Technical Report

WCET Analysis Considering Contention on Memory Bus in COTS-Based Multicores

Dakshina Dasari

Vincent Nelis

Björn Andersson

HURRAY-TR-111001

Version:

Date: 10-10-2011

WCET Analysis Considering Contention on Memory Bus in COTS-Based Multicores

Dakshina Dasari, Vincent Nelis, Björn Andersson

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: dndi@isep.ipp.pt, nelis@isep.ipp.pt, baa@isep.ipp.pt

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

Abstract

The usage of COTS-based multicores is becoming widespread in the field of embedded systems. Providing real-time guarantees at design-time is a pre-requisite to deploy real-time systems on these multicores. This necessitates the consideration of the impact of the contention due to shared low-level hardware resources on the Worst-Case Execution Time(WCET) of the tasks. As a step towards this aim, this paper first identifies the different factors that make the WCET analysis a challenging problem in a typical COTS-based multicore system.Then, we propose and prove, a mathematically correct method to determine tight upper bounds on the WCET of the tasks, when they are co-scheduled on different cores.

WCET Analysis Considering Contention on Memory Bus in COTS-Based Multicores

Dakshina Dasari*, Vincent Nelis* and Björn Andersson†*

*CISTER-ISEP Research Centre, Polytechnic Institute of Porto, Portugal
†Software Engineering Institute, Carnegie Mellon University, USA
{dndi, nelis, baa}@isep.ipp.pt, baandersson@sei.cmu.edu

***Abstract*—The usage of COTS-based multicores is becoming widespread in the field of embedded systems. Providing real-time guarantees at design-time is a pre-requisite to deploy real-time systems on these multicores. This necessitates the consideration of the impact of the contention due to shared low-level hardware resources on the Worst-Case Execution Time (WCET) of the tasks. As a step towards this aim, this paper first identifies the different factors that make the WCET analysis a challenging problem in a typical COTS-based multicore system. Then, we propose and prove, a mathematically correct method to determine tight upper bounds on the WCET of the tasks, when they are co-scheduled on different cores.**

I. Introduction

Multicores developed using Commercially available Off-The-Shelf (COTS) components have become an integral choice in the design of embedded systems. Unfortunately, COTS-based multicores are designed to increase the average-case performance and not towards timing predictability or timing compositionality. This makes the application of Worst-Case Execution-Time (WCET) analysis, a key phase in designing real-time systems very challenging as the system design becomes more complex. Another drawback of COTS-based systems is that most of the key underlying protocols, set-up parameters, and implementation details are not documented; either because vendors keep them confidential or the related documents merely omit to specify them. As a consequence, system analysts in the research community either make some generalized assumptions which lead to non-tight WCET estimates, or they base their analysis on over-simplified models that do not reflect the underlying architecture on which the end-application is eventually deployed. Until the industry does not change its current trend in building systems towards *predictable* systems, there is a strong need to fully analyze the current available COTS-based systems and develop models that efficiently capture their runtime behavior.

Apart from the unavailability of the required information, the analysis process is further complicated by the existence of low-level hardware resources shared between processor cores, such as memory for instance. These shared resources give rise to contention between the tasks (running on different cores) eager to access them. An implication of this contention is that the WCET of a task scheduled on a multicore system is not an inherent property of the task itself. Rather, its WCET is impacted by the resource accesses issued from the tasks co-scheduled on the other cores. In a typical COTS-based architecture, all the cores access the main memory via a shared memory bus called "Front-Side-Bus" (FSB). Because this FSB can get saturated, it can cause the cores to stall while waiting for requests to be served, thereby generating a non-negligible increase in the WCET of the tasks running on them. Most of the WCET techniques developed for uniprocessors [1] do not consider the impact of shared low-level hardware resources and hence they cannot be applied to typical COTS-based multicore systems. However, the problem of analyzing the impact of shared hardware resource on the WCET of the tasks is of significant importance, and among the most interesting studies in this research field, one can cite [2], [3], [4], [5]. Unfortunately, all these approaches are not applicable to typical COTS-based systems, because of the assumptions they make.

