicore systems with TDMA policy. We make the following major contributions. (1) We propose a finer-grained approach to analyze bank conflict and bus access interference based on request timing, which can improve the tightness of interference delays. (2) We make bank mapping to optimize conflict delays, and develop an algorithm for finding the best bank mapping according to the queue of cores, such that the effect of inter-core bank conflict on the WCET estimation is minimized.

The rest of the paper is organized as follows. Section 2 describes the system model. The bank conflict analysis is described in Sect. 3. Section 4 presents the optimization algorithm of bank mapping, and experimental results are provided in Sect. 5. The conclusion is presented in Sect. 6.

2 System Model

2.1 Embedded Multi-core Architecture

We consider an embedded multi-core architecture consisting of N_{core} homogeneous core, $\mathbb{C} = \{C_1, C_2, \dots, C_{(N_{core})}\}$. Each core has its own private L1 instruction cache and L1 data cache. All the cores share an L2 combined cache \mathbb{B} which is partitioned into N_{bank} banks $\{B_1, B_2, \dots, B_{(N_{bank})}\}$, and each bank is subdivided into N_{column} columns. Bank access latency is L_M cycles (same for read/write operations for all banks). The real-time shared bus connecting cores and the shared L2 cache adopts TDMA policy and full-duplex as assumed by [12]. Every bus round has L_{round} equal time slots, $\mathbb{R} = \{S_1, S_2, \dots, S_{(L_{round})}\}$.

The length of one slot is L_B cycles, which is equal to the time of the bus completing one request. The core-to-slot mapping is one-to-one mapping, i.e., the $C_i(\in \mathbb{C})$ is mapped to $S_i(\in \mathbb{R})$. In addition, the penalty of L2 cache miss is $L2_{penalty}$ cycles.

2.2 Task Model

A hard real-time set comprises multiple independent hard real-time tasks(HRTs). All HRTs are partitioned to N_{core} cores in advance. The tasks allocated to the same core are executed sequentially and task migration is not allowed. In multicore systems, multiple tasks can be executed simultaneously in different cores. Let Γ_{sim} be the set of HRTs mapped to core $C_i(\in \mathbb{C})$, $HT_i = \{HRT_1, HRT_2, \dots, HRT_{n_i}\}$, n_i be the number of the HRTs in HT_i , and the demanded L2 cache of $HRT_i(\in HT_i)$ be $size_{HRT_j}$ columns. Thus, the demanded L2 cache size of C_i is $size_{C_i} = \max(size_{HRT_j} | 1 \leq j \leq n_j)$ columns.

3 Bank Conflict Delay Analysis

In this section, we analyze the bank conflict delay suffered by a HRT. Assuming that $m(\leq N_{core})$ cores $\{C_1, C_2, \dots, C_m\}$ sharing one bank try to access the bank in the lth bus round, and they do not miss their own bus slots. The value of $C_i(1 \leq i \leq m)$ is the index of the corresponding bus slot. Let $bcd_{ij}, i \neq 1$, be the bank conflict delay suffered by HRT running on C_i in the lth bus round shown in Fig. 1, which can be expressed by formulation (2).

$$bcd_{ij} = \text{Max}\{bcd_{(i-1)j} + L_M - (C_i - C_{i-1}) \cdot L_B, 0\} \quad (1)$$

In formulation (1), the $bcd_{(i-1)j}$ denotes the bank conflict delay suffered by the HRT in $lth - 1$ bus round. If $i = 1$ and $l = 1$, $bcd_{11} = 0$, otherwise, the bcd_{1l} can be computed by formulation (2). In formulation (2), C_{pre} denotes the predecessor core of C_1 and bcd_{pre_k} be the bank conflict delay suffered by the HRT running on C_{pre} in the kth bus round, which are shown in Fig. 2

$$bcd_{ij} = \text{Max}\{bcd_{pre_k} + (j - k - 1) \cdot N_{core} \cdot L_B - (N_{core} - (N_{core} - C_1)) \cdot L_B, 0\} \quad (2)$$

