MBBA: a Multi-Bandwidth Bus Arbiter for hard real-time

Roman Bourgade, Christine Rochange, Marianne De Michiel, Pascal Sainrat
Institut de Recherche en Informatique de Toulouse
CNRS – University of Toulouse
HiPEAC European Network of Excellence
{bourgade, rochange, michiel, sainrat}@irit.fr

***Abstract*—Multi-core architectures are being increasingly used in embedded systems as they offer several advantages: improved hardware integration, low thermal dissipation and reduced energy consumption, while they make it possible to improve the computing power. In order to run real-time software on a multi-core architecture, computing the Worst-Case Execution Time of every thread should be achievable. This notably involves bounding memory latencies by employing a predictable bus arbiter. However, state-of-the-art techniques prove to be irrelevant to schedule unbalanced workloads in which some threads require more bus bandwidth than the other ones. This paper proposes a new bus arbitration scheme that ensures that the shared bus latencies can be upper bounded. Compared to other schemes that make the bus latencies predictable, like the Round-Robin protocol, our approach defines several levels of bandwidth to meet requirements that may vary from one thread to another. Experimental results (WCET estimates) show that the worst-case bus latency is noticeably shortened, compared to Round-Robin, for the cores with highest priority that get the largest bandwidth. The relevance of the scheme is shown through an example workload composed of various benchmarks.**

I. Introduction

Even more than desktop computers or servers, embedded systems have strong constraints on power consumption or thermal dissipation. They also must achieve high cost-efficiency as the market is very competitive. Therefore it is becoming more and more difficult to improve their performance by the only means of increasing their clock rate or making their internal architecture more complex.

A solution to address this issue is the use of multicore processors (Chip MultiProcessors or CMP), that can run several tasks at the same time. On a CMP, the cores share some resources, like second or third level caches, buses or memories. In a CMP, the fact that the cores share part of the memory hierarchy contributes to extend memory latencies, due to conflicts. When executing hard real-time tasks, this is a major problem because memory latencies have to be bound to ensure the safeness of Worst-Case Execution Time estimates.

A conservative technique for hard real-time systems is to solve intercore conflicts to shared resources by using fair-sharing policies such as the Round-Robin protocol. It guarantees that the memory access latencies for a particular core remain independent from the memory requests by the other cores. Thus, latencies can be bounded and included in WCET estimates [1].

Another challenging issue in embedded systems is to keep the memory utilization rate as high as possible while all the threads executed on a multicore embedded platform do not have the same behaviour with respect to memory: those that benefit from temporal or spatial locality are not likely to issue as many requests as the other ones.

The focus of this paper is on MBBA, a new protocol for hard real-time bus arbiters. This protocol provides several priority levels to cores that access the shared memory bus and follows a Round-Robin policy between cores with the same priority. Experimental results show that the threads with highest priority benefit from a significant improvement of their WCET, in comparison to Round-Robin scheduling.

The paper is organized as follows. Section II gives an overview of related work and outlines the problems motivating our approach. We describe the proposed solution in Section III, then Section IV provides experimental results. Concluding remarks are given in Section V.

II. Background

Classical scheduling protocols for shared buses in multicores platforms follow different objectives: some of them intend to ensure good average performances, whereas others aim at making access penalties predictable.

A. Non-predictable bus scheduling

A simple priority-based scheduler considers bus requests according to priority levels: it always serves the requesting core that has the highest priority. Figure 1 shows an example of a fixed priority scheduling with four cores and $Prio(C_1) > Prio(C_2) > Prio(C_3) > Prio(C_4)$. Each core can access the bus as soon as a core of higher priority has not requested it. With this protocol, only the highest-priority core has predictable latencies. This is not enough if we consider several concurrent critical threads running on different cores.

Another common scheduling policy in shared bus platforms is the First-Come-First-Serve (FCFS) protocol. Requests are served with regard to their arrival date: the older a request is, the sooner it will be processed. As a consequence, a core that issues frequent requests will be granted access to the bus more often than the other cores, which ensures good overall performance [2] [3]. Some variants, as First-Ready-First-Come-First-Serve (FR-FCFS), also take into account the

spatial locality of accesses: they first serve requests to the active memory row, which prevents the memory controller from loading a new row and reduces the memory access latency for the requests that benefit from row locality [4]. With these schemes, the bus latencies experienced by a critical thread depend on the dynamic behaviour of co-scheduled threads which prevents them from being determined by static analysis.

