# Finding an Upper Bound on the Increase in Execution Time Due to Contention on the Memory Bus in COTS-Based Multicore Systems

Björn Andersson

Arvind Easwaran

Jinkyu Lee

HURRAY-TR-100104

Version:

Date: 01-25-2010

# Finding an Upper Bound on the Increase in Execution Time Due to Contention on the Memory Bus in COTS-Based Multicore Systems

Björn Andersson, Arvind Easwaran, Jinkyu Lee

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

## Abstract

Contention on the memory bus in COTS based multicore systems isbecoming a major determining factor of the execution time of atask. Analyzing this extra execution time is non-trivial because(i) bus arbitration protocols in such systems are oftenundocumented and (ii) the times when the memory bus isrequested to be used are not explicitly controlled by the operatingsystem scheduler; they are instead a result of cache misses.We present a method for finding an upper bound on the extraexecution time of a task due to contention on the memory bus in COTSbased multicore systems. This method makes no assumptions on the busarbitration protocol (other than assuming that it iswork-conserving).

# Finding an Upper Bound on the Increase in Execution Time Due to Contention on the Memory Bus in COTS-Based Multicore Systems

Björn Andersson and Arvind Easwaran
*Polytechnic Institute of Porto, Portugal*
*bandersson@dei.isep.ipp.pt, aen@isep.ipp.pt*

Jinkyu Lee
*Dept. of Computer Science, KAIST, South Korea*
*jinkyu@cps.kaist.ac.kr*

***Abstract*—Contention on the memory bus in COTS based multicore systems is becoming a major determining factor of the execution time of a task. Analyzing this extra execution time is non-trivial because (i) bus arbitration protocols in such systems are often undocumented and (ii) the times when the memory bus is requested to be used are not explicitly controlled by the operating system scheduler; they are instead a result of cache misses. We present a method for finding an upper bound on the extra execution time of a task due to contention on the memory bus in COTS based multicore systems. This method makes no assumptions on the bus arbitration protocol (other than assuming that it is work-conserving).**

## I. Introduction

The multicore processor is today a generic building block in the design of embedded real-time computing systems. Typically, a multicore processor chip is comprised of a set of processor cores, each with a private cache memory (L1) and potentially a cache memory (L2) that is shared among the processor cores. This chip is connected through an interconnection network (such as a bus) to a set of main-memory modules. When an instruction cannot be served by the on-chip caches, it is necessary for the processor to perform a transaction on the interconnection network in order to fetch data from the main-memory modules. Already today, this interconnection network is a performance bottleneck for many applications [1]. Moreover, since the number of processor cores in a multicore chip is increasing dramatically, the amount of traffic on the interconnection network increases accordingly, and consequently this problem is expected to be exacerbated in the future [1], [2].

The interconnection network has an impact also on the execution time of an individual task. Consider a task $\tau_1$ executing on processor $P_1$ and another task $\tau_2$ executing on processor $P_2$. The task $\tau_2$ generates a cache miss but before the transaction on the interconnection network has finished, task $\tau_1$ generates a cache miss as well. Then in this case, serving the cache miss of $\tau_1$ requires more time than would have been the case if $\tau_1$ was the only task in the system because the bus must finish serving the other processor. Therefore, it is important to develop a method for finding an upper bound on the extra execution time of a task due to contention on the interconnection network between processor cores. This problem differs from studies in Worst-Case Execution-Time (WCET) analysis because WCET analysis is performed on a task in isolation, whereas our problem concerns the interaction of tasks. It is also different from works in real-time communication because these studies find an upper bound on the queuing time of individual message transmission requests, whereas our problem concerns the cumulative waiting time of many requests to perform transactions on the memory bus.

The scientific community has nonetheless provided some initial insights into the problem of contention on the interconnection network. The software of a task can be structured to be separated into a fetch phase (where cache misses are allowed) and an execution phase (where cache misses are not allowed) and then the memory bus and tasks are scheduled to ensure that no two processor cores are in a fetch phase simultaneously [3], thus eliminating contention. Or, a rate-limiter is added to the memory controller to ensure that no processor core will generate too much traffic in a time interval of pre-specified duration and then network calculus is used to analyze the processor cores [4]. Unfortunately, common to these approaches is that they require control of the arbitration of the memory bus and hence they cannot be used on COTS-based multicores.