As a first contribution, we provide a description of a typical COTS-based system, in which we identify the key unknown parameters that prevent the system designers from deriving an accurate timing analysis of memory transactions. That is, we clearly draw the boundary between the parameters that can be accurately computed (or simply found in the literature), those that can merely be estimated through experiments, and those marked as vendor-proprietary. Secondly, after estimating an upper-bound on the time of a memory transaction, we provide a WCET analysis for each task in the system, considering that they interfere with each other while accessing the shared FSB.

II. Identification of the known/unknown parameters in a typical COTS-based system

A. Architecture and components overview

This section gives a brief overview of a typical COTS-based multicore system and highlights the sources of non-determinism in the off-processor-chip boundaries, focusing on the FSB and the memory controller. The sources of non-determinism in the processors (e.g pipelines, branch-prediction units, etc.) are not described here.

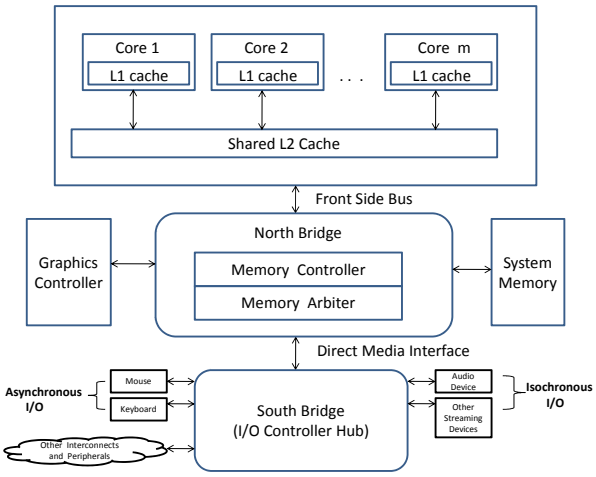

Fig. 1. A typical COTS-based multicores architecture.

B. A typical COTS-based multicore system

A typical COTS-based multicore system is illustrated in Figure 1. It comprises a multicore chip, in which each core has a Level-1 cache and the cores (may) share the Level-2 cache. To focus on the impact of bus-contention, this paper considers multicores with either private caches or assumes that shared cache if present, is disabled. The multicore chip is connected to the North-Bridge (NB) via the FSB. The NB typically handles communications among the CPUs, the RAM, the PCI Express (or AGP) video cards, and the South-Bridge (SB). It directly interacts with the shared system-memory and consists of a memory controller and a memory arbiter. Generally, a graphics controller is connected to the NB (or is sometimes integrated into the NB depending on the chipset design). The SB, often referred to as the I/O Controller Hub, handles communication with the peripherals (such as the hard-disk, keyboard, printer, etc.) over a variety of buses (like PCI and PCI express). The peripherals can be connected in various ways depending on the chipset design.

As seen in Figure 1, the memory is shared over the NB between multiple entities, which we shall henceforth refer to as agents. The main agents that access the system-memory are the multicore chip, the graphics controller and the SB unit.

C. Upper-bounds on memory transactions

In the typical architecture presented above, there are two main levels of contention before a request issued from a core can access the shared memory. First, the requests from the different cores contend for access to the shared FSB and then, once the requests have been brought to the memory controller in the NB, they may still be delayed from being serviced because of contention from other agents like the SB and the graphics controller. Here, we are interested in *the longest time* $\widehat{\mathrm{TR}}$ *needed to serve a memory request from a core*. $\widehat{\mathrm{TR}}$ can be expressed as the sum of (i) the time that a request spends on each intermediate component and (ii) the time needed to transmit the request between components. $\widehat{\mathrm{TR}}$ can be formally expressed as follows:

