Design and Implementation of a Fair Credit-Based Bandwidth Sharing Scheme for Buses

Mladen Slijepcevic[‡,†], Carles Hernandez[†], Jaume Abella[†], Francisco J. Cazorla[†,*]
[‡]Universitat Politecnica de Catalunya (UPC), Spain
[†]Barcelona Supercomputing Center (BSC), Spain
[*]Spanish National Research Council (IIIA-CSIC), Spain

***Abstract*—Fair arbitration in the access to hardware shared resources is fundamental to obtain low worst-case execution time (WCET) estimates in the context of critical real-time systems, for which performance guarantees are essential. Several hardware mechanisms exist for managing arbitration in those resources (buses, memory controllers, etc.). They typically attain fairness in terms of the number of slots each contender (e.g., core) gets granted access to the shared resource. However, those policies may lead to unfair bandwidth allocations for workloads with contenders issuing short requests and contenders issuing long requests. We propose a Credit-Based Arbitration (CBA) mechanism that achieves fairness in the cycles each core is granted access to the resource rather than in the number of granted slots. Furthermore, we implement CBA as part of a LEON3 4-core processor for the Space domain in an FPGA proving the feasibility and good performance characteristics of the design by comparing it against other arbitration schemes.**

I. Introduction

Across domains such as space and automotive, performance demands of critical real-time embedded systems have steadily grown over time. This is due to the increase in the number and complexity of functions carried out by those systems. For instance, autonomous driving and unmanned vehicles in the automotive and avionics domain respectively need orders of magnitude higher performance than that delivered by simple single-core microcontrollers [10]. To respond to these demands, real-time industry is assessing the use of multicores comprising multi-level cache hierarchies and relatively powerful cores. These features, however, challenge functional and timing verification as required in critical real-time embedded systems. For the latter – the focus of this paper – complex hardware designs challenge the mandatory step of deriving worst-case execution time (WCET) bounds required for scheduling real-time tasks.

In particular, multicore contention has been shown to be a key performance limiter for real-time functions when hardware is not designed properly and/or timing analysis is unable to account for its impact reliably and tightly [14]. Given that in these domains, industry has just started using multicores, core counts are in many cases limited to up to four or eight. In this scenario shared buses have been shown to be effective to cope with the bandwidth requirements[1] [21]. Several arbitration policies have been proposed to control the access to the shared bus and other resources. Among those, we find round-robin, FIFO, TDMA, lottery, and random permutations [12], [13], [15], [17]. All those policies have been shown to provide a high degree of fairness across cores in terms of number of requests (slots) granted. However, whenever requests from different cores have different duration, fairness is lost since cores with larger requests enjoy most of the bandwidth to the detriment of cores with shorter requests. For instance, let us assume two cores that are granted access to a resource alternatively, with one of them having 5-cycle requests and the other 45-cycle requests. The first core uses 10% of the bandwidth and the latter 90%, despite receiving the same number of slots.

In this paper we tackle this issue by proposing a credit-based arbitration (CBA) policy that implements a credit-based flow-control mechanism [4], [16] to balance shared resource utilization by tracking how long each contender has used the shared resource. CBA provides each contender with a time budget that is decreased by the amount of time the shared resource is used and recovered slowly later to ensure that no contender (core) overuses the shared resource. In this way, those cores issuing short requests are granted access more often than those issuing long requests, thus achieving bandwidth fairness. Moreover, we implement CBA in a 4-core LEON3 processor for the Space domain in a FPGA prototype, proving that CBA has affordable implementation costs and achieves the performance required. We focus on measurement-based timing analysis since it is the dominant industrial practice in these domains [2], [18], [23]. In particular we show that CBA is compatible with Measurement-Based Probabilistic Timing Analysis (MBPTA) [6], which targets complex processors.

II. Background on Arbitration Policies

Arbitration policies are used to grant access to shared resources when there are several requestors. For multicores with cores connected to L2, memory and/or I/O with buses, fairness is required across different cores to access the bus and so the other hardware shared resources. Many policies exist for that purpose, but only some of them have been shown to be amenable for critical real-time systems where WCET needs to be tightly and reliably estimated for scheduling purposes.