In this work we develop the first approach for finding an upper bound on the extra execution time of a task due to contention on the memory bus in COTS-based multicore systems. We also present a technique for characterizing the interconnection-network-traffic generation pattern of tasks, which is then used as input for finding the upper bound on task execution time. In taking this first step, we make the following assumptions:

A1. The interconnection network is a shared bus (denoted as *memory bus* henceforth);
A2. The shared L2 cache is either partitioned between the processor cores or disabled if it cannot be partitioned. The rationale for this assumption is explained in Section II-C;
A3. Tasks are statically assigned to processors and all jobs execute on the processor to which the task is assigned (partitioned scheduling);
A4. Non-preemptive scheduling is used on each pro-

cessor;

A5. The bus arbitration protocol is work-conserving (that is, the memory bus is idle only if no processor core requests to use the bus).

Additionally, the technique developed in this paper has the following properties.

P1. It does not assume any specific arbitration protocol;

P2. It works for constrained-deadline sporadic tasks.

## II. System Model

### A. Task model

We assume that the workload is comprised of a set of tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. We assume the constrained-deadline sporadic task model which characterizes a task $\tau_i$ by $C_i$, $T_i$ and $D_i$ ($D_i \leq T_i$), with the interpretation that $\tau_i$ releases a sequence of jobs such that two subsequent jobs from $\tau_i$ are released at least $T_i$ time units apart and the exact times of the releases of these jobs cannot be controlled by the scheduling algorithm. Each job released by $\tau_i$ requests to perform $C_i$ units of execution at most $D_i$ time units from its release; otherwise it misses its deadline.

Note that $C_i$ denotes an upper bound on the execution time of a job of task $\tau_i$ when the job executes with no contention on the memory bus from other tasks. $C_i$ can be found by WCET analysis techniques. In this work, we are interested in finding $C'_i$ which denotes an upper bound on the execution time when the job executes *with* contention on the memory bus from other tasks on other processors.

### B. Architecture

Many high-performance processors today allow more than one instruction to be issued in parallel, executed in parallel and committed in parallel. Our model allows this. Some high-performance processors allow instructions to be committed in another order than they were issued. Such processors can significantly complicate WCET analysis [5] even on a single processor without bus contention. Therefore, we assume that processor cores are in-order processors. Also, some processors may switch to another thread when a long-latency instruction is executed (for example a data-cache miss). We assume this is not the case, that is, we assume that when a processor core waits for accessing the memory bus, the processor core is simply stalled. For these reasons, we can compute $C'_i$ as $C'_i = C_i + Q_i$, where $Q_i$ is an upper bound on the amount of time that task $\tau_i$ stalls execution when it executes for $C'_i$ time units because of waiting for accessing the memory bus.

Some computer systems use split transaction buses, where a memory transaction is split into a request part (address) and a reply part (data). We assume that the computer system does not use split transactions.

Different bus transactions may take different amount of time. For example, a bus transaction resulting from a load which succeeds another load instruction with the same row-address can be served faster because only the column address of the DRAM memory needs to be changed. We let $\mathrm{TR}$ denote an upper bound on the amount of time for performing a bus transaction. This time, $\mathrm{TR}$, includes the time to assert the address, the time for the memory latency of the main-memory module and the time for the main-memory module to deliver the data to the processor. Note that $\mathrm{TR}$ denotes an upper bound on the amount of time for performing a bus transaction for the case that the bus was idle; therefore, $\mathrm{TR}$ does not include any queuing delay on the memory bus.

We make no assumption on the number of processor chips or the number of processor cores per processor chip. For example, we allow systems with a single processor chip comprising two processor cores. We also allow systems with two processor chips, each comprising four processor cores. Further, we assume that tasks are already assigned to processor cores. Therefore, we let $\tau^p$ denote the set of tasks assigned to processor core $P_p$.