$$\begin{aligned}\widehat{\mathrm{TR}} &= t_{\mathrm{fsb}} + t_{\mathrm{fsb}\to\mathrm{nb}} + t_{\mathrm{nb}} + t_{\mathrm{nb}\to\mathrm{mem}} + t_{\mathrm{mem}} \\ &\quad + t_{\mathrm{mem}\to\mathrm{nb}} + t_{\mathrm{nb}\to\mathrm{fsb}} \end{aligned} \quad (1)$$

where t_{fsb} is the maximum time to get access to the FSB; $t_{\mathrm{fsb}\to\mathrm{nb}}$ ($t_{\mathrm{nb}\to\mathrm{fsb}}$) denotes the maximum time to transmit the request to (from) the NB over the FSB; t_{nb} is the maximum time that the service of a request is delayed, due to contention in the NB; $t_{\mathrm{nb}\to\mathrm{mem}}$ denotes the maximum time to transmit the request from the NB to the memory, and vice-versa regarding $t_{\mathrm{mem}\to\mathrm{nb}}$, and t_{mem} is the maximum memory access time (needed to load/store the requested data). The quantities $t_{\mathrm{fsb}\to\mathrm{nb}}$, $t_{\mathrm{nb}\to\mathrm{fsb}}$, $t_{\mathrm{mem}\to\mathrm{nb}}$ and $t_{\mathrm{nb}\to\mathrm{mem}}$ are functions of the speed and width of the bus (the FSB for $t_{\mathrm{fsb}\to\mathrm{nb}}$ and $t_{\mathrm{nb}\to\mathrm{fsb}}$, and the North-Bridge-to-memory bus for $t_{\mathrm{mem}\to\mathrm{nb}}$ and $t_{\mathrm{nb}\to\mathrm{mem}}$), as well as the length of the transmitted information (the maximum length of a request for $t_{\mathrm{fsb}\to\mathrm{nb}}$ and $t_{\mathrm{nb}\to\mathrm{fsb}}$, and the length of the returned data for $t_{\mathrm{mem}\to\mathrm{nb}}$ and $t_{\mathrm{nb}\to\mathrm{mem}}$). This information is usually documented, enabling us to accurately compute these four quantities. Concerning t_{fsb}, t_{nb} and t_{mem}, additional information is required.

Estimation of t_{fsb} : The contention over the FSB is usually resolved based on a simple Round-Robin (RR) arbitration mechanism (See [6] for more details), in which all the cores are treated equally. The order in which the cores acquire the ownership of the bus is fixed apriori. Only one core can be the owner of the bus at a time and only the owner of the bus can transmit information. If m denotes the number of cores competing for the FSB, and $t_{\mathrm{own\_switch}}$ is the maximum number of system clock cycles to pass the bus-ownership from one core to another, then t_{fsb} can be computed as $t_{\mathrm{fsb}} \stackrel{\mathrm{def}}{=} (m-1) \times (t_{\mathrm{fsb}\to\mathrm{nb}} + t_{\mathrm{own\_switch}})$. As the required information is provided, t_{fsb} can be determined at design time.

Estimation of t_{nb} : There exist many inter-agent arbitration policies which can be used by the memory-arbiter to determine the order of servicing the memory requests. We will refer to the arbitration scheme proposed in [7]. This arbitration scheme tries to balance the *quality of service* (QoS) requirements of streaming devices with the *low-latency* requirements of memory requests from the CPU. This mechanism uses the concept of "schedule periods" which consists of a fixed number of system-clock cycles[1]. In the particular schedule-period presented in Table I, requests issued from agents are categorized by the nature of their requests (isochronous, asynchronous or maintenance) and each type is allotted a minimum number of service slots.

Maintenance requests encompass refresh requests, current calibration (ICal) and temperature calibration (Tcal) requests issued by the system-memory (DRAM). DRAM refresh refers to the operation which cycles through a DRAM, reading each row and writing it back again in order to compensate for the gradual leakage of charge from the capacitors that store the data. These maintenance requests are typically not a part of

[1]For example, in a system with a system-clock-frequency of 100 MHz, each schedule-period may span 128 cycles or 1.28 microseconds.