In general, priorities cannot be used for arbitration if all cores need to run real-time tasks [19], since cores with high priority can potentially issue requests uninterruptedly thus preventing the other cores with lower priorities to access the bus. Instead, policies such as FIFO, round-robin, and TDMA ensure that all cores will be granted access to the shared resource eventually and simplify WCET estimation. For a detailed comparison of those policies we refer the interested reader to the work in [12]. In general, FIFO, round-robin and TDMA can be regarded as fair w.r.t. the number of requests, but not w.r.t. the duration of those requests. For instance, let us assume a scenario where all cores issue requests constantly to the shared bus. In the case of FIFO and round-robin, the different cores will access the bus alternatively keeping it fully utilized. In the case of TDMA, time is typically split across cores homogeneously and it is also common using time slots whose duration matches the longest duration of any request [12]. Since the duration of a request is unknown a priori (i.e. whether it will hit/miss in L2, whether it will produce a dirty line eviction in L2, etc.), TDMA typically allows requests to be issued only during the first cycle of the slot for each core. Allowing a request whose duration is unknown to be issued at any other time could prevent

[1]In high core-count multicores with shared L2, a NoC is a better solution as communication means [22].

978-3-9815370-8-6/17/$31.00

requests from other cores being issued at their expected time – negatively impacting the derivation of WCET estimates. Therefore, in our scenario where all cores issue requests constantly, cores will be granted access to the bus alternatively, but leaving the bus idle whenever a request takes less than the maximum latency.

Other policies such as lottery [17] and random permutations [13] have been shown to be also compatible with probabilistic timing analysis and, in particular, with MBPTA. Those policies assign the grant randomly with different constraints with the aim of making the worst case being closer to the average case and thus, improving WCET estimates. In our particular example, with a fully congested bus, in the long run under these policies bus bandwidth is homogeneously shared among cores *in terms of number of requests*, as it is the case for FIFO, round-robin and TDMA.

Overall, existing policies share the common characteristic that under high congestion, they balance the number of slots, i.e. how many times each core is granted access to the bus, however they neglect how long each core uses the bus.

Illustrative Example. We illustrate this phenomena with focus on our target hardware platform, in which we implement of CBA. In particular a 4-core processor deploying a bus to connect cores with the partitioned L2 cache and the memory system and in which requests hold the bus until it is fully served, i.e it is a non-split bus. In this setup, let us assume that the task under analysis issues frequent requests that access the L2 cache with a total turnaround latency of 6 cycles once granted access to the bus. On the other hand, tasks in the other cores are streaming applications issuing constantly read requests to memory that take 28 cycles. Under this setup the bus would be fully saturated by any of the tasks. When consolidating the 4 tasks in the 4 cores so that all of them run simultaneously, any request-fair arbitration policy will lead to a situation where roughly 25% of the requests in the bus belong to each of the cores. Hence, each 6-cycle request of the task under analysis will have to wait for around 28x3=84 cycles to be granted access to the bus. If the task under analysis runs for 10,000 cycles in isolation out of which 6,000 cycles are spent accessing the bus (1,000 requests), its execution time with contention will be easily close to $(10,000 - 6,000) + 1,000 \times (6 + 84) = 94,000$. In other words, in a 4-core processor this task can experience a 9.4x slowdown. The high slowdown breaks the common wisdom that the expected slowdown due to contention would be up to 4x when consolidating in a multicore 4 tasks that fully utilize a shared resource each one in isolation. This is so because inappropriate design choices may lead to performance issues such as those related to short requests competing with long request, as already shown in some processors [9]. Thus, specific arbitration policies that fairly share bandwidth in terms of time rather than in terms of requests are needed. We propose CBA to cover this gap.

