JCSE Journal of Computing Science and Engineering

Regular Paper

Journal of Computing Science and Engineering,
Vol. 7, No. 1, March 2013, pp. 53-66

Bounding Worst-Case DRAM Performance on Multicore Processors

Yiqiang Ding, Lan Wu, and Wei Zhang*
Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA, USA
dingy4@vcu.edu, wul3@vcu.edu, wzhang4@vcu.edu

Abstract

Bounding the worst-case DRAM performance for a real-time application is a challenging problem that is critical for computing worst-case execution time (WCET), especially for multicore processors, where the DRAM memory is usually shared by all of the cores. Typically, DRAM commands from consecutive DRAM accesses can be pipelined on DRAM devices according to the spatial locality of the data fetched by them. By considering the effect of DRAM command pipelining, we propose a basic approach to bounding the worst-case DRAM performance. An enhanced approach is proposed to reduce the overestimation from the invalid DRAM access sequences by checking the timing order of the co-running applications on a dual-core processor. Compared with the conservative approach, which assumes that no DRAM command pipelining exists, our experimental results show that the basic approach can bound the WCET more tightly, by 15.73% on average. The experimental results also indicate that the enhanced approach can further improve the tightness of WCET by 4.23% on average as compared to the basic approach.

Categories: Embedded computing

Keywords: Performance; Reliability; Real-time scheduling; WCET; Multicore processor

I. INTRODUCTION

With the rapid development of computing technology and the diminishing return of complex uniprocessors, multicore processors are being used more widely in the computer industry. Future high-performance real-time systems are likely to benefit from multicore processors due to the significant boost in processing capability, low power consumption, and high density.

In real-time systems, especially hard real-time systems, it is crucial to accurately obtain the worst-case execution time (WCET) for real-time tasks to ensure the correctness of schedulability analysis. Although the WCET of a real-time application can be obtained by measurement-based approaches, the results are generally unreliable due to the impossibility of exhausting all the possible program paths. Alternatively, static WCET analysis [1] can be used to compute the WCET, which should be safe and as accurate as possible. The WCET of a real-time application is not only determined by its own attributes, but also affected by the timing of architectural components, such as pipelines, caches, and branch predictors. Most prior research works have focused on WCET analysis for single-threaded applications running on uniprocessors [2-6], but these methods cannot be easily applied to estimate the WCET on multicore processors with shared resources, such as a shared L2 cache and DRAM memory. This is because the possible interferences in the shared resources between different threads can significantly increase the complexity of WCET analysis.

Due to its structural simplicity, high density, and volatility, DRAM is usually utilized in current popular proces-

Open Access http://dx.doi.org/10.5626/JCSE.2013.7.1.53 http://jcse.kiise.org

Received 12 February 2013, **Accepted** 1 March 2013

*Corresponding Author

pISSN: 1976-4677 eISSN: 2093-8020

sors, including multicore processors. A DRAM system consists of multiple components, such as a memory access controller, command/address bus, data bus, and DRAM devices. The latency of an access to the DRAM varies due to the status of each component when accessed. One recent work studied the vulnerability of current multicore processors due to a new class of denial of service (DoS) attacks [7]. Under the current DRAM architecture, a thread with a particular memory access pattern can overwhelming the shared resources in the DRAM, preventing other threads from using these resources efficiently. Therefore, the latencies of the DRAM accesses from other threads could be prolonged.

There have been several studies to model and predict DRAM memory performance. Ahn et al. [8] performed a performance analysis of scientific and multimedia applications on DRAM memory with various parameters, and found that the most critical performance factors are high read-write turnaround penalties and internal DRAM bank conflicts. They then developed an accurate analytical model for the effective random-access bandwidth given DRAM technology parameters and the burst-length. Yuan and Aamodt [9] proposed a hybrid analytical model to predict DRAM access efficiency based on memory trace profiling. Bucher and Calahan [10] modeled the performance of an interleaved common memory of a multiprocessor using queuing and simulation methods. Choi et al. [11] presented an analytical model to predict the DRAM performance based on the DRAM timing and memory access pattern parameters. However, these prior studies have focused on predicting the average-case DRAM performance, rather than the worst-case. For example, the DRAM access patterns assumed in these studies were based on typical access patterns or derived from simulated traces, which cannot be safely used to represent the worst-case DRAM access patterns to derive the WCET.

Research was performed recently to bound the worst-case DRAM performance on a uniprocessor by considering the impact of the row-buffer management policy [12]. However it is more challenging to conduct WCET analysis on a multicore processor by bounding the worst-case DRAM performance for the following reasons. First, the DRAM access pattern of a thread depends on its accesses to higher-level cache memories, such as the L2 cache. If the DRAM memory is shared with different cores, the accesses of a thread can be greatly impacted by inter-core DRAM access interference. Second, the worse-case latency of a DRAM access of a thread is determined by not only the number of the simultaneous DRAM accesses from other threads, but also the timing order of all these DRAM accesses and the spatial locality of the data fetched by them. However, the timing order of simultaneous DRAM accesses from co-running threads is hard to determine through static analysis, because all the threads are running independently on different cores.

To overcome these difficulties, this paper first investigates the timing characteristics of DRAM accesses with a focus on DRAM devices. Our study shows that the DRAM commands from multiple consecutive DRAM accesses can be pipelined on DRAM devices, and the degree of the DRAM command pipelining varies according to the spatial locality of the data accessed, which may impact the worst-case latency of each access. A basic approach is then proposed to estimate the worst-case situation of DRAM command pipelining, which leads to the worst-case latency for a DRAM access among a sequence of consecutive DRAM accesses. An enhanced approach is proposed to reduce the overestimation from the invalid DRAM access sequences by checking the timing order constraints of concurrent applications. In addition, we utilize the extended integer linear programming (ILP) approach [4] to model the constraints between the accesses to the higher-level cache memory and the DRAM accesses. The worst-case DRAM performance is integrated into the objective function of the extended ILP approach to bound the WCET of a real-time task running on a multicore processor.

The rest of the paper is organized as the follows. First, the multicore architecture studied in this work is described in Section II. Next, the background of the DRAM system is introduced in Section III. Section IV presents the timing characteristics of DRAM accesses, with a focus on DRAM devices. Then, we introduce two approaches to bound the worst-case DRAM performance in Section V. Section VI introduces the evaluation methodology, and Section VII gives the experimental results. Finally, conclusions are presented in Section VIII.

II. SYSTEM ARCHITECTURE

Fig. 1 shows the system architecture of a multicore processor with N cores studied in this paper ($N > 1$). Each

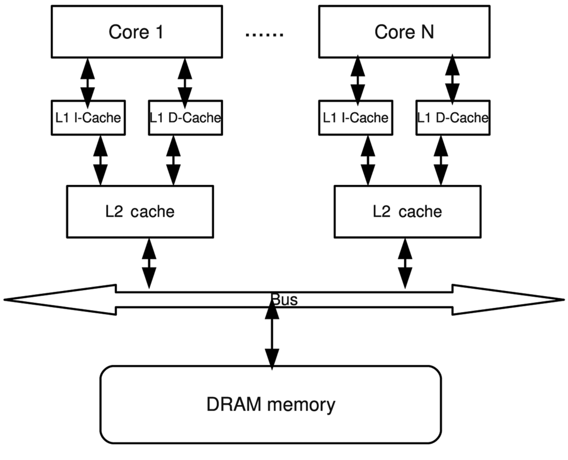


Fig. 1. Target system architecture.

core is symmetrical, with its own processing unit, pipeline, L1 instruction and data caches, and private L2 cache, which are not uncommon in commercial multicore designs. The DRAM is shared by all cores through a shared bus. In order to focus on bounding the worst-case DRAM performance, the interactions between the DRAM and the hard disk are ignored in our study. It is assumed that all the code and data of a thread are loaded into the DRAM beforehand, such that no page fault would occur during subsequent execution.