Resource Block	Service Slots in Cycles
Maintenance(Refresh)	X
Maintenance(ICal)	Y
Maintenance(TCal)	Z=42
Display(Isochronous)	48
CPU Requests	30
Total	**128**

TABLE I
A TYPICAL SCHEDULE-PERIOD FROM [7].

each schedule period [2], but must be serviced at the highest priority whenever they occur and cannot be preempted by other requests. Also, the number of cycles required to complete every maintenance requests can vary from one DRAM to another.

Isochronous (or periodic) requests are issued by streaming devices which need bandwidth guarantees. For example, a graphics controller typically needs f frames per second for acceptable quality of rendering, where f is technology dependent. Requests issued by the cores or other interactive devices (like keyboard for instance) are *asynchronous*, i.e., the request pattern of such components/devices is typically non-uniform and latency-sensitive.

Memory controllers designed for providing a certain level of QoS to all the agents segregate the requests into different groups. Each request type is pre-assigned certain service slots in every schedule period. As seen in Table I, the highest priority is given to the maintenance requests, followed by isochronous and finally asynchronous requests. If there are no requests of a given type, then all the remaining service slots assigned to that type are distributed amongst the lower-priority request types [7]. This ensures guaranteed service times, while preventing the memory controller from being idle while there are other requests waiting to be served. The extra time t_{nb} due to contention for memory access in the NB depends on the inter-agent arbitration policy in the NB. Considering the policy described here, a request may be serviced directly (if there are no higher-priority pending requests) or it may be serviced at the end of the schedule period (or even in a further scheduler period if the number of service-slots is insufficient to serve it, or if the request requires multiple schedule periods to be serviced). In short, the time spent by each request in the NB is variable due to the flexible but non-predictable arbitration mechanism. None of the parameters specified in Table I are specified in the system manuals and hence determination of t_{nb} is very difficult.

Estimation of t_{mem} : A memory transaction is generally processed by first mapping each memory request to a bank, row and column number and issuing commands to the memory module. After selecting the right bank, the given row is activated (or "opened"). The selected columns are activated and data is read from (or written to) the selected columns with a latency equal to the Column Access Strobe (CAS) Latency. The row is then pre-charged (or "closed") before the next access. To exploit the spatial locality of requests, the overhead of pre-charging and reactivating the same row again can be avoided by keeping the row open until the next request [8]. This policy always lets the current row in the open state after a transaction. Therefore, if the subsequent transaction requests a word which is in the opened row, then this second transaction is served with a latency of only CAS cycles. Otherwise, this second transaction will bear the additional cost of closing the current row and opening the required one. The memory latency can thus be different for each request, depending on the transaction-re-ordering mechanisms present in memory controllers. This adds to the non-determinism during the computation of the overall time to serve a memory request, because the time to access memory constitutes a significant portion of that total time. The memory access time for each request is variable and is influenced by the choice of the memory-access-scheduling policies employed by the underlying memory-subsystem. All this information is generally undocumented and hence, obtaining an accurate estimate of the memory latency is non-trivial.

Summary: Critical information required for computing tight bounds on the time to serve a request is undocumented in COTS-based systems. To re-iterate, information necessary to compute t_{nb}, such as the exact arbitration policy in the memory controller (which includes pre-allocated service time slots assigned to different types of requests) is not specified and cannot be easily obtained. Information necessary to compute t_{mem}, such as the exact transaction scheduling and re-ordering policy employed by the memory controller, is difficult to obtain. The value of these key parameters is not documented in the specifications documents and obtaining an accurate $\widehat{\mathrm{TR}}$ by adding up the maximum time spent in each intermediate component is therefore very difficult. Instead we calculate a new upper bound $\mathrm{TR}^{\mathrm{iso}}$ on the time to complete an entire bus transaction. This is obtained by performing end-to-end measurements and then recording the maximum observed time to serve a memory request. In the next section, we analyze a simple model considering a round-robin bus arbitration algorithm and compute the WCET using this upper bound $\mathrm{TR}^{\mathrm{iso}}$.