Fig. 1. Example of fixed priority scheduling with four cores.

B. WCET-aware scheduling

Real-time aware protocols process the requests of every particular core in such a way that the delays due to conflicts with requests from other cores can be upper bounded. As an example, the Round-Robin policy grants the bus to each core in turn. As a consequence, the maximum delay a core can experience when requesting the bus is a function of the number of cores that share the bus, and of the latency of a bus access. This delay is the same for each core, is predictable and does not depend on the tasks running on the others cores [1]. An example is given in Figure 2.

Fig. 2. Example of Round-Robin scheduling with eight cores.

Time-Division Multiple Access (TDMA) policies are also being increasingly used in embedded hard real-time systems. As in Round-Robin, all the cores are served in turn, but a core can be granted several successive bus slots instead of a single one. Slot allocation is determined off-line, like in [5] and [6].

However, as mentioned in [7], it cannot be determined during static analysis whether a bus request will occur during one the granted slots. Then the TDMA policy might not provide the predictability required for WCET computation.

Budget schedulers ([7], [8]) have also been designed to satisfy variable bus bandwidth requirements. However, instead of determining a static bus scheduling like TDMA schemes, they grant a given amount of bandwidth to each core. According to its priority level, and as far as it has not consumed its allocated bus slots budget, a core can be granted the bus as soon as it issues a request. 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 far more complex for irregular and dynamic accesses to memory like those required to fill the instruction cache on cache misses (the instruction cache analysis generally cannot determine the cache behaviour for some of the instruction fetches).

To sum up, bus scheduling policies that have been designed to fulfill performance requirements are not suited for real-time constraints, as they don't allow bounding bus latencies values. Moreover, predictable protocols are generally not conceived to improve the execution times of the threads because they solely focus on processing all the requests in equity. Our motivation is to propose a scheme that is able to combine these two demands.

III. A NEW PROTOCOL FOR SHARED BUSES

A. A general overview

A challenging problem in using multicore processors in real-time systems is the existence of inter-core interferences. They occur when several cores sharing a resource, e.g. a bus, a cache or a memory, try to access it at the same time. A predictable arbitration mechanism is needed. In the following section, we present our new approach to schedule concurrent memory requests on the shared bus in a multicore platform.

The MBBA (Multi-Bandwidth Bus Arbiter) protocol was designed with the two following goals in mind: (i) providing different bus bandwidths to the cores, to meet different requirements (by the way of priority levels); (ii) making the worst-case bus latency predictable for each core. Our protocol enforces that a memory request from a core cannot be delayed longer than a known Upper Bound Delay (UBD), making the WCET analysis of a task independent of the tasks executed by the others cores. In addition, several priority levels are assigned to the cores, each level exhibiting a different UBD (shorter UBD for higher-priority cores).

B. Detailed specification

Each core is assigned one of the n possible priority levels ($n > 0$). All the cores having the same priority belong to the same group: G_i is the set of cores belonging to group with priority i, and contains N_i cores ($i \in [1, n]$, with $i = 1$ the highest priority group). The bus is granted to cores according to the following policy:

- group G_i is favoured over lower-priority groups (all groups G_j, with $j > i$) every $\frac{1}{2^i}^{th}$ slot. For example, group G_1 is granted the bus every second slot in the worst-case (i.e. if requests from other groups are pending). In the same way, assuming G_1 is assigned the bus every second slot, G_2 would have priority over the other groups every other slot, but can access the bus only every second time. As a result, group G_2 is assigned the bus every fourth slot in the worst-case. More generally, group i can get the bus every $\frac{1}{2^i}^{th}$ slot. Proof is given in Appendix.
- the last group, G_n, uses the remaining slots, i.e. every $\frac{1}{2^{n-1}}^{th}$ slot.
- when the bus is granted to group G_i, one core in this group is selected following a Round-Robin policy (if the selected core has not requested the bus, the next core is served).
- if none of the cores in the selected group has requested the bus, grant is given to the lower priority group.