III. DRAM MEMORY SYSTEM

Generally, a DRAM memory system comprises three major components, as shown in Fig. 2. The DRAM devices store the actual data; the memory controller is responsible for the communication between the DRAM devices and the processor; and the buses connect the DRAM devices and the memory controller to transfer addresses, commands, and data.

DRAM device: Multiple levels of store entities are organized hierarchically in a DRAM device, such that DRAM accesses can be served in parallel on a certain level according to the spatial locality of the data being accessed. The memory array is the fundamental storage entity in a DRAM device. A bank is a set of independent memory arrays, and has a two-dimensional structure with multiple rows and columns. A bank also has a row buffer, and data can only be read from this buffer. A rank consists of a set of banks sharing the same I/O gating, and operates in lockstep to a given DRAM command. A channel is defined as a set of ranks that share the data bus. For example, multiple DRAM accesses to different ranks in the same channel can be executed in parallel, except when the data are transferred on the shared data bus.

Memory controller: The memory controller manages the flow of data in and out of DRAM devices connected to it. The row-buffer management policy, the addressing mapping scheme, and the memory transaction and DRAM command ordering scheme are three important design considerations and implementations for the memory controller.

There are two types of row-buffer management policies: the open-page policy and the close-page policy. The open-page policy is designed to favor memory accesses to the same row of memory by keeping the row buffer open and holding a row of data for ready access. In contrast, the close-page policy is designed to favor accesses to random locations in the DRAM, and optimally supports the DRAM access patterns with low degrees of spatial locality. In a multicore processor, the intermixing of DRAM access sequences from multiple threads reduces the spatial locality of the access sequence. The close-page policy can achieve better performance [13] without any optimization on the memory controller [14]. The DRAM access transactions and DRAM commands are queued in the memory controller. The queuing delay also affects the performance of DRAM. DRAM commands can be scheduled by various scheduling algorithms [15, 16] based on different factors, such as the availability of resources in DRAM devices. In our study, the memory controller is assumed to have no optimization, and the close-page policy and the first come first serve (FCFS) scheduling algorithm are used.

IV. TIMING OF ACCESSING DRAM MEMORY SYSTEMS

In this section, we study the timing characteristics of the DRAM access, both in the case of an individual DRAM access and multiple consecutive DRAM accesses. Also, the worst-case latency for a DRAM access among a sequence of consecutive DRAM accesses is derived.

Generally, the timing of a DRAM access consists of three parts: the latency through the bus between the processor and the memory controller, the queuing delay in the memory controller, and the latency of accessing the DRAM device. In this paper, as we focus on the estimation of the latency of accessing DRAM devices, the worst-case bus latency and the worst-case queuing delay in the memory controller are estimated safely as con-

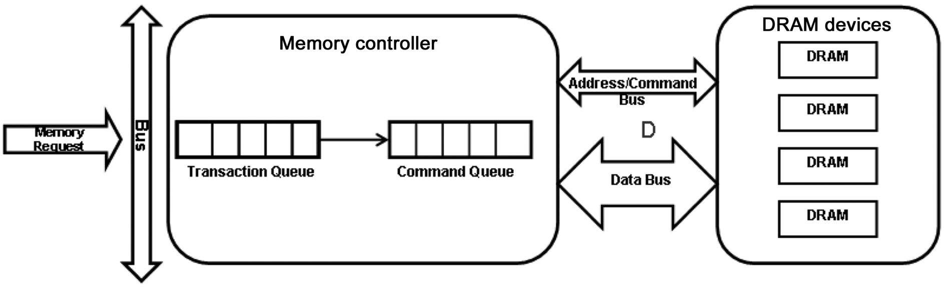


Fig. 2. DRAM architecture.

stants by a conservative approach, assuming that a given DRAM access from one core should wait for the bus transferring and the memory controller queuing of the other N-1 DRAM accesses issued from other cores simultaneously on a multicore processor with N cores.

A. Generic DRAM Access Protocol

Typically, a DRAM access can be translated into several DRAM commands to move data between the memory controller and the DRAM devices. A generic DRAM access protocol can be modeled by only considering some necessary basic DRAM commands and related timing constraints. It is assumed that two different commands can be fully pipelined on a DRAM device only if they do not have any conflict on a shared resource at a given time, which can be called DRAM command pipelining. The whole procedure for DRAM commands of a given DRAM access to fulfill the data movement are illustrated in Fig. 3. The figure also shows the resources required by these commands that cannot be shared by the commands from other DRAM accesses concurrently. In the first phase, the command is transported via the command and address buses and decoded by the DRAM device. In the second phase, the data are moved into a bank. The data are transported on the shared I/O gating circuit in the third phase. Finally, the data are transferred to the memory controller by the data bus.

In the generic DRAM access protocol, three generic DRAM commands are defined: row access commands, column access commands, and precharge commands. The timing parameters related to these commands are shown in Table 1. t_{RCD}, t_{CAS}, and t_{BURST} are all a part of t_{RAS}, as shown in Fig. 4. The DRAM refresh command is not covered in the generic DRAM protocol, because it is not issued from any DRAM access, and could interrupt the command pipeline periodically.

Table 1. Timing parameters defined in the generic DRAM access protocol

Parameter	Description
t_{BURST}	Data burst duration
t_{CMD}	Command transport duration
t_{CAS}	Column access strobe latency
t_{RAS}	Row access strobe latency
t_{RCD}	Row to column command delay
t_{RP}	Row precharge duration

B. Timing of an Individual DRAM Device Access

A typical cycle of an individual DRAM device access to read data consists of three major phases: row access, column access, and precharge. The details of the cycle are illustrated in Fig. 4. Including the time of command transferring, the latency for the whole cycle can usually be computed by using Equation (1), the timing parameters of which are illustrated in Fig. 4. As the data movement for a DRAM access is finished at the end of the column access, the latency of a read cycle without considering the precharge phase can be described by Equation (2), the timing parameters of which are also shown in Fig. 4.