Instead, if a cycle-fair arbitration is used, execution time would be $(10,000 - 6,000) + 1,000 \times (6 + 18) = 28,000$, so a 2.8x slowdown. While such slowdown is still high, it is expected when tasks saturating a given resource in isolation are consolidated together in the multicore: given N cores, the slowdown should be at most N times.

III. Credit-Based Arbitration

CBA builds upon three principles. First requests will be eventually granted access regardless of their duration, as long as a maximum duration exists and its latency is known (or can be upperbounded). We define this maximum duration (or its upperbound) as $MaxL$. Second, requests will be eventually granted access regardless of the core they belong. And third, bandwidth is fairly distributed across contenders in terms of cycle counts rather than in terms of request counts.

In practice, this is achieved by allocating each core a given credit (budget) matching $MaxL$. Then, arbitration is performed across all cores with pending requests and an available budget of exactly $MaxL$ cycles. When a request is granted access to the bus, the budget of the corresponding core is decreased by the bus hold time. For instance, this can be implemented decreasing by 1 the budget of the core using the bus. Further, every cycle all cores get their budget increased as shown in Equation 1:

$$Budget_i(t+1) = \min(Budget_i(t) + 1/N, MaxL) \quad (1)$$

where $Budget_i(t)$ stands for the budget of core i at cycle t and N stands for the number of cores. Note that budget saturates at $MaxL$ to prevent the case in which one core spends long time not using the bus and then it tries to hog the bus during a long period. Otherwise, the effective bandwidth enjoyed by one task would depend on the shared resource utilization performed by previously executed tasks in all cores, which could lead to any arbitrary budget imbalance. Instead, with our approach we only need to collect measurements at analysis time making the task under analysis (TuA) start with zero budget. Also note that, although conceptually the budget is increased by a fraction, this can be implemented by multiplying all factors in Equation 1 by N. In that case, when using the bus, the budget should also be decreased by N every cycle instead of by 1.

A. Arbitration Choices

CBA acts as a filter to determine the pending requests that are eligible to be arbitrated: only those whose core has $MaxL$ budget can be arbitrated. Then, any arbitration policy can be applied. For MBPTA, the the particular timing analysis considered in this paper, several arbitration policies have been shown to be compatible: round-robin, lottery, random permutations [13] and TDMA [8].

With CBA, heterogeneous arbitration across different cores, i.e. granting higher bandwidth to one core than others, can be achieved in several ways: (1) by allowing the budget of a given core grow above $MaxL$ or (2) by increasing the budget of all cores (in total) by 1 every cycle but in a heterogeneous way. For instance, in the former case we could allow the budget of core 1 grow to up to $2 \cdot MaxL$, and in the latter case we could make bandwidth grow by $1/2$ for core 1 and by $1/6$ for each of the other 3 cores in a 4-core processor. Each approach has its own pros and cons. For instance, letting the budget grow above $MaxL$ allows requests from one core to be issued back-to-back, which is good for this core but creates some temporal starvation to the others.

B. WCET Estimation

MBPTA relies on measurements probabilistically capturing the worst execution conditions that can arise during system operation. As it has been shown in other works in the context of buses or more complex NoCs [13], [22], for the interconnection network this worst-case operation-time conditions can be reproduced by collecting execution times of the TuA under maximum contention scenarios. In our case this can be done enforcing the following conditions: first, contending cores are assumed to always have a request ready to compete with the TuA, but new requests are created only if the TuA has a request ready; and second, contending requests always take the maximum latency, $MaxL$.

The first condition creates the highest contention since the requests of the TuA always find the maximum number of

