Predictable Bus Arbitration Schemes for Heterogeneous Time-Critical Workloads Running on Multicore Processors

Roman Bourgade, Christine Rochange, Pascal Sainrat
Institut de Recherche en Informatique de Toulouse
University of Toulouse, France
{bourgade,rochange,sainrat}@irit.fr

Abstract

Multi-core architectures are now considered as possible candidates to implement future time-critical embedded systems. The challenge is to make the worst-case execution time (WCET) of each task predictable. In this paper, we investigate bus arbitration schemes with upper-bounded bus latencies. We focus on heterogeneous workloads in which tasks exhibit distinct requirements in terms of bandwidth. The proposed schemes perform a two-level arbitration: the cores are organized into groups and all the cores in the same group benefit from the same bandwidth. Different algorithms are considered to share the bus slots among the groups. Experimental results (WCET estimates) show an improved global WCET compared to usual round-robin schemes. This will enhance the schedulability of heterogeneous task sets.

I. Introduction

The constraints of embedded systems in terms of power consumption, thermal dissipation, cost-efficiency and performance can be met by using multicore processors (CMP or chip multiprocessors). On typical medium-size CMPs, the cores share a bus to the highest levels of the memory hierarchy. This contributes to increase memory latencies, due to conflicts in the interconnection (bus). When executing hard real-time tasks, this is a major problem because memory latencies have to be bound to allow a safe estimation of the worst-case execution times (WCETs) of tasks. We believe that the design of a predictable bus arbiter should be done with the goal of keeping the worst-case bus latencies for any task independent of the other tasks. This simplifies the analysis and favors timing composition.

A conservative technique for hard real-time systems is to solve inter-core conflicts to shared resources by using fair policy such as the round-robin protocol [6]. This approach perfectly fits systems running homogeneous workloads. However, in some cases, favoring some highly-demanding tasks to fasten their execution may help in achieving the schedulability of the whole system. Also, when considering parallel applications with inter-task dependencies, it may be desirable to accelerate the tasks on the critical path. This motivates our work that aims at providing several levels of bandwidth while keeping the worst-case latencies computable.

We introduce two-level bus arbiters that offer various levels of service to the cores. This allows allocating the tasks onto the cores that fulfil their needs. The objective is to improve the performance of a task set instead of the performance of individual tasks.

The paper is organized as follows. Section II gives an overview of related work and outlines the problems that motivate our approach. We describe two-level bus arbiters in Section III. Section IV provides experimental results and concluding remarks are given in Section V.

II. Related work

To be eligible in the context of a system supporting hard real-time threads, a bus arbiter must insure that the worst-case latency for a given core to get access the memory hierarchy can be computed. Such real-time-aware protocols exist. As an example, the round-robin policy grants the bus to each core in turn, so that the maximum delay a core can undergo when requesting the bus is a linear function of the number of cores that share the bus. This delay is predictable, does not depend on the tasks running on the others cores and is the same for each core [6].

In Time-Division Multiple Access (TDMA) policies, the allocation of slots to the cores is determined off-line, like in [2] and [8]. However, static WCET analysis of a task that runs on a given core cannot consider any possible alignment of memory accesses with respect to the slots allocated to this core. Its feasibility relies instead on abstracting hardware states which, as far as bus latencies are concerned, would lead to consider overwhelmingly pessimistic values.

Budget schedulers ([7], [1]) allocate a given amount

of bandwidth to each core. The prediction of worst-case latencies within WCET computation requires considering the distance between two requests to the bus by the thread under analysis. This might be possible for some particular applications, e.g. when considering accesses to data for a data stream application, but is far more complex for irregular and dynamic accesses to memory like those required to fill the instruction cache on cache misses.

Recently we proposed the MBBA scheme that aimed at offering distinct bus bandwidths to the cores in a multicore architecture [4]. In this paper, we improve this scheme in a way that makes it possible to specify it formally and to implement it in hardware, and we extend the concept of time-predictable two-level bus arbiters so that different arbitration algorithms can be used. We also report a stronger experimental analysis that investigates all the possible arbiter configurations and task allocations for a small set of benchmarks.

III. Two-level Bus Arbitration for Heterogeneous Workloads

A round-robin bus arbiter serves all the tasks in a fair way. For hard real-time applications, an identical worst-case latency ($N \times L$ where N is the number of cores and L the bus latency) should be assumed when analysing the worst-case execution time of any task [6]. In this paper, we consider heterogeneous workloads that include tasks with various demands to the memory system. A task issues more bus requests than another task either because it experiences a larger amount of cache misses or simply because it has a longer execution time.