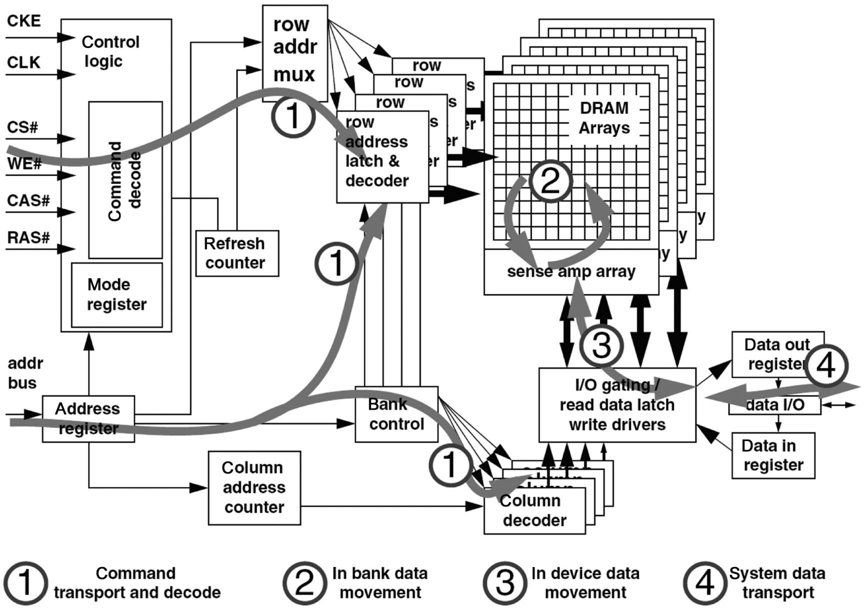


Fig. 3. Command and data movement for an individual DRAM device access on a generic DRAM device [13].

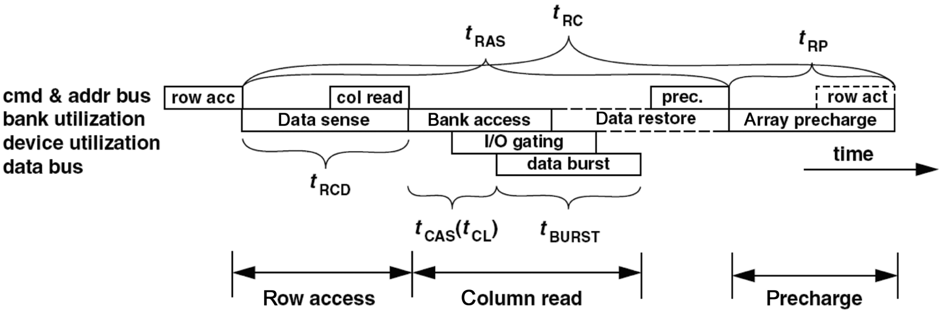


Fig. 4. A cycle of a DRAM device access to read data [13].

$$t_{READ} = t_{CMD} + t_{RAS} + t_{RP} \tag{1}$$

$$t_{READ} = t_{CMD} + t_{RCD} + t_{CAS} + t_{BURST} \tag{2}$$

C. Timing of Consecutive DRAM Device Accesses

Multiple consecutive DRAM accesses happen more frequently on a multicore processor than on a uniprocessor for the following reasons. First, the number of DRAM accesses issued concurrently increases as the number of cores increases. Second, there is no data dependency or control flow constraint between the DRAM accesses from the threads running on different cores. However, the DRAM commands of consecutive DRAM accesses can rarely be fully pipelined, because these commands need to share resources in the DRAM devices concurrently. The degree of the DRAM command pipelining depends on the spatial locality of the data fetched by the consecutive DRAM accesses, as well as the state of the DRAM devices, which can possibly impact the latency of a DRAM access among a sequence of consecutive DRAM accesses.

Fig. 5 demonstrates the latencies of two consecutive DRAM device accesses in three cases with different data spatial locality between them. Both DRAM accesses are ready to be executed at the same time. The latency of the first access T_1 is the same in all cases according to Equation (2), which is not affected by the second access at all. However, the latency of the second access T_2 varies, because the degrees of DRAM command pipelining are different in three cases. As the data fetched by both accesses are in the same bank in Fig. 5a, the first command of the second access is not released until the data fetched by the first access are restored and the row has been precharged. Because the second access has to wait for the full cycle of the first access, and only the transportation of its row access command is pipelined with the precharge phase of the first access, its latency T_2 can be described in Equation (3). In Fig. 5b, where both accesses fetch the data in different banks of the same rank, both accesses will only conflict on the I/O gating circuit and the data bus. Also, as the row required by the second access should be precharged in the case of a bank conflict, the first command of the second access is executed after the start of the first access with the time interval of at least $t_{RP} + t_{RCD}$ to avoid conflicts. So, the latency of the second access is defined as the sum of this minimal time interval and the latency of a read cycle without the precharging phase, as described in Equation (4). In Fig. 5c, the data fetched by both accesses are on different ranks, so both accesses only conflict on the data bus. Similar to Fig. 5b, the minimal timing interval between the start of both accesses to avoid the conflict turns out to be only t_{BURST}. T_2 in this case can be computed by Equation (5).

$$T_{2\_case\_a} = t_{RAS} + t_{RP} + t_{CMD} + t_{RCD} + t_{CAS} + t_{BURST} \tag{3}$$

$$T_{2\_case\_b} = t_{RCD} + t_{RP} + t_{CMD} + t_{RCD} + t_{CAS} + t_{BURST} \tag{4}$$

$$T_{2\_case\_c} = t_{BURST} + t_{CMD} + t_{RCD} + t_{CAS} + t_{BURST} \tag{5}$$

It can easily be concluded that the later of two consecutive DRAM accesses will have the worst-case latency if both accesses fetch data on the same bank. Furthermore, it can be extended to the case of N consecutive DRAM accesses ($N > 2$), since they can be divided into multiple instances of two consecutive DRAM accesses. Therefore, the worst-case latency of a given access is T_n, as shown in Equation (6), if it is the last one in the sequence of consecutive DRAM accesses, and all the accesses fetch the data in the same bank as well.

$$T_n = (N-1) * (t_{RAS} + t_{RP}) + t_{CMD} + t_{RCD} + t_{CAS} + t_{BURST} \tag{6}$$

V. ANALYZING WORST-CASE DRAM PERFORMANCE

Our assumptions: In this work, we develop a WCET analysis method to derive the WCET for real-time applications running on multicore processors by modeling and bounding the worst-case DRAM performance. We focus on studying the instruction accesses through the memory hierarchy, and assume the data cache is perfect. Also, in our WCET analysis, we have not considered the timing