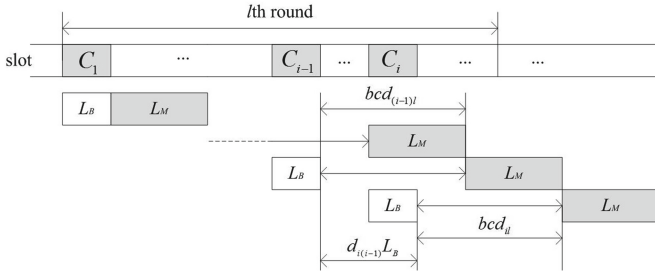


Fig. 1. The bank conflict delay suffered by HRT running on C_i in the lth bus round

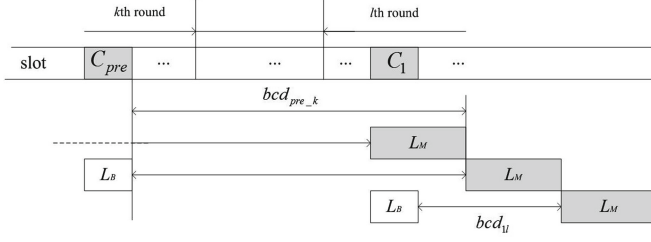


Fig. 2. The bank conflict delay suffered by C_1 in the l th bus round

The formulation (2) can be simplified as formulation (3):

$$bcd_{ij} = \text{Max}\{bcd_{pre_k} + L_M + (j - k) \cdot N_{core} \cdot L_B - (C_{pre} - C_1) \cdot L_B, 0\} \quad (3)$$

In formulation (3), the bank conflict delay suffered by C_1 is transferred from the previous bus round, which is the initial bank conflict delay of the shared bank in the current bus round. There are two factors to affect the initial bank conflict delay, i.e., the number of requests to access bank B_i in one bus round and the distribution of the corresponding slots. A bus round can contain at most $\left\lfloor \frac{L_{round} \cdot L_B}{L_M} \right\rfloor$ requests without any bank access conflict, for one L2 cache access needs be at least L_M cycles. Let $N_{c,b}^i$ be the number of cores sharing bank B_i . In the worst case, the total requests in one bus round to access one bank is $N_{c,b}^i$. If $N_{c,b}^i > \left\lfloor \frac{L_{round} \cdot L_B}{L_M} \right\rfloor$ and $N_{c,b}^i \cdot L_M > L_{round} \cdot L_B$, the total time of access L2 cache is greater than the length of a bus round, and the initial bank conflict delay in the following bus round is $N_{c,b}^i \cdot L_M - L_{round} \cdot L_B$ cycles.

In addition, the distribution of the corresponding slots affect the initial bank conflict delay in the following bus round even. The initial bank conflict delay cannot be propagated across bus rounds if $N_{c,b}^i \leq \left\lfloor \frac{L_{round} \cdot L_B}{L_M} \right\rfloor$. Otherwise, the initial bank conflict delay cannot be transmitted in bus rounds if more than $\left\lfloor \frac{L_{round} \cdot L_B}{L_M} \right\rfloor$ cores have requests to access L2 cache in several sequent bus rounds. In this paper, we define transferred bank conflict delay of one bank in one bus round as the initial bank conflict delay of the bank in the bus round if the initial bank conflict delay can be propagated across bus rounds.

Transferred bank conflict delay brings more negative impact to the predictability of the hard real-time multi-core system for it can be propagated across bus rounds. In order to minimize the bank conflict delay on one bank, the number of cores sharing one bank need be less than or equal to $\left\lfloor \frac{L_{round} \cdot L_B}{L_M} \right\rfloor$ to guarantee no transferred bank conflict delay. If not, the number of the cores shared one bank is minimum to decrease transferred bank conflict delay.

4 Optimizing Bank-to-core Mapping

In this section, we present bank-to-core mapping to reduce bank conflict, and design an algorithm for this optimization problem.