Fig. 3. Example of scheduling with our arbitration protocol.

An example of scheduling with this protocol is given in Figure 3. We consider an eight-core processor with a bus

latency of one cycle. The protocol configuration used is 2/2/4, i.e. there are three levels of priority: two cores belong to group G_1, two others to group G_2, and the four other cores are in group G_3. The bus arbiter schedules accesses as described below:

- at cycle 0, G_1 and G_2 issue a request. G_1 has priority over other groups since it did not access the bus during the last slot.
- at cycle 1, G_2 gets priority. Within this group, core C_3 is the first to be served.
- at cycle 2, G_2 still holds priority since there are not any others requests.
- at cycle 3, G_1 emits two requests. The first core to be served is C_2 since C_1 was served the last time G_1 was granted a bus slot. Then the requests between cycles 4 and 6 are scheduled with regard to the priority levels.
- at cycle 7, several cores request the bus. G_1 is the first to be served since it did not access the bus during the last slot.
- at cycle 8, G_1 was granted the bus during the previous slot, so G_2 gets priority. Within this group, core C_3 is the first to be served.
- at cycle 9, G_1 gets priority back and C_1 is scheduled to the bus.
- at cycle 10, since G_1 and G_2 accessed the bus the last time they were granted to, group G_3 gains priority and is assigned the bus, etc.

C. Upper Bound Delays

Now we have to compute the Upper Bound Delay to access the bus as a function of the number of priority levels and the number of cores in each group.

In a single level configuration, the different cores in G_1 share the bus using a Round-Robin policy. In this case, the worst-case delay to get access to the bus is:

$$UBD_1 = (N_1 - 1).L_{bus}$$

In a two-level configuration, G_1 has not priority over G_2 if it used the bus during the previous slot. So the worst-case delay for G_1 becomes:

$$UBD_1 = (1 + 2(N_1 - 1)).L_{bus}$$

The worst-case delay for G_2 is:

$$UBD_2 = (1 + 2(N_2 - 1)).L_{bus}$$

In a three-level configuration, G_2 can get the bus every fourth cycle in the worst-case, which gives:

$$UBD_2 = (3 + 4(N_2 - 1)).L_{bus}$$

Similarly:

$$UBD_3 = (3 + 4(N_3 - 1)).L_{bus}$$

The general formula for the worst-case delay is:

$$UBD_i = \begin{cases} (\sum_{k=0}^{i-2}(2^k) + 2^{i-1}(N_i - 1)).L_{bus} & \text{if } (n = i) \\ (\sum_{k=0}^{i-1}(2^k) + 2^i(N_i - 1)).L_{bus} & \text{if } (n > i) \end{cases}$$

Formal proof is given in Appendix. Table I shows how UBD_i is computed considering up-to-four-level configurations.

IV. Experimental results

A. Methodology

The protocol presented in this paper has been modeled within OTAWA, a framework dedicated to static WCET analysis [9] that includes:

- a binary code loader and CFG builder
- an instruction cache analyzer based on abstract interpretation techniques [10] [11]
- a timing analyzer that evaluates the worst-case execution time of basic blocks taking into account the target architecture and the results of the instruction cache analysis [12]
- a flow-fact loader that reads flow fact annotations provided by the oRange tool [13]
- a WCET computer that builds an integer linear program according to the IPET method [14]. This program is solved using the lp_solve tool [15].

Hardware parameters

The experiments reported in this paper were carried out considering in-order superscalar cores implementing the PowerPC ISA with the parameters given in Table II. Each core includes a private instruction cache and a perfect (*Always Hit*) data cache. Small instruction caches are considered so that the code of most of the benchmarks does not fit in: 1 KB, 2-way associative with 16-byte lines. The bus latency is one cycle and the memory worst-case latency is 5 cycles.