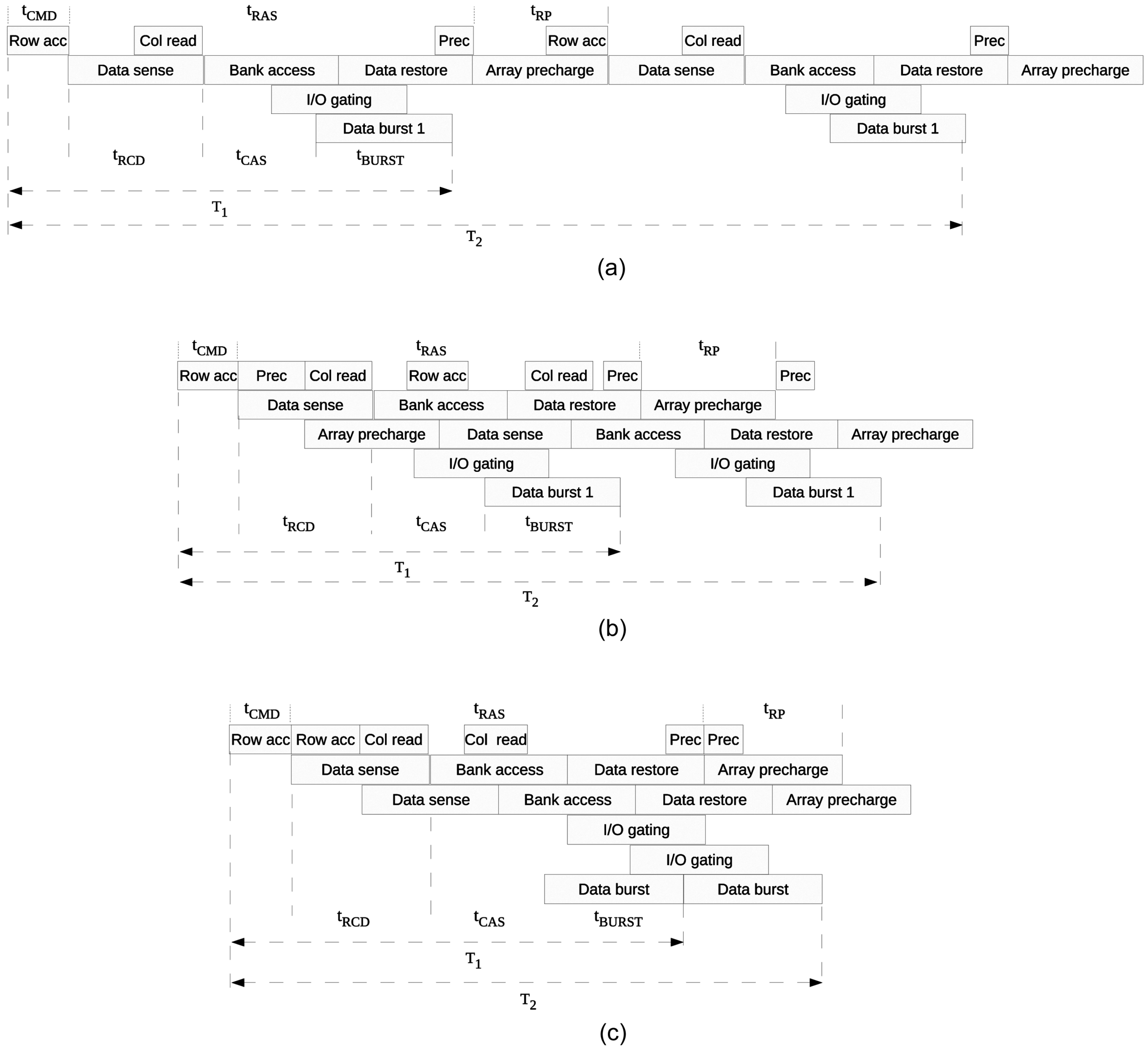


Fig. 5. The latencies of two consecutive DRAM device accesses with different spatial locality, which are calculated using Equations (3)–(5), respectively. (a) Two consecutive DRAM memory accesses to the same bank, (b) two consecutive DRAM memory accesses to different banks on the same rank, and (c) two consecutive DRAM memory accesses to different ranks.

caused by bus conflicts and DRAM memory refreshing. We assume in-order pipelines without using branch prediction. In our software model, we assume a set of independent tasks execute concurrently on different cores, and there is no data sharing or synchronization among those tasks.

We extend the implicit path enumeration technique (IPET) technique [4] to obtain the WCET of a real-time application on a multicore processor with its worst-case DRAM performance. In IPET, the objective function of the integer ILP problem to calculate the WCET is subject to structural constraints, functionality constraints, and micro-architecture constraints, all of which can be usually described as linear equations or inequalities. Also, some equations are created to describe the equality relationship between the execution counts of basic blocks and cache line blocks to connect the control flow graph (CFG) and the cache conflict graph (CCG).

As there are only private L1 and L2 caches in the multicore architecture studied, our WCET analysis approach only needs to construct the CCG on each L1 and L2 cache to build the cache constraints. The CCG on an L2 cache describes the constraints between the L2 cache accesses and the DRAM accesses, as an L2 cache miss will result in a DRAM access. In order to consider the worst-case DRAM performance, the objective function of the WCET for each thread is given in Equation (7), which includes the computing time, the latency to access the L1 cache, and the latency to access the L2 cache. The last part $\sum_{i=1}^{n} C_i*M_i$ indicates the total latency of the DRAM accesses. Specifically, C_i is the worst-case latency of a given DRAM access, and M_i denotes its number of execution, which is bounded by the cache constraints from the CCG of the L2 cache.

$$WCET = \sum Computing\ Time + \sum L1\$\ Latency + \sum L2\$\ Latency + \sum_{i=1}^{n} C_i * M_i \quad (7)$$

A. Conservative Approach

If there are N identical cores sharing the DRAM on a multicore processor, it will be safe but pessimistic to estimate the worst-case latency of each DRAM access based on two assumptions. The first assumption is that each DRAM access from a thread is always issued with other *N-1* DRAM accesses simultaneously from other *N-1* co-running threads, and this access starts to be executed after all other accesses finish the execution. Second, all these consecutive DRAM accesses fetch the data in the same bank, which will result in the worst-case scenario, as described in Section IV-C. Therefore, the worst-case latency of a DRAM access can be computed by Equation 8, where $(N-1) * (t_{RAS}+t_{RP})$ is the delay to wait for the finish of the other *N-1* accesses, and $t_{CMD}+t_{RCD}+t_{CAS}+t_{BURST}$ stands for the actual DRAM device access latency for this access. In addition, the calculation of C_i should include the latency of bus access t_{BUS} and the queuing delay from the memory controller t_{QUEUE}, both of which are safely estimated as constants, as discussed in Section IV. Although it is safe, this approach is pessimistic, which may result in much overestimation.

$$C_i = (N-1) * (t_{RAS}+t_{RP}) + t_{CMD} + t_{RCD} + t_{CAS} + t_{BURST} + t_{BUS} + t_{QUEUE} \quad (8)$$

B. A Basic Approach

In order to reduce the overestimation in the conservative approach, the basic approach is proposed by considering the effect of DRAM command pipelining. As discussed in Section IV-C, the performance of DRAM command pipelining among consecutive DRAM accesses depends on the spatial locality of the data fetched. The worst-case situation of DRAM command pipelining happens when the data fetched by consecutive DRAM accesses are on the same bank, which would degrade the degree of DRAM command pipelining mostly. Given a thread (task), the basic approach first checks the DRAM address of the data fetched by each DRAM access. Then, it determines the maximum number of DRAM accesses from other threads fetching the data on the same bank with this access. If no DRAM access from other threads is found to fetch the data on the same bank, it then examines the number of DRAM accesses from other threads fetching the data on the same rank.

The basic approach is described in Algorithm 1. The input of this algorithm is N co-running threads, and the output is the WCET objective function of each thread. The worst-case DRAM performance of each co-running thread is estimated individually. The worst-case latency for a given DRAM access M_j in a given thread T_i is estimated as follows. First, *addr*, the DRAM address of the data fetched by M_j, is translated from the physical address according to the given address mapping policy, and the bank id *b* and rank id *r* are both derived from *addr.* Then, the number of other co-running threads with DRAM memory accesses fetching the data on the same bank *b* is denoted as N_b at line 9. These N_b threads are excluded from the remaining procedure. Since it is possible that DRAM accesses from the remaining threads are still fetching data on the same bank b_k other than *b*, the maximum number of threads with DRAM accesses fetching data on the bank of b_k is calculated as $N_{0b}[k]$ and stored in an array from lines 11 to 15. These threads are also excluded from the remaining procedure. In the next step, the number of threads with DRAM accesses fetching the data in the same rank of *r* is calculated as N_r. At the end of the processing for M_j, the number of the threads with DRAM accesses fetching the data on different ranks is computed as N_{dr}. The worst-case latency C_j for M_j is calculated based on Equations (3)–(5). The algorithm will terminate when the worst-case DRAM performance has been estimated and added into the WCET objective function based on Equation (7) for all the co-running threads.