4.1 Optimization of Bank Conflict Delay

Let N_c^i be the total bus rounds demanded by the HRT running on C_i , and D_c^i be the total bank conflict delay suffered by the HRT, which can be expressed as $D_c^i = \sum_{l=1}^{N_c^i} bcd_{il}$. Let D_{jl} be the total bank conflict delay on bank B_j in the l th bus round, which can be expressed by $D_{jl} = \sum_{i=1}^{K_{jl}} bcd_{il}$, where K_{jl} is the total count of the requests to access bank B_j in the l th bus round. Let C_{bj} be the core set mapped to bank B_j , N_b^j be the maximum number of bus rounds of the HRTs in the core set C_{bj} , and D_b^j be the total bank conflict delay on B_j . The D_b^j can be expressed as follows:

$$D_b^j = \sum_{l=1}^{N_b^j} D_{jl} = \sum_{l=1}^{N_b^j} \sum_{i=1}^{K_{jl}} bcd_{il} = \sum_{i=1}^{N_{c \rightarrow b}^i} D_c^i = \sum_{i=1}^{N_{c \rightarrow b}^i} \sum_{l=1}^{N_c^i} bcd_{il} \quad (4)$$

As bank conflict delay suffered by one HRT is nonnegative, minimizing the total bank conflict delay for each HRT is equivalent to minimizing the total bank conflict delay on each bank, and that can be expressed by $\min(D_c^i | \forall C_i \in \mathbb{C}) \Leftrightarrow \min(D_b^j | \forall B_j \in \mathbb{B})$. The total bank conflict delay on one bank also is nonnegative for the bank conflict delay suffered by one HRT is nonnegative, therefore, we can derive the following formulation $\min(\sum_{j=1}^{N_{bank}} D_b^j) \Leftrightarrow \min(\sum_{j=1}^{N_{bank}} \sum_{l=1}^{N_b^j} \sum_{i=1}^{K_{jl}} bcd_{il})$. Let x_{ij} be the mapping from $C_i (\in \mathbb{C})$ to $B_j (\in \mathbb{B})$, and n_{ij} be the column number of C_i mapped to B_j . If $C_i \in \mathbb{C}$, x_{ij} is equal to 1, otherwise, x_{ij} is 0. As the demanded cache of HRTs are exclusively mapped to columns, n_{ij} is integer and $0 \leq n_{ij} \leq \min\{N_{column}, Size_{C_i}\}$. If $x_{ij} = 1$, then $n_{ij} \geq 0$. Otherwise, $n_{ij} = 0$. x_{ij} and n_{ij} are the decision variables. The optimization model to minimize the bank conflict delay suffered by each HRT can be described as follows:

Objective function:

$$\min(\sum_{j=1}^{N_{bank}} \sum_{l=1}^{N_b^j} \sum_{i=1}^{K_{jl}} bcd_{il}) \quad (5)$$

Constraints:

$$N_{bank} \cdot N_{column} \geq \sum_{i=1}^{N_{core}} size_{C_i} \quad (6)$$

$$\forall C_i \in \mathbb{C} \quad size_{C_i} = \sum_{j=1}^{N_{bank}} n_{ij} \cdot x_{ij} \quad (7)$$

$$\forall B_i \in \mathbb{B} \quad N_{column} \geq \sum_{i=1}^{N_{core}} n_{ij} \cdot x_{ij} \quad (8)$$

In this optimization problem, the objective function describes that the overall bank conflict delay is minimum. In bank-column cache partitioning, a HRT exclusively accesses its allocated columns, therefore, the size of shared L2 cache need meet the total demand of N_{core} cores, which is expressed as constraint (6). Constraint (7) describes that the model need meet the demanded caching of each core. Constraint (8) is the size constraint of one bank.

4.2 Design of the Proposed Optimization Algorithm

Based on above conflict analysis, the optimization problem is transformed to find the optimal bank mapping, and the bank conflict delay suffered by the cores sharing the bank is minimum. In this subsection, we design algorithm for this optimization problem without any specific initial bank-to-core mapping. Algorithm 1 shows our algorithm for bank mapping optimization. It iteratively calls itself, until the column requirement of all tasks is satisfied. Algorithm 1 takes the number of cores and the column requirement of each tasks as an input. Line 1 initializes the initial minimal total conflict delays to infinity, and initializes decision variables used to *false* for performing recursion call. A recursive function *FindOptimalMapping*() is defined to search the optimal bank mapping in the solution space in lines 2~31. Lines 5~15 generate the mapping of bank to core *BtoCmapping*[] based on the core queue *c_seq*{}. Then, based on bank mapping *BtoCmapping*[], we calculate the conflict delays of each HRT in line 16. In line 17, we compute the conflict delays suffered by all HRTs. In lines 18~22, the best bank mapping is saved. The recursion of the algorithm is practiced in lines 23~30. In line 27, a core queue is generated in the recursive walk, and the generated core queue is stored in the array *c_seq*{}.