Fetch stage width	4
Fetch queue size	8
Decode stage width	2
Commit stage width	2
Functional units *(latency)*	
integer ALU *(1 cycle)*	1
fp ALU *(3 cycles)*	1
multiplier *(6 cycles)*	1
divider *(15 cycles)*	1
memory *(2 cycles)*	1

TABLE II
CORE CONFIGURATION.

Benchmarks

We used some benchmarks from the SNU [16] and Malardalen [17] suites. They are listed in Table III.

n	UBD_1	UBD_2	UBD_3	UBD_4
1	$N_1 - 1$	—	—	—
2	$1 + 2(N_1 - 1)$	$1 + 2(N_2 - 1)$	—	—
3	$1 + 2(N_1 - 1)$	$3 + 4(N_2 - 1)$	$3 + 4(N_3 - 1)$	—
4	$1 + 2(N_1 - 1)$	$3 + 4(N_2 - 1)$	$7 + 8(N_3 - 1)$	$7 + 8(N_4 - 1)$

TABLE I
UPPER BOUND DELAY COMPUTATION.

Benchmark	**Function**	**# inst**	**size** (bytes)
`insertsort`	Insertion sort	84	336
`ns`	Search in a multi-dim array	120	480
`matmul`	Matrix product	140	560
`ludcmp`	LU decomposition	476	1904
`jfdctint`	JPEG implementation of the DCT transform	504	2016
`fdct`	Forward Discrete Cosine Transform	640	2560

TABLE III
BENCHMARKS.

B. Performance of the protocol

Table IV shows the impact on the Worst-Case Execution Time of a Round-Robin bus scheduler: the WCET increases linearly with the number of cores. The values given in the table are increase rates compared to the WCET computed considering a single-core configuration.

The increase on the WCET can be large for the benchmarks that do not fit in the instruction cache. As an example, on an 8-core processor with Round-Robin bus scheduling, the WCET of `fdct` is increased by 144.5%, independently of co-scheduled tasks.

In Tables V and VI, we consider an 8-core processor with the MBBA protocol and three bandwidth levels. The impact on the WCET of the bus scheduling is given for each group of cores, as a function of the number of cores belonging to the group. For a given configuration, we must have $N_1 + N_2 + N_3 = 8$.

It can be observed that the WCET increase for a core in G_1 with 4 cores in G_1 is the same as the one observed with 8-core Round-Robin scheduling. This is due to similar Upper Bound Delays in both cases:

$$UBD = 1 + 2(N_1 - 1) = 2.N_1 - 1$$

As a consequence, on an N-core processor, each thread running on a core in G_1 experiences an improvement of its WCET, compared to Round-Robin scheduling, provided $N_1 < \frac{N}{2}$. On the contrary, cores in G_2 and G_3 experience longer UDBs and then higher WCETs compared to Round-Robin scheduling.

C. Example of use of the MBBA protocol

Table VII shows how the overall WCET can be computed for a mixed workload composed of four threads (`matmul`, `fdct`, `jfdctint` and `insertsort`) running on a multi-core. The values given in this table are the WCETs, in number of cycles, computed considering:

- a single-core configuration: in this case, the four threads must be executed in sequence and the overall WCET is the sum of the individual WCETs.
- a 4-core configuration with Round-Robin bus scheduling: the overall WCET is the maximum of the individual WCETs. As expected, it is lower than that computed for the single-core configuration, due to thread parallelism. However, it can be observed that the maximum WCET is for `fdct` and `jfdctint` and that these values are significantly higher than those observed for the single-core processor (respectively +49.1% and +33.1%).
- an 8-core configuration with Round-Robin bus scheduling: in this case, the number of cores exceeds the number of threads belonging to the workload (which is likely to happen in a real-life system). The increase on the individual WCETs and thus on the overall WCET is even more dramatic than with 4 cores. Even if the global WCET is reduced against a sequential execution, the improvement due to parallelization is not as high as expected due to unbalanced bus requirements (the most demanding threads being responsible for poor gain on WCET).
- an 8-core configuration with MBBA bus arbitration and the following assignement of threads to bandwidth-level groups: `fdct`, which has the highest WCET, is assigned the highest priority group (G_1); `jfdctint`, which also exhibits a high WCET, is put into group G_2; the two other threads, that have shorter WCETs belong to group G_3. With these assignments, the overall WCET of our mixed workload is improved by 13.4% against the 4-core Round-Robin configuration and by 40.1% compared to the 8-core Round-Robin architecture.