Because of our definition of $C'_i$ it holds that if a task executed on a processor for $C'_i$ time units, then it performs $C_i$ units of execution. Therefore, we can check schedulability of all tasks assigned to processor $p$ by using a uniprocessor schedulability test on processor $p$, but for each task $\tau_i$, replace $C_i$ by $C'_i$. See for example [6].

### C. Bus requests

We assume that $BR_i(t)$ is a function that denotes an upper bound on the number of bus requests that task $\tau_i$ can generate during any time interval of duration $t$. We will use $BR_i(t)$ in Section III for computing $C'_i$. It is also necessary to find $BR_i(t)$; Section IV shows this.

The function $BR_i(t)$ is clearly dependent on task $\tau_i$. The task generates a bus request when it misses the shared L2 cache and before that it must also have missed its private L1 cache. Note that since the L2 cache is shared, the function $BR_i(t)$ is actually not only a property of task $\tau_i$ and the caches but it is also a property of the interaction between task $\tau_i$ and the tasks on other processor cores. This makes the analysis very complicated. Therefore, in order to simplify our study, we made assumption A2 regarding the shared L2 cache (partitioned or disabled). As a result $BR_i(t)$ does not depend on the behavior of tasks in other processor cores.

### D. Problem statement

Our problem can therefore be stated as:

Given $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each $\tau_i$ is characterized by $T_i$, $D_i$ and $C_i$ and the function $BR_i(t)$, and given that each task is assigned to a processor and executes on it non-preemptively, find $C'_i$ for each $\tau_i$.

## III. Analysis on the Worst-Case Execution Time with Contention on the Memory Bus

In this section, we show how to calculate an upper bound on the execution time when the job executes with contention on the memory bus from other tasks on other processors (*i.e.*, $C'_i$) using given $BR_i(t)$. We only assume that the bus arbitration protocol is work-conserving, and do not assume any other specific arbitration protocol.

Consider a job $J_{i,k}$ released by $\tau_i$. Bus transactions requested from $J_{i,k}$ during its execution can be delayed due to: (a) bus transactions requested from other jobs on other processors during the execution of $J_{i,k}$ and (b) any bus transactions requested but not performed before the execution of $J_{i,k}$ (backlogged bus transactions). Considering this delay, we can calculate $C'_i$ as follows.

$$C'_i = C_i + BL_i \cdot TR + \sum_{\tau_j \in (\tau \setminus \tau^{proc(\tau_i)})} BR_j(C'_i) \cdot TR \qquad (1)$$

where $BL_i$ is the maximum number of backlogged bus transactions at the beginning of the execution of a job released by $\tau_i$, and $proc(\tau_i)$ is a processor to which $\tau_i$ is assigned. We consider (a) by $\sum_{\tau_j \in (\tau \setminus \tau^{proc(\tau_i)})} BR_j(C'_i) \cdot TR$. Here note that we use $BR_j(C'_i)$ instead of $BR_j(C_i)$, so that we can care for additional bus requests caused by additional execution time ($C'_i - C_i$). We also consider (b) by $BL_i \cdot TR$, but we need to know how large $BL_i$ is.

To calculate $BL_i$, we first define the maximum busy period of bus transactions (denoted as $BP$), and $BP$ can be calculated by the following recurrence equation.

$$BP = TR + \sum_{\tau_j \in \tau} BR_j(BP) \cdot TR \qquad (2)$$

The structure of this equation is similar to that of calculating response time. We know $BR_j(\cdot)$ is a non-decreasing function so that we calculate Eq. (2) in an iterative manner (*i.e*, $BP^{(k+1)} = f(BP^{(k)})$ starting from $BP^{(0)} = TR$. If we replace $BP$ with $t$ in Eq. (2), the right-hand side of the equation means the longest time interval of bus transactions requested during $t$. If we consider that bus transactions are continuously performed during the busy period, $BL_i$ can be calculated by the following equation.

$$BL_i = \max_{0 \le t \le BP} \left\lceil \frac{TR + (\sum_{\tau_j \in \tau} BR_j(t) \cdot TR) - t}{TR} \right\rceil \qquad (3)$$

Now we know all variables in Eq. (1) and thus calculate the equation similar to Eq. (2) since the right-hand side of the equation is a non-decreasing function of $C'_i$.