TABLE I. SUMMARY OF SIGNALS

	Every cycle	When using bus
$BUDG_i$	$\min(BUDG_i + 1, 228)$	$BUDG_i - 4$

	WCET mode	Operation mode
$COMP_1$	—	—
$COMP_{2,3,4}$	$BUDG_i == 228 \wedge REQ_1 == 1$	1
REQ_1	when request ready	when request ready
$REQ_{2,3,4}$	1	when request ready

contenders ready. Note that by not creating requests when the TuA has no pending requests, contenders are more likely to have all their budget available by the time the TuA generates a new request. In other words, contenders only consume their budget when their requests compete against those of the TuA. The second condition relates to the fact that, when competing, requests from contenders are greedy creating the highest contention as soon as possible. This holds under the premise that the impact of contention in execution time is the same for different requests of the TuA, which is often the case in simple in-order processors such as those used for critical real-time functions [5]. Hence, if at some point contenders do not have enough budget to create contention for a given request of the TuA, it can only be because that budget was used creating contention for an earlier request of the TuA. Finally, as stated before, measurements for the TuA are collected under worst conditions which implies setting its initial budget to zero, thus delaying the most the issuing of the first request of the TuA. By that time all contenders will also have all their budget available to compete with the TuA.

C. Implementation

Assessing implementation complexity and overheads is fundamental for industrial adoption. We have implemented CBA in an FPGA design of a 4-core LEON3 processor used in the Space domain [11]. In particular, we have integrated CBA with random permutations arbitration since it has been shown to be a MBPTA-compliant efficient policy [13].

Our target processor implements a non-split AMBA bus [1]. $MaxL$ is 56 cycles since this is the longest duration of a request (see Section IV). It is worth noting that, despite buses with split transactions have more homogeneous request sizes, the worst-case situation, having very long and very short requests, is possible since atomic operations by definition cannot be split. We have implemented CBA as a part of the AMBA bus arbiter and connected it to the APRANDBANK module that delivers random bits every cycle for random choices of the random permutations arbitration [3], [11]. Arbitration decisions are performed in one clock cycle.

Next we describe the signals used, which are conveniently summarized in Table I. For each core the arbiter has an 8-bit budget counter ($BUDG_i$) that saturates at 228 (56x4). Every cycle all $BUDG_i$ saturated counters are incremented by 1. Also, every cycle the core using the bus (if any) gets its $BUDG_i$ counter decreased by 4. Our implementation allows configuring the platform as either *WCET estimation* mode (for analysis) or *operation* mode. During *WCET estimation* mode the request (REQ_i) signals of cores 2, 3 and 4 are always set. Each of those cores has also a compete ($COMP_i$) bit. The $COMP_i$ bit is set when $BUDG_i$ is 228 (so $MaxL$) and REQ_1 is set, thus meaning that the TuA, which runs in core 1, has a request ready. $COMP_i$ is reset whenever core i is granted access to the bus. Also, during *WCET estimation* mode cores 2, 3 and 4 keep the bus busy during 56 cycles when granted access.

During *operation* mode, REQ_i signals are only activated when the corresponding core has a request ready and $COMP_i$ signals are always set, thus not making requests wait for requests in core 1.

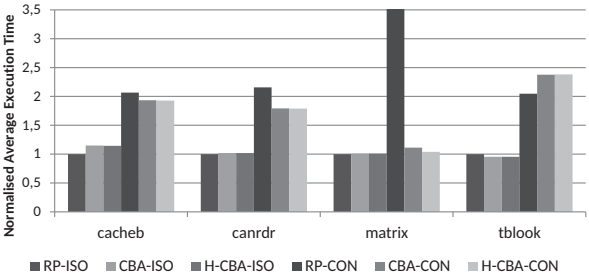

Fig. 1. Slowdown with and without CBA for EEMBC on the FPGA multicore. ISO stands for isolation and CON for maximum contention.

IV. EVALUATION

In this section we present the evaluation framework and some results comparing CBA vs no-CBA in the context of a bus implementing random permutations arbitration and using MBPTA to derive the WCET.

A. Experimental Framework

We use a 4-core processor setup with each core having private L1 data and instruction caches. The L2 cache is shared among cores. Each core implements a pipelined in-order SparcV8 LEON3 processor. Caches implement random-placement and random-replacement caches to facilitate MBPTA application [11]. Data L1 cache implements write-through policy, whereas L2 cache implements write-back policy. Cores are connected to a shared (partitioned) L2 cache through an AMBA bus [1]. L2 cache is connected to a memory controller that serves as bridge to DRAM DDR2 memory. We use random permutations policy to take arbitration decisions [13]. This architecture has been prototyped in an ALTERA TerasIC DE4 FPGA.