5 Evaluation

In this section, we implement above approaches, and set up experiments to demonstrate the effectiveness of our optimization.

5.1 Experimental Setup

In our experiment, the multi-core architecture consists of 6 cores, 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, 8-byte per line, 1 cycle access and LRU replacement policy). The L2 cache is shared among all cores, the total size is 4KB, 4 banks (each of which is 1KB), 4-way, 32-byte per line, 4 cycles access (i.e., cycles), and LRU replacement policy. Each bank is partitioned into 8 columns and the size of each one is 128B. Bus access

Algorithm 1. Optimizing bank-to-core mapping

Require: N_{core} , $size_{C_i}$ ($C_i \in \mathbb{C}$)
Ensure: the total conflict delays ($MinDelay$), the bank mapping ($OptimalMap[][]$)

```

1:  $MinDelay = \text{Infinity}$ ,  $used[j] = \text{false}$  ( $1 \leq j \leq N_{core}$ );
2: function  $FindOptimalMapping(N)$ 
3:   if  $N > N_{core}$  then
4:      $ncol = N_{column}$ ;  $nbank = 1$ ;
5:     for each core  $C_i$  in  $c\_seq[]$  do
6:       if  $size_{C_i} \geq ncol$  then
7:         while  $size_{C_i} \geq ncol$  do
8:            $BtoCmapping[i][nbank] = ncol$ ;
9:            $size_{C_i} = size_{C_i} - ncol$ ;  $nbank++$ ;  $ncol = N_{column}$ 
10:        end while
11:         $BtoCmapping[i][nbank] = size_{C_i}$ ;  $ncol = ncol - size_{C_i}$ ;
12:      else
13:         $BoCmapping[i][nbank] = size_{C_i}$ ;  $ncol = ncol - size_{C_i}$ ;
14:      end if
15:    end for
16:    Computing  $conflict\_delay[]$  suffered by each  $HRT_i$  based on formulation (3) ;
17:     $Total\_delay = \sum_{\forall C_i \in \mathbb{C}} Interference\_Delay[i]$ ;
18:    if  $MinDelay > Total\_delay$  then
19:       $MinEnergy = Total\_delay$ ;
20:       $OptimalMap[][] = BtoCmapping[][]$ ;
21:    end if
22:  end if
23:  for  $j = 1$ ;  $j \leq N_{core}$ ;  $j++$  do
24:    if  $!used[j]$  then
25:       $c\_seq[N] = C_j$ ;
26:       $used[j] = \text{true}$ ;
27:       $FindMinMapping(N + 1)$ ;
28:       $used[j] = \text{false}$ ;
29:    end if
30:  end for
31: end function

```

latency is 2 cycles (i.e., $L_M = 4$ cycles). All benchmarks used in this section are part of Mälardalen wcet benchmarks [14] as shown in Table 1 including the byte size Bytes and the lines of code LOC.

In order to get the L2 cache size of all benchmarks demanded, we use Chronos [15] to measure their WCET that the L2 cache size is 128 B, 256 B, 512 B, 1 KB, 2 KB and 4 KB respectively. The configurations of measurement are: the instruction and data L1 cache are 64 B (1-bank, 2-way, 8-byte per line, LRU replacement policy), respectively. L2 cache is 4-way, 32-byte per line and LRU replacement policy. According to these results, we adopt the L2 cache size provided in Table 1 (column 3). In addition, columns 6–9 in Table 1 present the initial bank-to-core mapping.