## IV. Analysis on the Maximum Number of Bus Requests

In this section, we show how to calculate a function which is an upper bound on the number of bus requests that task $\tau_i$ can generate during any time interval of duration $t$ (*i.e.*, $BR_i(t)$) from experimental results.

We assume the following can be obtained from experiments:

1) An upper bound on the number of bus requests in an interval $[0,t]$ (denoted as $ARH_i^j(t)$), where $0$ denotes the beginning of execution of the $j^{th}$ execution path of $\tau_i$ and $t$ denotes some future time. Note that this measurement only depends on the execution of task $\tau_i$, because of our assumption A2 on the shared L2 cache. Therefore, we obtain $ARH_i^j(t)$ from measurements by executing jobs of $\tau_i$ independently on a processor core;
2) An lower bound on the number of bus requests in an interval $[0,t]$ (denoted as $ARL_i^j(t)$), where $0$ denotes the beginning of execution of the $j^{th}$ execution path of $\tau_i$ and $t$ denotes some future time;
3) The execution time of the $j^{th}$ path of task $\tau_i$ (denoted as $C_i^j$).

We note that different executions of the same path may result in different numbers of bus requests; this is the reason why we distinguish between $ARH_i^j(t)$ and $ARL_i^j(t)$. We let $paths(\tau_i)$ denote the set of all paths of task $\tau_i$. We assume that $ARH_i^j(t)$ and $ARL_i^j(t)$ are non-decreasing functions for each $i$ and $j$ and regard $C_i$ as $\max_{j \in paths(\tau_i)} C_i^j$.

To calculate $BR_i(t)$, we divide $t$ into three parts: the head, middle, and tail parts, and denote them as $f_i^H(t_H)$, $f_i^M(t_M)$, and $f_i^T(t_T)$, respectively. This idea of splitting a path to simplify the analysis has been used before; for instance, in [7], for analysis of recurring task models. As shown in Figure 1(a), the duration of the middle part is a multiple of $T_i$ so that there exist several complete executions. The duration of the head is less than $T_i$, so that there exists either one partial execution or one complete execution. Ditto for the tail. The head (tail) part includes the end (beginning) point of an execution as shown in Figure 1(a). Since the definition of $BR_i(t)$ is an upper bound on the number of bus requests, it can be calculated as follows.

$$BR_i(t) = \max_{t_H, t_M, t_T} f_i^H(t_H) + f_i^M(t_M) + f_i^T(t_T), \qquad (4)$$

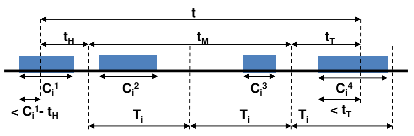
$$\text{where } t_H + t_M + t_T = t, \qquad (5)$$

$$t_H, t_T < T_i, \qquad (6)$$

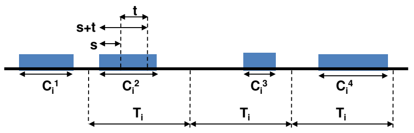
$$t_M = \left[\left\lfloor \frac{t}{T_i} \right\rfloor - 1\right]^+ T_i \text{ or } t_M = \left\lfloor \frac{t}{T_i} \right\rfloor T_i \qquad (7)$$

where $[A]^+$ means $\max\{A, 0\}$. Note that Eq. (7) is derived from Eq. (5) and (6). Also note that $t_M$ is a multiple of $T_i$.

The head part starts from any arbitrary point of an execution but includes the end point of the execution, so that the maximum number of bus requests of the head part is equal to the maximum difference between the number of bus requests during a complete execution and the one during a partial execution.