We introduce *two-level* bus arbiters based on *groups* of cores: in the first level, the arbiter allocates a slot to one group; in the second level, one core in the group is selected and granted the bus. This is illustrated in Figure 1, considering two groups of two cores each: solid lines show *request* signals and dashed lines show *grant* signals. The arbitration policies at each of the two levels define the bus latency seen by each core.

In this paper, we focus on time-critical systems: only time-predictable arbitration strategies are considered. In the first level, we consider either the round-robin (RR) approach [6] or the GL (*Geometric Latencies*) algorithm described in Section III-B1: it allocates different partitions of the bus bandwidth to the cores with power-of-two worst-case bus latencies. In the second level (to arbitrate between few cores in a group), we consider the RR scheme. This makes two arbiters: GRR (*Group Round Robin*) and GGL (*Geometric Group Latencies*).

Each scheme must be configured: each core must be assigned to one group (in this paper we consider up to 3 groups). For a given first-level algorithm, the number of cores in a group determines the worst-case latency seen by each of them. Let L_{G_i} be the worst-case bus latency for group G_i, N_i be the number of cores in G_i and N be the number of groups. If the cores in a group

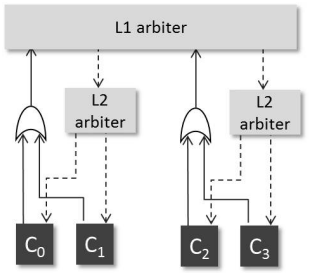

Fig. 1. A two-level bus arbiter

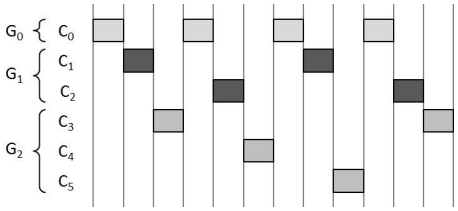

Fig. 2. Example GRR schedule

are arbitrated with a RR protocol, each core sees a worst-case bus latency equal to:

$$\forall i \in [0..N-1], \forall j \in [0..N_i - 1], \quad L_{C_{j \in G_i}} = N_i \times L_{G_i} \tag{1}$$

A. Group Round Robin (GRR) protocol

Figure 2 illustrates the GRR protocol considering 6 cores partitioned into 3 groups and assuming each core permanently issues requests to the bus. Each group gets the bus every third cycle and its cores share the granted slots in a round-robin fashion.

Let L be the bus latency for any request. For the GRR arbiter with N groups and N_i cores in group G_i, the worst-case bus latency seen by a group is given by:

$$\forall i \in [0..N-1], L_{G_i} = N \times L \tag{2}$$

From Equation 1, we get the worst-case latency seen by a core:

$$\forall i \in [0..N-1], \forall j \in [0..N_i - 1], \quad L_{C_{j \in G_i}} = N_i \times (N \times L) \tag{3}$$

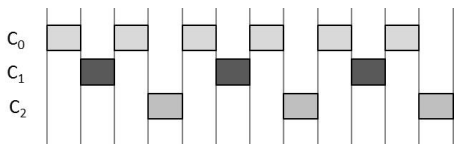

Fig. 3. Example GL schedule

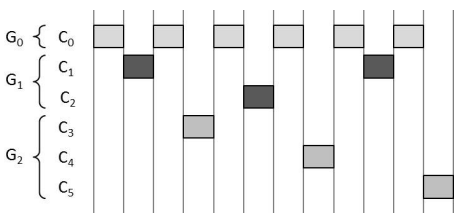

Fig. 4. Example GGL schedule

B. Geometric Group Latencies

The GGL protocol is based on the GL algorithm used to select groups in the first level. In the following, we first explain the GL algorithm before analysing the latencies for GGL.

1) Geometric Latencies: The main principle of the GL single-level arbiter is to enforce bus latencies that grow as a geometric series: core C_0 sees a latency of 2 ($\times L$), core C_1 a latency of 4, core C_2 a latency of 8, etc. (the two last cores have the same latency). The behavior of the GL arbiter is shown in Figure 3.

Formally, the GL arbiter can be specified as follows. The control signals involved in an hardware implementation of the protocol include r_i that indicate (when set) that core C_i has at least a pending request and g_i is set for the core that is granted the bus and cleared for all the other cores. In addition, state variable p_i is set when core C_i should be given priority on the next slot left by the lower-range cores (C_j with $j < i$). In the following, r_i, g_i and p_i are considered as boolean variables.