5.2 Experimental Results

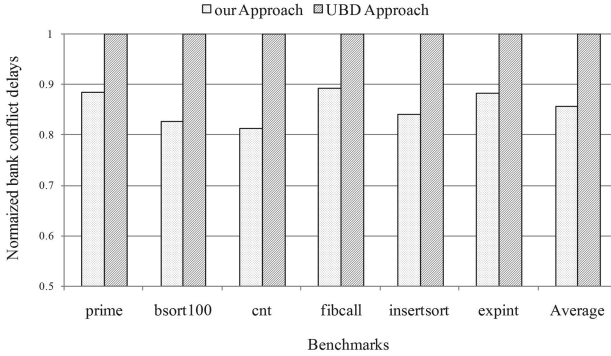
5.2.1 Our Conflict Analysis Approach Versus UBD Approach

In this experiment, our focus is the effectiveness of conflict analysis approach. Therefore, we assume that the bank mapping is used for HRTs in Table 1 (i.e., the order of core in queue $c_seq[]$ is $\{C_1, C_2, C_3, C_4, C_5, C_6\}$). We compare our approach with UBD approach where the upper bound delay can be expressed as $UBD = (N_{core} - 1) \cdot \text{Max}(L_B, L_M)$. Figure 3 illustrates this comparison

Table 1. The benchmarks

Benchmark	CodeSize (bytes)	Columns	LOC	Core	B_1	B_2	B_3	B_4
prime	797	1	47	C_1	1	0	0	0
bsort100	2779	16	128	C_2	7	8	0	0
cnt	2880	8	267	C_3	0	0	8	0
fibcall	3499	1	72	C_4	0	0	0	1
insertsort	3892	4	92	C_5	0	0	0	4
expint	4288	2	157	C_6	0	0	0	2

in bank conflict delays for all benchmarks in Table 1. The bank conflict delay values are normalized to the delays obtained by UBD approach. We can see that the bank conflict delays obtained by our approach is less than the delays obtained by UBD approach for HRTs, which is because our approach takes request timing into consideration on runtime inter-task conflict on shared cache, and can reasonably estimate the bank conflict delay. The proposed approach can improve the tightness of bank conflict delays by 14.68% on average compared to the UBD approach.

**Fig. 3.** Comparison of bank conflict delays of different HRT for two approaches

5.2.2 Impact of Bank-to-core Mapping on WCET

The sum of bank conflict delays suffered by all tasks in task set under different bank-to-core mapping are shown in Fig. 4. We can observe that the solution space of bank mapping is 720, and the total bank conflict delays suffered by all tasks have significant difference under different bank mapping. The bank conflict delays are 29836 cycles under the worst mapping. In contrast, the bank conflict delays suffered by all tasks are only 20242 cycles under the best mapping. In order to compare the effect of bank mapping on WCET. We use the mapping in Table 1 as the non-optimized mapping. One of the mappings with the minimum

conflict delays is shown in Table 2. In this bank-to-core mapping, the WCET for all tasks under the optimized bank-to-core mapping improved by 9.27% on average.

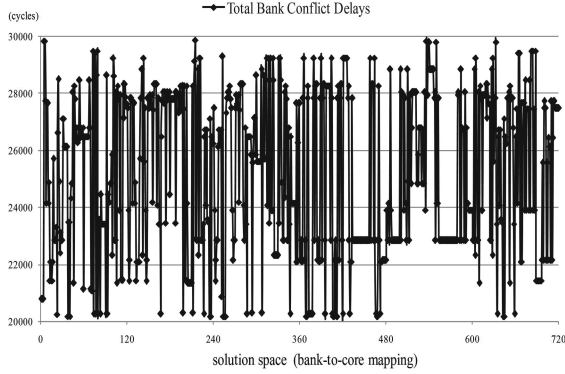


Fig. 4. The total bank conflict delays suffered by all HRTs

Table 2. The optimal bank-to-core mapping

Benchmark	Core	B_1	B_2	B_3	B_4
prime	C_1	0	1	0	0
bsort100	C_2	0	7	8	1
cnt	C_3	4	0	0	4
fibcall	C_4	0	1	0	0
insertsort	C_5	4	0	0	0
expint	C_6	0	0	0	2

6 Conclusion

In this paper, we presented a finer-grained approach to analyze the bank conflict. This approach can reasonably estimate conflict delays. Furthermore, we optimize the conflict delays through bank-to-core mapping and design the optimizing algorithms to find the optimal mapping. Using our analysis approach, the bank conflict delays can be improved by 14.68% on average compared to existing method. The experimental results show that bank mapping has great impact on WCET estimation, which can achieve a 9.27% improvement in WCET on average.