(a) For the general case



(b) For $t < T_i$

Figure 1. Calculation of $BR_i(t)$

$$f_i^H(t_H) = \max_{j \in paths(\tau_i)} ARH_i^j(C_i^j) - ARL_i^j([C_i^j - t_H]^+) \quad (8)$$

In the middle part, there exists exactly one instance of execution for every $T_i$, and thus we choose the maximum execution time among all paths.

$$f_i^M(t_M) = \frac{t_M}{T_i} \cdot \max_{j \in paths(\tau_i)} \{ARH_i^j(C_i^j)\} \quad (9)$$

The tail part includes the beginning point of an execution but ends at any arbitrary point, so we can calculate the maximum number of bus requests of the tail part similarly to that of the head part.

$$f_i^T(t_T) = \max_{j \in paths(\tau_i)} ARH_i^j(\min\{t_T, C_i^j\}) \quad (10)$$

Eq. (4) assumes that the head (tail) part includes the end (beginning) of an execution. If $t$ is smaller than $T_i$, there might be less than one entire execution during $t$. In this case, the assumption of Eq. (4) is broken, so we need to analyze $BR_i(t)$ in a different way. Once we consider the time interval that starts and ends at an arbitrary point of an execution as shown in Figure 1(b), $BR_i(t)$ can be found as:

$$BR_i(t) = \max_{j \in paths(\tau_i), 0 \le s \le C_i} ARH_i^j(\min\{s+t, C_i^j\}) - ARL_i^j(s) \quad (11)$$

Finally, we can calculate $BR_i(t)$ in the following ways: if $t \ge T_i$, use the result of Eq. (4); and if $t < T_i$, choose the maximum value between Eq. (4) and Eq. (11).

## V. Conclusions and Future Work

We have presented an approach for finding an upper bound on the extra execution of a task due to contention on the interconnection network between processor cores and memory in a COTS-based multicore system. To the best of our knowledge, the problem of analyzing such extra execution time was previously unsolved. Now, we are taking measurements of real programs and using them to build a model of the bus request pattern for each program. When we finish it, we take measurements of the response time of programs to test if our proposed methods for calculating an upper bound on the extra execution time of a task is valid in practice.

This paper is a starting point of considering the effect of bus contention for real-time tasks on multicores, and thus there are many potential research issues. Our future work includes (i) allowing preemptive scheduling, (ii) analyzing switched interconnection networks, (iii) avoiding the pessimism resulting from each task being analyzed individually (that is, task $\tau_1$ has to wait for all bus transactions from $\tau_2$, and task $\tau_2$ has to wait for all bus transactions from $\tau_1$.), and (iv) developing processor scheduling algorithms that facilitate the analysis of the memory bus contention.

## References

[1] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[2] K. D. Bosschere, W. Luk, X. Martorell, N. Navarro, M. OBoyle, D. Pnevmatikatos, A. Ramirez, P. Sainrat, A. Seznec, P. Stenström, and O. Temam, "Challenge 2.2 in high-performance embedded architecture and compilation roadmap," in *Transactions on HiPEAC*, 2007, pp. 5–29.

[3] J. Rosén, A. Andrei, P. Eles, and Z. Peng, "Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip," in *Proc. of IEEE Real-Time Systems Symposium*, 2007, pp. 49–60.

[4] L. Steffens, M. Agarwal, and P. van der Wolf, "Real-time analysis for memory access in media processing SoCs: A practical approach," in *Proc. of Euromicro Conference on Real-Time Systems*, 2008, pp. 255–265.

[5] T. Lundqvist and P. Stenström, "Timing anomalies in dynamically scheduled microprocessors," in *Proc. of IEEE Real-Time Systems Symposium*, 1999, pp. 12–21.

[6] G. Laurent, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," Tech. Rep., 1996.

[7] S. K. Baruah, "A general model for recurring real-time tasks," in *Proc. of IEEE Real-Time Systems Symposium*, 1998, pp. 114–122.

### *Acknowledgements*

This work was partially supported by ARTISTDesign Network of Excellence on Embedded Systems Design, funded by the European Commission under FP7 with contract number ICT-NoE-214373, the Portuguese Science and Technology Foundation (Fundação para Ciência e Tecnologia - FCT), IT R&D program of MKE/KEIT of Korea [2009-KI002090, Development of Technology Base for Trustworthy Computing], National Research Foundation of Korea (2009-0086964), and KAIST ICC, KIDCS, KMCC, and OLEV grants. This paper has been produced partially in a time when we have had general bus-contention discussions with Stefan Petters and Dakshina Dasari.