This simple example shows that the MBBA protocol can improve the WCET of mixed workloads that include threads with unbalanced bus bandwidth requirements, against the common Round-Robin scheme. Moreover, in addition to being helpful for heterogeneous workloads, the improvement on the worst-case performance is even more significant when the threads in the workload do not use all the cores: the cores that run non critical tasks do not contribute to augment bus

# cores	2	4	6	8
fdct	+16.4%	+49.1%	+81.8%	+114.5%
insertsort	+0.2%	+0.7%	+1.2%	+1.7%
jfdctint	+11%	+33.1%	+55.1%	+77.2%
ludcmp	+0.5%	+1.5%	+2.5%	+3.5%
matmul	+0.3%	+0.8%	+1.3%	+1.8%
ns	+0.1%	+0.2%	+0.4%	+0.5%

TABLE IV
IMPACT ON THE WCET OF ROUND-ROBIN BUS SCHEDULING.

N_1	1	2	3	4	5	6	7	8
fdct	+16.4%	+49.1%	+81.8%	+114.5%	+147.1%	+179.8%	+212.5%	+245.2%
insertsort	+0.2%	+0.7%	+1.2%	+1.7%	+2.2%	+2.7%	+3.2%	+3.7%
jfdctint	+11.0%	+33.1%	+55.1%	+77.2%	+99.2%	+121.3%	+143.3%	+165.4%
ludcmp	+0.5%	+1.5%	+2.5%	+3.5%	+4.4%	+5.4%	+6.4%	+7.4%
matmul	+0.3%	+0.8%	+1.3%	+1.8%	+2.3%	+2.9%	+3.4%	+3.9%
ns	+0.1%	+0.2%	+0.4%	+0.5%	+0.6%	+0.8%	+0.9%	+1.1%

TABLE V
WCET INCREASE WITH MBBA SCHEDULING AS A FUNCTION OF THE NUMBER OF CORES IN G_1 WITH $n = 3$.

N_2 **or** N_3	1	2	3	4	5	6	7	8
fdct	+65.4%	+130.8%	+196.2%	+261.6%	+326.9%	+392.3%	+457.7%	+523.1%
insertsort	+1.0%	+2.0%	+3.0%	+4.0%	+4.9%	+5.9%	+6.9%	+7.9%
jfdctint	+44.1%	+88.2%	+132.3%	+176.4%	+220.5%	+264.6%	+308.7%	+352.9%
ludcmp	+2.0%	+3.9%	+5.9%	+7.9%	+9.8%	+11.8%	+13.8%	+15.7%
matmul	+1.0%	+2.1%	+3.1%	+4.2%	+5.2%	+6.3%	+7.3%	+8.3%
ns	+0.3%	+0.6%	+0.9%	+1.1%	+1.4%	+1.7%	+2.0%	+2.3%

TABLE VI
WCET INCREASE WITH MBBA SCHEDULING AS A FUNCTION OF THE NUMBER OF CORES IN G_2 OR G_3 WITH $n = 3$.

Config.	matmul	fdct	jfdctint	insertsort	$WCET$
1-core	4987	7379	6575	3637	$\sum = 22578$
RR 4-core	5026	11001	8750	3664	$\max = 11001$
RR 8-core	5078	15825	11650	3700	$\max = 15825$
MBBA 8-core	5091	8589	9475	3709	$\max = 9475$

TABLE VII
WCET IMPROVEMENT USING MBBA SCHEDULING WITH MIXED BENCHMARKS.

latencies for hard real-time threads.

V. CONCLUSION

Much work has been done these last fifteen years to set up techniques to compute safe Worst-Case Execution Times upper bounds and these techniques are now mature. However, their validity relies on the assumption that the application can run on the target architecture without being impacted by external events that would impair determinism. In a multicore processor, such events could be related to conflicts on the shared bus with tasks that run on another core. As a consequence, it is necessary to implement a bus arbitration scheme that makes it possible to upper bound the bus latencies, in such a way that the WCET of one task can be computed without any knowledge about the tasks that may be executed simultaneously on the multicore.