C. An Example of the Basic Approach

An example of the basic approach is shown in Fig. 6. In this example, there are 4 threads running concurrently on a multicore processor with 4 cores. In each thread, there are multiple DRAM accesses. It is supposed that there are 4 ranks in the DRAM of this example, and each rank has 4 banks. A DRAM access is represented by a rectangle with the name M_i. The numbers inside the parentheses denote the DRAM address of the data fetched by this access, where the first number is the *rank id* and the second number is the *bank id*. For example, the first DRAM access in *Thread A* is named M_1 and its *rank id* and *bank id* are both 1. In addition, all the DRAM accesses are connected by the edges to indicate the timing order derived from the CFG.

The estimation procedure for *Thread A* starts with checking the DRAM address in M_1. Then, M_5 in *Thread B* and M_{10} in *Thread C* are found to fetch the data in *bank 1* of *rank 1* as well. So, N_b for M_1 is 2. As *Thread D* does not have any DRAM access to *rank 1*, N_r is equal to 0 and N_{dr} is 1. Therefore the worst-case latency of M_1 can be computed by Equation (9). Only one DRAM access M_9 in *Thread C* fetches the data in *bank 2* of *rank 1*, which is the same as M_2, and M_8 and M_{13} are found to access *rank 3* in *Thread B* and *Thread D*, respectively, so N_b is 1, N_r is 2, and N_{dr} is 1 for M_2. The worst-case latency for M_2 can

Algorithm 1 *Basic Approach*

```
begin
input: N co-running threads
output: the objective functions of all the threads
for thread $T_i$ in the co-running threads do
  for each DRAM access $M_j$ in thread $T_i$ do
    $addr$ = the DRAM address of the data fetched by $M_j$
    $b$ = the bank id of $addr$
    $r$ = the rank id of $addr$
    $N_b$ = the number of other threads with DRAM accesses to the bank $b$
    $N_{remain} = N_{remain} - N_b$
    while DRAM memory accesses in $N_{remain}$ threads fetching the data in the bank of $b_k$ exist do
      $N_{ob}[k]$ = the maximum number of threads with DRAM access fetching the data in $b_k$
      $N_{remain} = N_{remain} - N_{ob}[k]$
      $k = k + 1$
    end while
    $N_r$ = the number of threads in $N_{remain}$ threads with DRAM accesses fetching the data in the rank $r$
    $N_{dr}$ = the number of threads in $N_{remain}$ threads with DRAM accesses fetching the data in different ranks
    $C_j = N_{dr} * t_{BURST} + N_r * (t_{RCD} + t_{RP}) + (N_b + \sum N_{ob}[k])(t_{RAS} + t_{RP}) + t_{CMD} + t_{RCD} + t_{CAS} + T_{BURST} + t_{BUS} + t_{QUEUE}$
  end for
  $WCET_i = \sum Computing\ Time + \sum L1\$\ Latency + \sum L2\$\ Latency + \sum_{j=1}^{n} C_j * M_j$
end for
end
```

be calculated by Equation (10). This case is similar to the cases of M_3 and M_4. Although no other DRAM access fetches the data on the same bank as M_3 and M_4, there are either M_5 and M_{10} or M_6 and M_{11} in *Thread B* and *Thread C* accessing the same bank. However, there is no DRAM access fetching the data in same rank with M_3, such that the worst-case latency of M_3 can be derived as Equation (11). In contrast, either M_{12} or M_{14} fetches the data in *rank 4*. Therefore, the worst-case latency of M_4 can be computed by Equation (12).

Following the specific timing parameters given in Table 2, the worst-case latency for C_1, C_2, C_3, and C_4 are calculated as 63, 59, 63, and 66 cycles, respectively. In contrast, the worst-case latency for all these DRAM accesses in this example is estimated to be 73 cycles by the conservative approach. It is clear that the overestimation of the worst-case DRAM performance in the conservative approach can be reduced by the basic approach.

$$C_1 = t_{CMD} + t_{RCD} + t_{CAS} + 2 * t_{BURST} + 2 * (t_{RAS} + t_{RP}) + t_{BUS} + t_{QUEUE} \tag{9}$$

$$C_2 = t_{CMD} + t_{RCD} + t_{CAS} + t_{BURST} + t_{RAS} + t_{RP} + 2 * (t_{RP} + t_{RCD}) + t_{BUS} + t_{QUEUE} \tag{10}$$

$$C_3 = t_{CMD} + t_{RCD} + t_{CAS} + 2 * t_{BURST} + 2 * (t_{RAS} + t_{RP}) + t_{BUS} + t_{QUEUE} \tag{11}$$

$$C_4 = t_{CMD} + 2 * t_{RCD} + t_{CAS} + t_{BURST} + 2 * t_{RAS} + 3 * t_{RP} + t_{BUS} + t_{QUEUE} \tag{12}$$

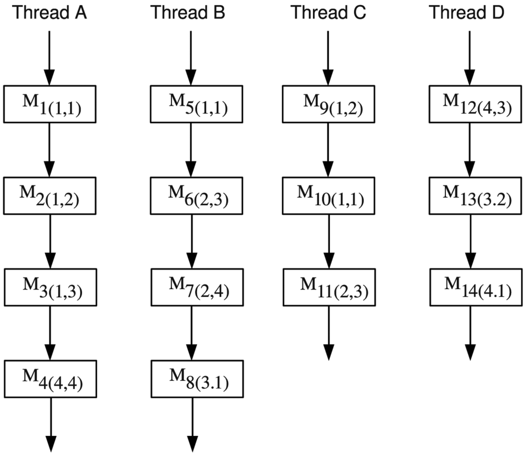


Fig. 6. An example of estimating the worst-case DRAM performance by the basic approach.

D. An Enhanced Approach

Although the basic approach considers the effect of DRAM command pipelining on the worst-case DRAM performance, there is still overestimation due to the timing order constraints of the co-running DRAM accesses, since the order of DRAM accesses of a given thread can impact the timing order of DRAM accesses of other threads. This problem can be explained by the example in Fig. 7. Assuming that there are two DRAM accesses in each thread, Thread 1 contains M_i, M_j and Thread 2 contains M_k, M_l, where M_i and M_l fetch data on the same bank and so do M_j and M_k. If the timing order is not taken into account, there are two bank conflicts. However, it is clear that the timing order of the threads is violated in the case of two bank conflicts. If M_i and M_l are issued simultaneously from both threads and a bank conflict is taken into account, Thread 2 must have reached the end of M_l, and Thread 1 has not started the execution of M_j. This indicates that the bank conflict between M_j and M_k cannot happen. The same analysis can be applied between M_j and M_k. Therefore, there is possibly only one bank conflict in the worst-case. Similarly, the same analysis can be applied to rank conflicts.