III. SYSTEM MODEL AND NOTATIONS

The hardware is composed of a set of m processor cores denoted by $\pi_1, \pi_2, \ldots, \pi_m$ and we assume that (i) no cache memory is shared between them and (ii) all of them share a Front-Side-Bus to access to the shared main memory. We assume that the application τ is composed of n tasks, in which every task τ_i is characterized by a WCET $\mathrm{C}_{\mathrm{i}}^{\mathrm{iso}}$ which denotes an upper bound on the execution time of task τ_i when it executes in *isolation*, i.e., with no contention on the memory bus. In that context, $\mathrm{C}_{\mathrm{i}}^{\mathrm{iso}}$ can be computed by well-known techniques in WCET analysis (see [1] for a state of the art). In this paper, we are interested in finding $\mathrm{C}_{\mathrm{i}}^{\mathrm{mix}}$ which denotes an upper bound on the execution time when τ_i executes *with* contention on the memory bus, i.e., assuming that other tasks

[2]For example, DRAM refresh requests may occur once every 7.8 microseconds [7],or once every 6 schedule periods for a schedule period as in Table I.

are running on the other cores. Clearly, the value of $\mathrm{C_i^{mix}}$ is not an inherent property of τ_i, but is co-runner dependent and depends on the memory request pattern of the other tasks scheduled to run during its execution. We also define a Per-Core-Request-Estimator function $\mathrm{PCRE}_j(t)$ that returns the maximum number of requests generated by the set of tasks assigned to core π_j in any time interval of duration t. The algorithm that computes $\mathrm{PCRE}_j(t)$ will be presented in future work.

We consider a partitioned scheme of task assignment in which tasks are assigned to cores before run-time and are not allowed to migrate from one core to another. We assume a non-preemptive scheduler and hence do not deal with cache related preemption overheads and task-switching overheads. Also, each task assigned to a given core releases its first job at time zero and every task runs to its expected WCET. Whenever a task completes earlier than its WCET (say on CPU π_j), the scheduler idles the core π_j up to the theoretical WCET of the task (the scheduler is thus non work-conserving). This assumption is made to ensure that the number of bus-requests within a time window, computed at design time, is not higher at run-time due to early executions of tasks.

IV. Determination of $\mathrm{C_i^{mix}}$

The value of $\mathrm{C_i^{mix}}$ is clearly dependent on the bus arbitration algorithm and the request pattern of the tasks co-scheduled on other cores. For each task τ_i, we denote by $\mathrm{RQST}_i(\mathrm{C_i^{iso}})$ a function that returns the maximum number of requests that τ_i can generate if it executes non-preemptively in isolation for its WCET $\mathrm{C_i^{iso}}$. Given a Round-Robin scheduling bus arbitration mechanism, each request generated by a task τ_i can be blocked by $(m-1)$ requests issued from the tasks running on the other cores. If $\mathrm{TR^{iso}}$ denotes the maximum time to serve each request when a task runs in isolation, then an upper-bound $\mathrm{C_i^{mix}}$ on the WCET of τ_i considering bus contention is given by

$$\mathrm{C_i^{mix}} = \mathrm{C_i^{iso}} + \mathrm{RQST}_i(\mathrm{C_i^{iso}}) \times (m-1) \times \mathrm{TR^{iso}} \qquad (2)$$

However, Expression 2 is an overly pessimistic bound, because it does not consider the case in which the tasks (if any) running on the other cores (i.e., excluding the core assigned to τ_i) may not be generating requests. Below, we tackle this over-pessimism and we provide a tighter upper-bound. Let $\bar{\tau}_\pi(i)$ denote the set of tasks that are not assigned to the same CPU as τ_i.