Several WCET-aware scheduling schemes have been proposed: for some of them, the predictability of the latencies during the WCET computation process is questionable; for other ones, like Round-Robin [1], the predictability is assessed but the algorithm does not fit well unbalanced workloads in which some threads require more bus bandwidth than the other ones.

In this paper, we have introduced MBBA, a new arbitration scheme that features a certain flexibility in the way bus slots are granted to the different cores, while keeping the

predictability of the Round-Robin protocol. In this scheme, several priority levels are defined, corresponding to different guaranteed bus bandwidths, with a Round-Robin scheduling within each level. This flexibility allows offering a larger bandwidth to the threads that have the largest requirements so that their WCET is less impacted by bus sharing.

Experimental results show that the choice of a configuration (number of levels, number of cores in each level) leads to WCETs more or less impacted by the bus sharing. From an example workload, we have shown how the MBBA scheme can reduce the overall WCET of a set of threads executed in parallel when these threads exhibit different bandwidth needs. The protocol also improves the worst-case performance when the number of critical threads is lower than the number of cores: while Round-Robin would consider a worst-case bus latency related to the total number of cores, MBBA considers a latency that only depends on the cores that really execute critical threads.

Future work will include setting up strategies to determine optimal configurations for the MBBA protocol as well as investigating the possibilities of improving the system-level scheduling of a set of tasks considering this kind of bus arbitration.

APPENDIX

A. Slot allocation proof

We want to check that :

- if group G_i is the lowest-priority group ($i = n$), it can get the bus every $\frac{1}{2^{i-1}}^{th}$ slot in the worst-case.
- otherwise ($i < n$), it can get the bus every $\frac{1}{2^i}^{th}$ slot in the worst-case.

To establish the first recurrence, we must prove that the formula holds for G_1.

- if G_1 is the only group ($n = 1$), it can be granted the bus at any time, i.e. every $\frac{1}{2^0}^{th}$ slot.
- if G_1 has to share the bus with others groups ($n > 1$) then it can get the bus every second slot ($\frac{1}{2^1}^{th}$).

The assertions are true for $n = 1$. Let us assume they still hold for G_{i-1}. We want to prove them for G_i. If $n > (i-1)$, G_{i-1} can get the bus every $\frac{1}{2^{i-1}}^{th}$ slot in the worst-case. The amount of slots allocated to group i depends on the value of n:

- if $n = i$: G_i gets as many slots as G_{i-1} since there is no lower-priority group than G_i. It can then access the bus every $\frac{1}{2^{i-1}}^{th}$ slot.
- otherwise, $n > i$: G_i gets twice less slots than G_{i-1}. It can access the bus every $\frac{1}{2^i}^{th}$ slot.

B. Proof for the Upper Bound Delay formula

We want to prove that:

$$UBD_i = \begin{cases} (\sum_{k=0}^{i-2}(2^k) + 2^{i-1}(N_i - 1)).L_{bus} & \text{if } (n = i)\ (1) \\ (\sum_{k=0}^{i-1}(2^k) + 2^i(N_i - 1)).L_{bus} & \text{if } (n > i)\ (2) \end{cases}$$

To establish the first recurrence, we must prove that the formula holds for G_1.

With a single core in the first group ($N_1 = 1$), there are two possibilities:

- G_1 is the only group ($n = 1$): this gives $UBD_1 = 0$, which is obviously correct (no wait).
- otherwise G_1 is not the only group ($n > 1$) and the formula gives $UBD_1 = L_{bus}$: the core cannot use the bus during two slots in a row if other cores have issued some requests (worst-case).