The following equations formally specify the GL protocol:

$$g_i^t = r_i^t.p_i^t.\prod_{j=0}^{i-1}\overline{p_j^t} \tag{4}$$

$$\begin{cases} p_i^{t+1} = p_i^t \bigoplus \prod_{j=0}^{i-1}\overline{p_j^t} & \forall i \neq N-1 \\ p_{N-1}^{t+1} = \overline{p_{N-2}^{t+1}} \end{cases} \tag{5}$$

Equation 4 tells that the bus is allocated to core C_i at cycle t if this core has requested the bus, it has priority over higher-range cores and no lower-range core has priority over it. Equation 5 indicates that core C_i will has priority over higher-range cores at cycle $t+1$ if (a) it has this priority at cycle t but cannot get the bus because a lower-range core has priority over it; or (2) core C_i has not priority over higher-range cores but no lower-range core has priority over it. Last core, C_{N-1} is served alternately with core C_{N-2}.

In addition to providing a formal specification of the protocol, these equations give insight into how it could be implemented in hardware as a Mealy machine.

From these equations, we get the expression of the worst-case latency for a core (proof is in [5]):

$$\begin{cases} L_{C_i} = 2^{i+1} \times L & \forall i < n-1 \\ L_{C_{N-1}} = 2^{N-1} \times L \end{cases} \tag{6}$$

2) Geometric Group Latencies: Figure 4 shows an example behavior of the GGL protocol with 6 cores partitioned into 3 groups that permanently issue requests to the bus. The core in G_0 gets every second bus slot. Group G_2 is granted every fourth slot and these slots are fairly shared by the three cores in this group.

Considering N groups, the worst-case latency for a group is given by Equation 6 and the worst-case latency for a core C_j in group G_i (with N_i cores: $j \in [0..N_i-1]$) is:

$$\begin{cases} L_{C_j \in G_i} = (N_i.2^{i+1}) \times L \quad for\ i \in [0..N-2] \\ L_{C_j \in G_{N-1}} = (N_{N-1}.2^{N-1}) \times L \end{cases} \tag{7}$$

IV. Experimental results

A. Methodology

The protocol presented in this paper has been modeled within OTAWA, a framework dedicated to static WCET analysis [3]. The experiments reported in [5] were carried out considering an 8-core CMP with in-order 2-way superscalar cores implementing the PowerPC ISA, with a private 2 KB, 2-way associative instruction cache (16-byte lines) and no data cache. The bus is 32-bit wide, with a single-cycle latency and the memory worst-case latency for a read or write operation is 5 cycles.

We define the *normalized WCET sensitivity σ_t to the bus latency changing from λ_1 to λ_2* of a task τ in a task set T, as:

$$\forall \tau \in T, \qquad \sigma_t = \frac{WCET_\tau^{\lambda_2} - WCET_\tau^{\lambda_1}}{\sum_{t \in T} WCET_t^{\lambda_1}} \tag{8}$$

where $WCET_t^\lambda$ if the WCET of task t considering latency λ. The normalization of the sensitivity to the sum of all the tasks WCETs makes this measure useful to select the tasks that should benefit of a larger bandwidth to the bus so that the performance of the whole task set is improved. The arbitration schemes described in the following aim at allowing an allocation of the bus slots to the cores that improves the global performance of a task set instead of enforcing fairness.

The WCET sensitivities of our tasks are reported in [5]. They illustrate the heterogeneity of the workload considered in the experiments.

Our objective in this section is to show how two-level arbitration schemes can help in improving the global worst-case performance of a task set. Our reference is a simple round-robin algorithm that provides the same worst-case latency to each task in the set. The two-level arbiters need to be configured to decide which task is allocated to which group. Possible configurations are listed in [5].