An enhanced approach is proposed to compute the worst-case DRAM performance more accurately by eliminating the bank conflicts and rank conflicts that can never occur. A type of variables named C_{pair} is introduced to define a pair of conflicting DRAM accesses between two co-running threads. The value of C_{pair} is 1 when the conflict may happen, whereas it is 0 if a conflict cannot happen. Initially, a C_{pair} set is constructed on both bank and rank levels to denote the possible bank or rank conflicts between the DRAM accesses from both threads only based on the DRAM addresses of the data to be fetched. The next step is to remove the C_{pair} set that are logically impossible due to the execution order of both threads from the C_{pair} set. This is implemented according to the algorithms of the construction of the C_{graph} and finding the valid C_{pair} sets proposed by Yan and Zhang [17].

The construction of C_{graph} is described in Algorithm 2. The C_{graph} is a directed graph, where all the vertices are the C_{pair} set, and they are connected by edges. An edge is added if and only if the execution of the DRAM accesses in the two C_{pair} are logically possible by checking the control flow graph of each thread. The next step is to construct the valid C_{pair} set as described in Algorithm 3. This algorithm initially uses Tiernan's algorithm [18] to find all the cycles in C_{graph}, and then inserts them into V. Then, the algorithm validates each cycle in V, as shown in Lines 6 to 10. If a cycle is invalid, it is then removed from V. The algorithm finishes if all the cycles contained by other cycles are removed from V.

Algorithm 2 *C_{graph} Construction*

```
begin
Input: $C_{pair}$ in Thread 1 and Thread 2
Output: $C_{graph}$
for each $c_i$ in $C_{pair}$ set in Thread 1 do
    for each $c_j$ in $C_{pair}$ set in Thread 2 do
        if $c_i$ does not conflicts with $c_j$ then
            add edge from $c_i$ to $c_j$
            add edge from $c_j$ to $c_i$
        end if
    end for
end for
end
```

Algorithm 3 *V alid C_{pair} Set Construction*

```
1:  begin
2:  Input: $C_{graph}$
3:  Output: valid $C_{pair}$ set $V$
4:  find all the cycles in $C_{graph}$
5:  insert each cycle to $V$
6:  for each cycle in $V$ do
7:      if check cycle valid is not true then
8:          remove the cycle from $V$
9:      end if
10: end for
11: for each cycle in $V$ do
12:     if if cycle is contained by other cycle in $V$ then
13:         remove the cycle from $V$
14:     end if
15: end for
16: end
```

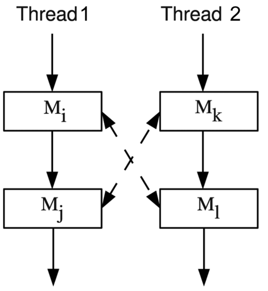


Fig. 7. An example of the overestimation without considering the timing order constraints of different threads.

VI. EVALUATION METHODOLOGY

In our evaluation, the simulation tool SimpleScalar [19] is extended to simulate the multicore architecture to obtain the simulated WCET. Also, the DRAM simulation tool named DRAMSim [20] is integrated into the extended SimpleScalar to support the accurate timing simulation of DRAM memory access. The WCET analysis tool consists of a front end and a back end. The front end of the WCET analysis tool compiles benchmarks into common object file format (COFF) binary code using the GCC compiler, which is targeted to SimpleScalar. Then, it obtains the global CFG and related information about the instructions by disassembling the binary code generated. Subsequently, the back end compiler performs static cache analysis and the DRAM memory access timing analysis with the enhanced ILP method. Finally a commercial ILP solver, CPLEX [21], is used to solve the ILP problem to obtain the estimated WCET.

In our evaluation, a homogeneous multicore processor is simulated, and each core has an inclusive two-level

Table 2. Timing parameters of the DRAM memory

Parameter	Value	Parameter	Value
t_{BUS}	10 cycles	t_{QUEUE}	10 cycles
t_{BURST}	4 cycles	t_{CMD}	1 cycles
t_{CAS}	3 cycles	t_{RAS}	10 cycles
t_{RCD}	3 cycles	t_{RP}	4 cycles

Table 3. Basic configuration of the cache in the simulated processor

L1 I-cache	256 bytes, direct-map, 16 bytes block, 1 cycle latency
L1 D-cache	perfect
L2 cache	1024 bytes, direct-map, 32 bytes block, 10 cycle latency

Table 4. Basic configuration of the DRAM memory

Channel	1	Rank	2
Bank	8	Row	64
Column	16	Column width	8 bytes

cache [22]. The cache configuration in each core is described in Table 3. In order to focus on DRAM memory accesses from instruction accesses, the L1 data cache in each core is assumed to be perfect. The size of the shared DRAM memory between different cores in our evaluation architecture is 128 kbytes, and its configuration is shown in Table 4. The timing parameters of the DRAM memory are given in Table 2.

We use 14 benchmarks from the Malardalen WCET benchmark suite [23], and run them on processors with 2 cores, 4 cores, and 8 cores. The salient characteristics of the benchmarks can be found in Table 5. In our experiments, we study the following four schemes:

Conservative scheme: the WCET is computed by the conservative approach.

Basic scheme: the WCET is calculated by the basic approach.

Enhanced scheme: the WCET is derived by the enhanced approach.

Simulated scheme: the WCET is obtained by the simulation.

VII. EXPERIMENTAL RESULTS

A. Results of Basic Scheme

Table 5. Salient characteristics of the benchmarks in case of 2 cores

Benchmark	WCET	Memory access time	L1 miss rate (%)	L2 miss rate (%)
Bs	689	373	28.24	45.83
Fibcall	589	342	10.53	45.45
Insertsort	1397	507	5.33	45.45
Matmul	1812	673	8.55	27.03
Biquad	2066	1113	16.57	35.56
Sqrt	2155	1096	9.34	48.44
Jfdct	2832	1564	14.91	38.18
Startup	4640	1220	7.42	16.51
Qurt	7175	2827	6.00	42.93
Ud	11846	1990	8.17	5.89
Ludcmp	14207	4560	11.10	15.87
Select	20483	7544	5.64	27.95
Qsort	20786	10831	10.32	34.71
Fft1	32523	15791	9.19	31.09

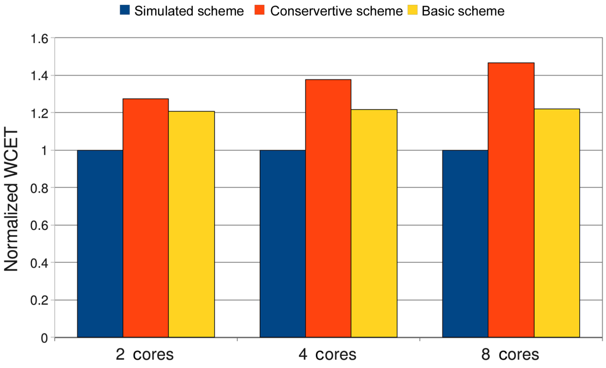


Fig. 8. The comparison of the averaged worst-case execution times (WCETs) of the conservative scheme and the basic scheme normalized with those of the simulated scheme in case of 2 cores, 4 cores, and 8 cores.

We first study the estimated WCET obtained by both the conservative approach and the basic approach in cases of 2 cores, 4 cores, and 8 cores. Fig. 8 demonstrates the averaged WCETs of the conservative scheme and the basic scheme, which are normalized with those of the simulated scheme in case of 2 cores, 4 cores, and 8 cores.