The formula being true with $N_1 = 1$, let us check it for $N_1 > 1$. There are again two possibilities:

- G_1 is the only group ($n = 1$), which gives:
  $$UBD_1 = (N_1 - 1).L_{bus}$$
  This is the delay of Round-Robin scheduling.
- there are several priority levels ($n > 1$): each time G_1 requests for the bus, it can be delayed for one slot by lower-priority cores. Then the Round-Robin delay within the group is doubled. In addition, the bus can be busy when the request is issued: G_1 has to wait for another slot. Finally the formula gives:
  $$UBD_1 = (1 + 2.(N_1 - 1)).L_{bus}$$

Now we have proved the formula for G_1 and we want to extend it to G_i, $i > 1$. Let us assume that it is true for G_{i-1}, and prove it for group G_i:

- if G_i is the lowest-priority group ($n = i$):
  $$UBD_{i-1} = (\sum_{k=0}^{(i-1)-1}(2^k) + 2^i(N_i - 1)).L_{bus}\ (2)$$
  According to the previous proof, G_i can get the bus as many times as G_{i-1}. The first part of the formula is still $\sum_{k=0}^{i-2}(2^k)$.
  G_i is granted the bus every $\frac{1}{2^{i-1}}^{th}$ slot so the maximum delay a core can suffer within G_i is $(2^{i-1}(N_i - 1))$. Eventually:
  $$UBD_i = (\sum_{k=0}^{i-2}(2^k) + 2^{i-1}(N_i - 1)).L_{bus}\ (1)$$
- if G_i is not the lowest-priority group ($n > i$):
  $$UBD_{i-1} = (\sum_{k=0}^{(i-1)-1}(2^k) + 2^i(N_i - 1)).L_{bus}\ (2)$$
  An additional group G_i with $i < n$ has to wait for G_{i-1} before being granted the bus, during 2^{i-1} cycles. The first part of the formula becomes $(\sum_{k=0}^{(i-1)-1}(2^k) + 2^{i-1})$.
  According to the previous proof, G_i is granted the bus every $\frac{1}{2^i}^{th}$ slot so the maximum delay a core can suffer within G_i is $(2^i(N_i - 1))$. Eventually:
  $$UBD_i = (\sum_{k=0}^{i-1}(2^k) + 2^i(N_i - 1)).L_{bus}\ (2)$$

REFERENCES

[1] 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)*, pp. 57–68, 2009.

[2] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007.

[3] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," *Proceedings of the 27th annual international symposium on Computer architecture (ISCA-27)*, pp. 128–138, 2000.

[4] M. K. Vernon and U. Manber, "Distributed round-robin and first-come first-serve protocols and their applications to multiprocessor bus arbitration," *Proceedings of the 15th Annual International Symposium on Computer architecture (ISCA-15)*, pp. 269–279, 1988.

[5] 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*, pp. 103–110, 2008.

[6] 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)*, pp. 479–484, 2006.

[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)*, pp. 275–284, 2009.

[8] 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)*, pp. 3–14, 2008.

[9] H. Cassé and C. Rochange, "OTAWA, Open Tool for Adaptative WCET Analysis," in *Design, Automation and Test in Europe (Poster session "University Booth") (DATE), Nice, 17/04/07-19/04/07*, 2007.

[10] C. Ferdinand, F. Martin, and R. Wilhelm, "Applying compiler techniques to cache behavior prediction," *ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*, 1997.

[11] C. Ballabriga and H. Cassé, "Improving the first-miss computation in set-associative instruction caches," *Euromicro Conference on Real-Time Systmes (ECRTS '08)*, 2008.

[12] C. Rochange and P. Sainrat, "A context-parameterized model for static analysis of execution times," *Transactions on HiPEAC, Springer*, vol. 2, no. 3, 2007.

[13] M. de Michiel, A. Bonenfant, H. Cassé, and P. Sainrat, "Static loop bound analysis of C programs based on flow analysis and abstract interpretation," in *IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2008.

[14] Y.-T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," in *Workshop on Languages, Compilers, and Tools for Real-time Systems*, 1995.

[15] "lp_solve Reference Guide," http://lpsolve.sourceforge.net/.

[16] "Homepage of SNU Real-Time Benchmark Suite," http://archi.snu.ac.kr/realtime/benchmark/.

[17] "Homepage of Malardalen WCET Research Group Benchmark Suite," http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.