Bus transactions take between 5 cycles for L2 read cache hit and 56 cycles. Memory latency is 28 cycles and the longest requests may produce 2 memory accesses, e.g. atomic operations produce a read and a write operation and L2 cache misses evicting a dirty line produce one access to write dirty data back to memory and another to fetch requested data.

Results on the FPGA have been obtained for the EEMBC Autobench suite [20], that reflects current real-world demand of some critical real-time functionalities.

B. Results

We run EEMBC benchmarks under 3 different bus configurations, all based on random permutations as arbitration policy. Those configurations include the baseline random-permutation bus (RP), the credit-based one (CBA), and a heterogeneous implementation of the CBA (H-CBA) where the TuA gets 50% of the bandwidth. The TuA is run under each of these 3 configurations in isolation and maximum contention scenarios.

Since the bus performance strongly depends on the number of bus accesses performed, for each benchmark we show average execution time results for 1,000 runs of each configuration. Having a significant number of runs is important to carry out a fair quantification of bus performance since in our platform cache behavior and bus arbitration are randomized.

Results under contention: Figure 1 shows per-benchmark slowdown results, i.e. performance normalized to the result obtained for RP in isolation. As shown, with EEMBC slowdowns are below 4x. This is so because in general real-time workloads such as EEMBC automotive do not saturate the bus. Yet, we can see that when CBA is not in place, slowdowns are high under maximum contention (3.34X for `matrix`).

When CBA is used, execution times of tasks under maximum contention are much lower suffering in the worst case a 2.34X slowdown. We have also evaluated a configuration in which the TuA receives more bandwidth than its contenders (H-CBA). In particular, each cycle the TuA recovers $1/2$ cycles of budget and each other core only $1/6$ cycles. This virtually allocates 50% of the bandwidth to the core where the TuA runs. As shown in the Figure, H-CBA reduces the maximum slowdown experienced across benchmarks, thus showing the effectiveness of CBA also with heterogeneous bandwidth allocation. However, this solution is better suited to applications with heterogeneous bandwidth requirements like those found is the space domain [7]. Since EEMBCs do not have significant bandwidth requirements we do not always get significant performance improvements and using CBA configuration would worsen the performance of any task running in the other cores.

Results in Isolation. Another important effect we can observe is the impact that CBA arbitration has in the execution time of tasks in isolation. Given that under CBA a core is not granted access to the bus until it has enough budget, the execution of a task can get stalled. However, as we can see in Figure 1, this effect is not very important and its impact depends on how often a program has a request ready before having recovered its budget. In fact, we observe that CBA increases only the execution time by 3% on average w.r.t. the RP bus in isolation. Moreover, when H-CBA is deployed, the impact is negligible being very close to the RP bus on average. Note that, for `tblook` we observe slightly better performance with CBA in isolation than with RP arbitration and worse performance with contention with CBA than with RP. We have investigated these scenarios and verified that two conditions concur in this case: (1) `tblook` is almost insensitive to the potential delays created by CBA since its bus requests barely occur consecutively in time; and (2) `tblook` is highly sensitive to the particular (random) cache placements experienced in the experiments. For cache sensitive applications like `tblook` cache placements influence notably average execution times depending on whether "bad" cache placements occur more or less often. While this may have an effect on average performance, it does not affect WCET estimates since MBPTA builds upon EVT, which keeps only the group of high execution times to predict the WCET.

Implementation Overheads. Hardware overheads of CBA can be regarded as negligible. In particular, the processor model has been synthesized at 100MHz – the maximum reachable frequency for our multicore processor in the TerasIC board – with and without CBA. Also, the FPGA occupancy without CBA is 73% and it has grown by far less than 0.1% to implement CBA. Hence, our arbitration policy is simple enough to be implemented in real designs.