Lemma 1: Considering that a task τ_i is executing *with* contention on the memory bus, an upper bound $\mathrm{C_i^{mix}}$ on its execution time is given by the first fixed point (i.e., $C_i^k = C_i^{k-1}$) of the following iterative process (the proof is omitted here due to the space limitation). The initialization step is $C_i^0 \leftarrow \mathrm{C_i^{iso}}$ and

$$\begin{array}{lcl} \mathrm{iqlen}_i^0 & \leftarrow & \mathrm{RQST}_i(C_i^0) \times (m-1) \\ \mathrm{xrqst}_i^0 & \leftarrow & \sum_{\tau_j \in \bar{\tau}_\pi(i)} \mathrm{PCRE}_j(C_i^0) \\ \mathrm{brqst}_i^0 & \leftarrow & \min(\mathrm{xrqst}_i^0, \mathrm{iqlen}_i^0) \end{array}$$

and the iteration step is:

$$\begin{array}{lcl} C_i^k & \leftarrow & C_i^{k-1} + \mathrm{brqst}_i^{k-1} \times \mathrm{TR^{iso}} \\ \mathrm{iqlen}_i^k & \leftarrow & \mathrm{iqlen}_i^{k-1} - \mathrm{brqst}_i^{k-1} \\ \mathrm{xrqst}_i^k & \leftarrow & \sum_{\tau_j \in \bar{\tau}_\pi(i)} \left(\mathrm{PCRE}_j(C_i^k) - \mathrm{PCRE}_j(C_i^{k-1})\right) \\ \mathrm{brqst}_i^k & \leftarrow & \min(\mathrm{xrqst}_i^k, \mathrm{iqlen}_i^k) \end{array}$$

V. Conclusions

The use of COTS components in real-time embedded systems is challenging as the WCET analysis gets complicated due to the presence of shared low-level hardware resources. The presence of such shared resources leads to contention amongst the cores, leading to a non-negligible increase in the WCET of the tasks. In this paper, we firstly identified the different factors that make the WCET analysis a challenging problem in a typical COTS-based multicore system. Then, we proposed an iterative process that computes an upper-bound on the WCET of the tasks, when they are co-scheduled on different cores. Future work involves experimental validation of the theory, carrying out further optimizations to compute tighter WCET estimates and extending the current theory to multicores with shared caches.

Acknowledgement

This work was supported by the REHEAT project (ref: FCOMP-01-0124-FEDER-010045) and the the RePoMuC project (ref: FCOMP-01-0124-FEDER-015050), funded by FEDER funds through COMPETE (POFC - Operational Programme 'Thematic Factors of Competitiveness) and by National Funds (PT) through FCT - Portuguese Foundation for Science and Technology and the RECOMP project, funded by also through the FCT, under grant ref. ARTEMIS/0202/2009, as well as by the ARTEMIS Joint Undertaking, under grant agreement number 100202.

References

[1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. B. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution time problem - overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, 2008.

[2] 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 *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, 2007, pp. 49–60.

[3] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling shared cache and bus in multicores for timing analysis," in *Proceedings of the 13th International Workshop on Software Compilers for Embedded Systems*, 2010, pp. 6:1–6:10.

[4] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo, "Timing analysis for resource access interference on adaptive resource arbiters," in *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011.

[5] S. Schliecker, M. Negrean, and R. Ernst, "Bounding the shared resource load for the performance analysis of multiprocessor systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 759–764.

[6] T. Shanley, *Pentium Pro and Pentium II system architecture (2. ed.)*. Addison-Wesley-Longman, 1998.

[7] S. Pawlowski and B. Baxter, "Apparatus for memory resource arbitration based on dedicated time slot allocation," U.S. Patent 6 363 461, 2002.

[8] B. Jacob, N. G. Spencer, and D. Wang, *Memory Systems Cache, DRAM, Disk*. Morgan Kaufmann, 2007, pp. 497–520.