The overestimation of the conservative scheme, as compared to the simulated scheme, increases with the increase of the number of cores in the experiments. For 2 cores, the normalized WCET of the conservative scheme is 27.5% larger than that of the simulated scheme on average, and this difference increases to 37.5% and 46.7% for 4 cores and 8 cores, respectively. This can be explained by Equation (8) derived from the conservative

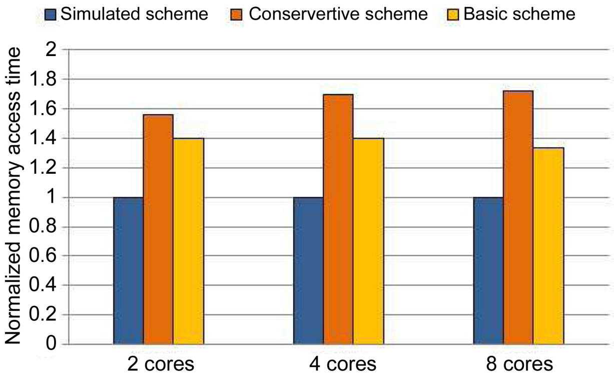


Fig. 9. The comparison of the averaged memory access time of the conservative scheme and the basic scheme normalized with that of the simulated scheme in case of 2 cores, 4 cores and 8 cores.

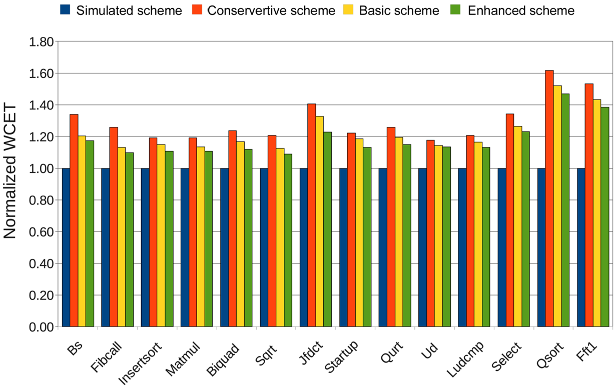


Fig. 10. The comparison of the worst-case execution times (WCETs) of the conservative scheme, the basic scheme, and the enhanced scheme normalized with that of the simulated scheme in case of 2 cores.

approach, where the number of cores N is an important factor to determine the value of this equation. With the increase of N, the assumed conditions of the conservative approach are less likely to happen.

In contrast, the WCET differences between the basic scheme and the simulated scheme are much lower, at 20.7%, 21.8%, and 21.9% for 2 cores, 4 cores, and 8 cores, respectively. The reason for the lower overestimation is that the basic approach considers the effects of DRAM command pipelining among the consecutive DRAM accesses. The overestimation of the basic approach originates from the assumption that a given DRAM access is always executed after other co-running DRAM accesses, and from the ignorance of the timing order constraints of the co-running threads.

We also compare the memory access time of the conservative scheme, the basic scheme, and the simulated scheme in Fig. 9. For the conservative scheme, the memory access time is overestimated by 56.3%, 69.8%, and 72.1% compared to the simulated scheme for 2 cores, 4 cores, and 8 cores, respectively. The Basic Scheme overestimates the memory access time by 39.9%, 39.7%, and 33.3% compared with the simulated scheme for 2 cores, 4 cores, and 8 cores, respectively.

B. Results of Enhanced Scheme

Fig. 10 compares the WCETs of each benchmark for all four schemes. As expected, the enhanced scheme has the tightest WCET results, which are only 16.8% higher on average than that of the simulated WCET. The tighter WCET estimation is a result of the more accurate analysis of the memory access time. Fig. 11 shows the comparison of the memory access time of each benchmark for all

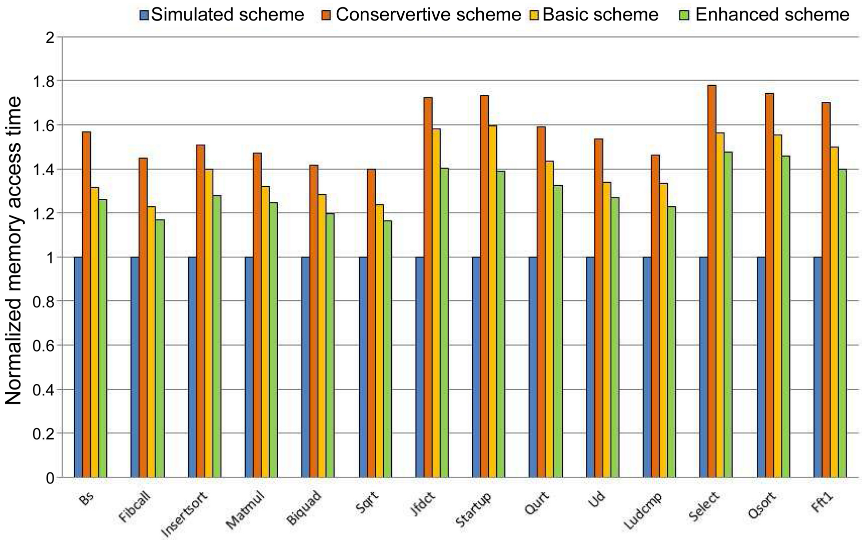


Fig. 11. The comparison of the memory access time of the conservative scheme, the basic scheme, and the enhanced scheme normalized with that of the simulated scheme in case of 2 cores.

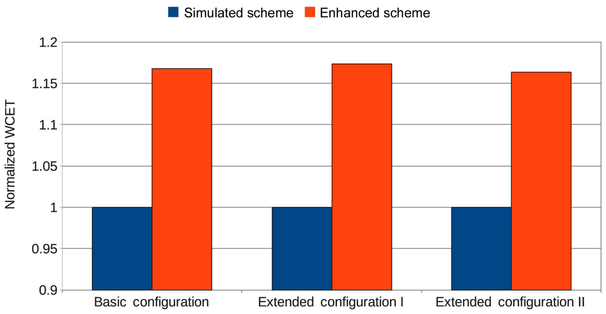


Fig. 12. The comparison of the worst-case execution time (WCET) of the enhanced scheme normalized with that of the simulated scheme in basic configuration, extended configuration I and extended configuration II for 2 cores.

four schemes. Clearly, the enhanced scheme has the smallest overestimation, which is 30.4% on average. In contrast, the conservative approach and the basic approach result in 57.7% and 40.6% overestimation on average, respectively.

C. Sensitivity Analysis

In addition, the basic configuration of the DRAM memory in our experiment is extended to two configurations by changing the number of the banks while keeping the total size of the DRAM the same. In extended configuration I, the number of banks is increased to 16, and the number of rows in each bank is reduced to 32. In contrast, the number of banks is decreased to 4, but the number of rows is increased to 128 in extended configuration II. Experiments are conducted for the Enhanced Scheme and the Simulated Scheme in these three DRAM configurations for 2 cores. Fig. 12 shows the comparison of the WCET of the enhanced scheme normalized with that of the simulated scheme in the basic configuration, extended configuration I, and extended configuration II for 2 cores. The estimated WCETs of the enhanced scheme are 16.8%, 17.3%, and 16.3% larger than those of the simulated scheme for the basic configuration, extended configuration I, and extended configuration II, respectively, indicating that the enhanced approach also works well with different numbers of banks.