V. Conclusions

Critical real-time embedded systems need to build upon multicores to reach the level of performance required for increasingly complex functionalities. However, existing policies to arbitrate the access to hardware shared resources focus mostly on achieving fairness in terms of request counts rather than in terms of time. This leads to severe slowdowns for tasks issuing frequent-but-short requests to shared resources.

In this paper we propose an credit-based arbitration (CBA) policy that allows a fair sharing of hardware resources by balancing the true utilization of those resources. Moreover, we show that implementation costs of CBA are affordable by implementing it in a Space multicore prototyped in a FPGA. Our results show that the maximum slowdown roughly matches the core count – as one would expect – when all tasks saturate the shared resource, which compares to existing policies whose slowdown is virtually unbounded.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROXIMA Project (www.proxima-project.eu), grant agreement no 611085. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Mladen Slijepcevic is funded by the *Obra Social Fundación la Caixa* under grant Doctorado "la Caixa" - Severo Ochoa. Carles Hernández is jointly funded by the Spanish Ministry of Economy and Competitiveness (MINECO) and FEDER funds through grant TIN2014-60404-JIN. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

References

[1] *AMBA Bus Specification. http://www.arm.com/products/system-ip/amba/amba-open-specifications.php*.

[2] J. Abella et al. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *SIES*, 2015.

[3] I. Agirre et al. IEC-61508 SIL 3-compliant pseudo-random number generators for probabilistic timing analysis. In *DSD*, 2015.

[4] B. Akesson, L. Steffens, E. Strooisma, and K. Goossens. Real-time scheduling using credit-controlled static-priority arbitration. In *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 3–14, Aug 2008.

[5] Cobham Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and Users Manual*, 2011.

[6] L. Cucu-Grosjean et al. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.

[7] Javier Jalle et al. A dual-criticality memory controller (dcmc): Proposal and evaluation of a space case study. In *Real-Time Systems Symposium,RTSS*, 2014.

[8] Milos Panic et al. Enabling TDMA arbitration in the context of MBPTA. In *2015 Euromicro DSD*, 2015.

[9] M. Fernández et al. Assessing the suitability of the ngmp multi-core processor in the space domain. In *EMSOFT*, 2012.

[10] E. Francis. Autonomous cars: no longer just science fiction. *Automotive Industries*, 193, 2014.

[11] C. Hernandez et al. Towards making a LEON3 multicore compatible with probabilistic timing analysis. In *DASIA*, 2015.

[12] J. Jalle et al. Deconstructing bus access control policies for real-time multicores. In *SIES*, 2013.

[13] J. Jalle et al. Bus designs for time-probabilistic multicore processors. In *DATE*, 2014.

[14] Ahmed Jerraya, Luca P. Carloni, Florence Maraninchi, and John Regehr, editors. *Proceedings of the 12th International Conference on Embedded Software, EMSOFT 2012, part of the Eighth Embedded Systems Week, ESWeek 2012, Tampere, Finland, October 7-12, 2012*. ACM, 2012.

[15] T. Kelter et al. Static analysis of multi-core TDMA resource arbitration delays. *Real-Time Systems*, 50(2):185–229, 2014.

[16] H. T. Kung, Trevor Blackwell, and Alan Chapman. Credit-based flow control for atm networks: Credit update protocol, adaptive credit allocation and statistical multiplexing. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, pages 101–114, New York, NY, USA, 1994. ACM.

[17] K. Lahiri et al. LOTTERYBUS: a new high-performance communication architecture for system-on-chip designs. In *DAC*, 2001.

[18] E. Mezzetti and T. Vardanega. On the industrial fitness of wcet analysis. In *WCET Workshop*, 2011.

[19] M. Paolieri et al. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, 2009.

[20] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

[21] E. Salminen et al. Benchmarking mesh and hierarchical bus networks in system-on-chip context. *J. Syst. Archit.*, 53(8), August 2007.

[22] M. Slijepcevic et al. pTNoC: Probabilistically time-analyzable tree-based noc for mixed-criticality systems. In *DSD*, 2016.

[23] R. Wilhelm et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7:1–53, May 2008.