For each configuration and every possible allocation of the tasks, we have determined the *maximum WCET* over the task set (if the tasks were threads of a parallel application, the maximum WCET would impact the global WCET of the application) and the *sum* of all the tasks

Configuration			GRR arbitration		GGL arbitration	
N_1	N_2	N_3	best *maximum WCET*	best *sum of WCETs*	best *maximum WCET*	best *sum of WCETs*
8	-	-	*46,021,703*	*181,588,379*	*46,021,703*	*181,588,379*
1	7	-	58,496,999	146,514,523	58,496,999	246,514,523
2	6	-	48,134,578	191,861,771	48,134,578	191,861,771
3	5	-	36,030,829	170,941,235	36,030,829	170,941,235
1	1	6	**71,810,338**	**255,936,614**	**95,486,098**	**328,264,133**
1	2	5	53,581,234	195,898,091	71,131,639	248,212,769
1	3	4	37,865,309	184,518,725	50,244,323	233,040,281
2	1	5	53,581,234	195,898,091	71,131,639	227,136,803
2	2	4	37,561,765	**166,591,427**	49,802,341	181,418,639
2	3	3	**36,296,698**	174,423,776	48,134,578	191,861,771
3	1	4	37,865,309	184,518,725	49,802,341	174,759,233
3	2	3	**36,296,698**	174,423,776	34,808,477	**161,371,013**
4	1	3	37,865,309	184,518,725	**33,738,971**	166,302,383
5	1	2	53,581,234	195,898,091	36,030,829	175,872,605

TABLE I. Best WCET results over the possible configurations (# cycles)

WCETs (that gives insight into the utilization of the cores). This is shown in Table I.

The best results are for the GGL arbiter: for the best task allocation on the GGL-{4-1-3} configuration, the maximum WCET over the task set is improved by 26.7% compared to a simple RR scheme; and the greatest reduction of the sum of the tasks WCETs (-11.1%) is obtained with the best allocation on the GGL-{3-2-3} configuration.

These results show that it is possible to improve the performance of a simple round-robin scheme with only slightly more complex schemes as those proposed in in this paper (GRR and GGL two-level arbiters) while still offering the possibility of determining worst-case latencies that do not depend on the co-scheduled tasks. This property is the key for keeping WCET analysis tractable. As future work, we will investigate joined allocation and scheduling algorithms that will exploit the possibility of our arbitration scheme.

V. Conclusion

Multicore architectures appear to be relevant candidates to implement embedded systems. However, some critical embedded applications are subject to strict timing constraints. Thus, it is necessary to be able to analyze their worst-case execution time to check that deadlines can be met.

In this paper, we have introduced two-level bus arbiters that feature a certain flexibility in the way bus slots are granted to the different cores, while keeping the predictability of the round-robin protocol. The cores are partitioned into groups and groups are guaranteed either a fair latency (GRR scheme) or a geometric (power-of-two) latency (GGL scheme). The cores in a group shared the group bus slots following a round-robin algorithm. The way the arbiter is configured (number of groups and number of cores in each group) determined the range of latencies seen by the cores. This flexibility allows offering a larger bandwidth to the tasks that have the largest requirements so that the worst-case performance of the task set is improved. Experimental results show that the GGL protocol achieves a maximum WCET and a sum of WCETs over our test-case task set respectively improved by 27% and 11%.

Future work will focus on setting up strategies to determine optimal configurations for the GRR and GGL protocols as well as investigating the possibilities of improving the system-level scheduling of a set of tasks considering this kind of bus arbitration.

References

[1] B. Akesson, L. Steffens, E. Strooisma, and K. Goossens. Real-time scheduling using credit-controlled static-priority arbitration. *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '08)*, pages 3–14, 2008.

[2] A. Andrei, P. Eles, Z. Peng, and J. Rosen. Predictable implementation of real-time applications on multiprocessor system-on-chip. *International Conference on VLSI Design*, pages 103–110, 2008.

[3] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat. OTAWA: an open toolbox for adaptive wcet analysis. In *IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS)*, 2010.

[4] R. Bourgade, C. Rochange, M. de Michiel, and P. Sainrat. MBBA: a multi-bandwidth bus arbiter for hard real-time. In *5th Int'l Conference on Embedded and Multimedia Computing (EMC)*, 2010.

[5] R. Bourgade, C. Rochange, and P. Sainrat. Predictable bus arbitration schemes for heterogeneous time-critical workloads running on multicore processors. Technical report, IRIT, 2011.

[6] M. Paolieri, E. Quinones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for WCET analysis of hard real-time multicore systems. *Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09)*, pages 57–68, 2009.

[7] J. Staschulat and M. Bekooij. Dataflow models for shared memory access latency analysis. *Proceedings of the seventh ACM international conference on Embedded software (EMSOFT '09)*, pages 275–284, 2009.

[8] E. Wandeler and L. Thiele. Optimal tdma time slot and cycle length allocation for hard real-time systems. *Proceedings of the 2006 Asia and South Pacific Design Automation Conference (ASP-DAC '06)*, pages 479–484, 2006.