VIII. CONCLUSIONS

We have address the difficulties in bounding the worst-case DRAM performance for a real-time application running on a multicore processor. The timing characteristics of DRAM accesses were investigated based on a generic DRAM access protocol, and it was found that the DRAM command pipelining of consecutive DRAM accesses can possibly impact the worst-case DRAM performance of a real-time application running on a multicore processor. A basic approach has been proposed for bounding the worst-case DRAM performance by considering the worst-case timing effects of DRAM command pipelining for the sequences of consecutive DRAM accesses. An enhanced approach has also been proposed for reducing the overestimation of DRAM access time by checking the timing order constraints of the co-running applications to identify and remove impossible DRAM access sequences.

Our experiments show that compared with the conservative approach, which assumes that no DRAM command pipelining exists, the basic approach can bound the WCET more tightly. For example, 15.7% improved performance was achieved on average for a 4-core processor. Moreover, the enhanced approach further improves the tightness of WCET by 4.2% on average compared with the basic approach.

In the future, our work will be extended to bound the worst-case DRAM performance with data DRAM accesses. The impact of the DRAM refresh on the WCET of a real-time application [24] will be integrated in future work.

ACKNOWLEDGMENTS

This work was funded in part by the United States National Science Foundation (NSF) Computing and Communication Foundations (CCF) grant 0914543.

REFERENCES

1. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Walley et al., “The worst-case execution time problem: overview of methods and survey of tools,” *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, article no. 36, 2008.
2. C. A. Healy, D. B. Whalley, and M. G. Harmon, “Integrating the timing analysis of pipelining and instruction caching,” in *Proceedings the 16th IEEE Real-Time Systems Symposium*, Pisa, Italy, 1995, pp. 288-297.
3. F. Stappert, A. Ermedahl, and J. Engblom, “Efficient longest executable path search for programs with complex flows and pipeline effects,” in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, Atlanta, GA, 2001, pp. 132-140.
4. Y. T. S. Li and S. Malik. “Performance analysis of embedded software using implicit path enumeration,” in *Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference*, San Francisco, CA, 2005, pp. 456-461.
5. Y. T. S. Li, S. Malik, and A. Wolfe, “Cache modeling for real-time software: beyond direct mapped instruction caches,” in *Proceedings of the 17th IEEE Real-Time Systems Symposium*, Washington, DC, 1996, p. 254.
6. G. Ottosson and M. Sjodin, “Worst-case execution time analysis for modern hardware architectures,” in *Proceedings of ACM/SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems*, Las Vegas, NV, 1997.

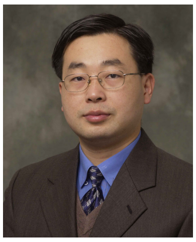
7. T. Moscibroda and O. Mutlu, "Memory performance attacks: denial of memory service in multi-core systems," in *Proceedings of the 16th USENIX Security Symposium*, Boston, MA, 2007.
8. J. H. Ahn, M. Erez, and W. J. Dally, "The design space of data-parallel memory systems," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, Tampa, FL, 2006, article no. 80.
9. G. L. Yuan and T. M. Aamodt, "A hybrid analytical DRAM performance model," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, Austin, TX, 2009.
10. I. Y. Bucher and D. A. Calahan, "Models of access delays in multiprocessor memories," *IEEE Transactions on Parallel Distributed Systems*, vol. 3, no. 3, pp. 270-280, 1992.
11. H. Choi, J. Lee and W. Sung, "Memory access pattern-aware DRAM performance model for multi-core systems," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, TX, 2011, pp. 66-75.
12. R. Bourgade, C. Ballabriga, H. Casse, C. Rochange, and P. Sainrat, "Accurate analysis of memory latencies for WCET estimation," in *Proceedings of the 16th International Conference on Real-Time and Network Systems*, Rennes, France, 2008, pp. 161-170.
13. B. Jacob, S. W. Ng, and D. T. Wang, Memory Systems: Cache, DRAM, Disk, Amsterdam: Elsevier, 2008.
14. E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana, "Self-optimizing memory controllers: a reinforcement learning approach," in *Proceedings of the 35th International Symposium on Computer Architecture*, Beijing, China, 2008, pp. 39-50.
15. K. J. Nesbit, N. Aggarwal, J. P. Laudon, and J. E. Smith, "Fair queuing memory systems," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, Orlando, FL, 2006, pp. 208-222.
16. S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, Vancouver, Canada, 2000, pp. 128-138.
17. J. Yan and W. Zhang, "WCET analysis for multi-core processors with shared L2 instruction caches," in *Proceedings of 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, St. Louis, MO, 2008, pp. 80-89.
18. J. C. Tiernan, "An efficient search algorithm to find the elementary circuits of graph," *Communication of the ACM*, vol. 13, no. 12, pp. 722-726, 1970.
19. SimpleScalar, http://www.simplescalar.com.
20. DRAMSim2, http://www.ece.umd.edu/dramsim/.
21. IBM ILOG CPLEX Optimizer, http://www.ilog.com/products/cplex/.
22. D. Hardy and I. Puaut, "WCET analysis of multi-level non-inclusive set-associative instruction caches," in *Proceedings of the 29th Real-Time Systems Symposium*, Barcelona, Spain, 2008, pp. 456-466.
23. J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Malardalen WCET benchmarks: past, present and future," in *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis*, Brussels, Belgium, 2010, pp. 137-147.
24. P. Atanassov and P. Puschner, "Impact of DRAM refresh on the execution time of real-time tasks," in *Proceedings of IEEE International Workshop on Application of Reliable Computing and Communication*, Seoul, Korea, 2001.

Yiqiang Ding

Yiqiang Ding is currently a Ph.D. student in Electrical and Computer Engineering at Virginia Commonwealth University. He received the B.S. degree of computer science in 2002 and the M.S. degree of computer engineering in 2005 from the Beijing University of Posts and Telecommunications in China. He worked in Motorola China Design Center as a system engineer from 2005 to 2007. His research interests are in embedded and real-time computing systems, computer architecture and compiler.

Lan Wu

Lan Wu received her B.S. of Computer Science in July 2004 from University of Science and Technology of China, and M.S. of Computer Engineering in July 2007 from North China Institute of Computing Technology. She is now a Ph.D. student of Computer Engineering in Virginia Commonwealth University. Her research interests focus on Real-time and Embedded Systems, Computer Architecture and Virtualization.



Wei Zhang

Wei Zhang is an associate professor in Electrical and Computer Engineering of Virginia Commonwealth University. Dr. Zhang received his Ph.D. from the Pennsylvania State University in 2003. From August 2003 to July 2010, Dr. Zhang worked as an assistant professor and then as an associate professor at Southern Illinois University Carbondale. His research interests are in embedded and real-time computing systems, computer architecture, compiler, and low-power systems. Dr. Zhang has received the 2009 SIUC Excellence through Commitment Outstanding Scholar Award for the College of Engineering, and 2007 IBM Real-time Innovation Award. His research has been supported by NSF, IBM, Intel, Motorola and Altera. He is a senior member of the IEEE. He has served as a member of the organizing or program committees for several IEEE/ACM international conferences and workshops.