Acknowledgements. This work is supported by the National Natural Science Foundation of China(No.61370062,61462004). We thank the anonymous reviewers for their feedback.

References

1. ARC Advisory Group, Process Safety System Worldwide Outlook, Market Analysis and Forecast through (2015)
2. Qiu, M., Zhong, M., Li, J., Gai, K., Zong, Z.: Phase-change memory optimization for green cloud with genetic algorithm. *IEEE Trans. Comput.* **64**(12), 3528–3540 (2015)
3. Gai, K., Qiu, M., Zhao, H., Tao, L., Zong, Z.: Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *J. Netw. Comput. Appl.* **59**, 46–54 (2016)
4. Wilhelm, R., Engblom, J., Ermedahl, A., et al.: The worst-case execution-time problem - overview of methods and survey of tools. *ACM TECS* **7**(3), 1–53 (2008)
5. Chattopadhyay, S., Chong, L.K., Roychoudhury, A., Kelter, T., Marwedel, P., Falk, H.: A unified WCET analysis framework for multi-core platforms. *ACM Trans. Embed. Comput. Syst.* **13**(4), 1–29 (2014)
6. Yan, J., Zhang, W.: WCET analysis for multi-core processors with shared L2 instruction caches. In: *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 80–89 (2008)
7. Zhang, W., Yan, J.: Accurately estimating worst-case execution time for multi-core processors with shared direct-mapped instruction caches. In: *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 455–463 (2009)
8. Radojković, P., Girbal, S., Grasset, A., Quinones, E., Yehia, S., Cazorla, F.J.: On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments. *ACM Trans. Arch. Code Optim.* **8**(4), 1–25 (2012)
9. Kelter, T., Falk, H., Marwedel, P., Chattopadhyay, S., Roychoudhury, A.: Bus-aware multicore WCET analysis through TDMA offset bounds. In: *Proceedings of the 2011 Euromicro Conference on Real-Time Systems*, pp. 3–12 (2011)
10. Gai, K., Qiu, M., Tao, L., Zhu, Y.: Intrusion detection techniques for mobile cloud computing in heterogeneous 5G. *Secur. Commun. Netw.* **9**, 1–10 (2015)
11. Qiu, M., Gai, K., Thuraishingham, B., Tao, L., Zhao, H.: Proactive user-centric secure data scheme using attribute-based semantic access controls for mobile clouds in financial industry. *Future Gener. Comput. Syst.* (2016, in press)
12. Paolieri, M., Quiñones, E., Cazorla, F.J., Bernat, G., Valero, M.: Hardware support for WCET analysis of hard real-time multicore systems. In: *Proceedings of the 36th IEEE/ACM International Symposium on Computer Architecture*, pp. 57–68 (2009)
13. Yoon, M.K., Kim, J.E., Sha, L.: Optimizing tunable WCET with shared resource allocation and arbitration in hard real-time multicore systems. In: *Proceedings of the 32th IEEE Real-Time Systems Symposium*, pp. 227–238 (2011)
14. Gustafsson, J., et al.: The Malardalen WCET benchmarks-past, present and future. In: *WCET workshop*
15. Li, X., Liang, Y., Mitra, T., Roychoudhury, A.: Chronos: a timing analyzer for embedded software. *Sci. Comput. Program.* **69**(1), 56–67 (2007)
16. RapiTime: Worst-case execution time analysis. User Guide. Rapita Systems (2014)
17. Gai, K., Li, S.: Towards cloud computing: a literature review on cloud computing and its development trends. In: *The 4th International Conference on Multimedia Information Networking and Security*, pp. 142–146, Nanjing, China (2012)

Smart Computing and Communication

First International Conference, SmartCom 2016,

Shenzhen, China, December 17-19, 2016, Proceedings

Qiu, M. (Ed.)

2017, XIII, 589 p. 233 illus., Softcover

ISBN: 978-3-319-52014-8