Bus Designs for Time-Probabilistic Multicore Processors

Javier Jalle[*,†], Leonidas Kosmidis[*,†], Jaume Abella[†], Eduardo Quiñones[†], Francisco J. Cazorla[†,‡]
[*]Universitat Politècnica de Catalunya [†]Barcelona Supercomputing Center [‡]Spanish National Research Council (IIIA-CSIC)

Abstract—**Probabilistic Timing Analysis (PTA) reduces the amount of information needed to provide tight WCET estimates in real-time systems with respect to classic timing analysis. PTA imposes new requirements on hardware design that have been shown implementable for single-core architectures. However, no support has been proposed for multicores so far.**

In this paper, we propose several probabilistically-analysable bus designs for multicore processors ranging from 4 cores connected with a single bus, to 16 cores deploying a hierarchical bus design. We derive analytical models of the probabilistic timing behaviour for the different bus designs, show their suitability for PTA and evaluate their hardware cost. Our results show that the proposed bus designs (i) fulfil PTA requirements, (ii) allow deriving WCET estimates with the same cost and complexity as in single-core processors, and (iii) provide higher guaranteed performance than single-core processors, 3.4x and 6.6x on average for an 8-core and a 16-core setup respectively.

I. Introduction

Computational demands for many industries such as avionics, automotive, railway and medical have experienced an unprecedented growth to cope with more sophisticated functionalities. The value of Critical Real-Time Embedded Systems (CRTES) increasingly depends on their software component, hence, achieving high guaranteed performance is of paramount importance in all these markets. This has motivated the use of processors with high-performance features including cache memories and multicores. Unfortunately, the adoption of such performance-improving features challenges the computation of tight worst-case execution time (WCET) estimates [12].

In this context, Probabilistic Timing Analysis (PTA) [10][8][9] has appeared as an alternative to conventional timing analysis. PTA provides probabilistic WCET (pWCET) estimates, for arbitrarily low exceedance probabilities (e.g 10^{-15}). In that sense, PTA expresses timing correctness with probabilities of failure, as occurs with current system reliability of an embedded safety-critical system that is expressed in terms of probabilities for hardware failures, software functional faults and for the system as a whole. The main advantage of PTA is that it is less dependent on execution history, allowing to significantly reduce the amount of information required to obtain tight WCET estimates in comparison to other timing analysis approaches. pWCET estimates have shown to be competitive with the estimates obtained with other timing analysis techniques [29].

PTA can be applied either in a static (SPTA) [8] or measurement-based (MBPTA) [9] manner. In this paper we focus on the latter as it is closer to industrial practice. MBPTA derives probabilities by collecting execution time observations of end-to-end runs of an application running on the target hardware. MBPTA requires that the timing events under consideration, i.e. the observed program execution times, have a distinct probability of occurrence and can be modelled with *independent and identically distributed* (i.i.d.) random variables. Solutions for single-core architectures [17][9] show how processor cores with a similar processor architecture to the Aeroflex Gailser LEON3 and LEON4 [4] can be easily adapted to achieve PTA requirements. In particular, limited modifications are required in the cache placement and replacement [17] policies to make them PTA-compliant.

Multicores offer several benefits to CRTES such as higher performance per watt than single-core systems and co-hosting several tasks into the same chip, reducing the overall hardware procurement, hence reducing size, weight and power costs. Unfortunately, to the best of our knowledge, no multicore architecture has been proven to meet PTA requirements. The main stumbling block in proving probabilistic bounds to the execution time of applications in multicores is the *deterministic* and *history-dependent* behaviour of shared resources such as bus policies, which impeded the application of PTA approaches.

The interconnection network is one of the most critical shared resources in multicore processors. Several studies show that hierarchical bus configurations scale well to systems with high number of cores, while providing good area-performance trade-offs and retaining many of the advantageous features of simpler bus arrangements [26]. In the same line, other studies show that bus-based networks decrease energy consumption and simplify network protocol design and verification, with no loss in performance [11].

In this paper, we describe new low-cost PTA-compliant bus arbitration policies that break (1) the deterministic behaviour of the bus and (2) the dependence of a given application's execution time on the behaviour of co-hosted applications by means of randomised arbitration policies.

- Lottery arbitration bus: Under this bus arbitration, which is based on [19], on every round the arbiter selects one core to access the bus randomly. This breaks any dependence between applications in the access to the bus, but makes that a request may have to potentially wait an infinite number of arbitration rounds to be granted access to the bus, which degrades the pWCET estimates derived with PTA techniques.
- Randomised-permutation arbitration bus: We propose this new bus arbitration policy under which every N rounds, where N is the number of bus contenders, the arbiter generates a random permutation for the contenders. That sequence determines when each contender can use the bus. This arbitration provides an upper-bound to the number of rounds a request has to wait and leads to tighter pWCET estimates than the lottery arbitration bus.

Our results show that our proposed bus designs (i) fulfil PTA requirements which is proven by using proper independence and identical distribution tests, (ii) allow deriving pWCET estimates with *exactly* the same methods and tools as for single-core processors, thus keeping timing verification costs low, and (iii) allow increasing guaranteed performance, for an exceedance probability of 10^{-15} per run, by 1.8x, 3.4x and 6.6x for 4, 8 and 16-core setups respectively.

II. Background on PTA

PTA generates a distribution function, or pWCET function, that upper-bounds the execution time of the program under analysis, guaranteeing that the execution time of a program only exceeds the corresponding execution time bound with a

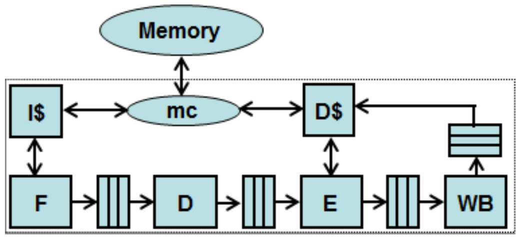

Fig. 1. Block diagram of the PTA-compliant core architecture used in [17].

probability lower than a given target probability (e.g., 10^{-15} per hour). The probabilistic timing behaviour of a program or an instruction (or in fact any component) can be represented with an Execution Time Profile (ETP). The different execution times of a program (or latencies of an instruction or a particular resource) and their associated probabilities are defined in the ETP. That is, an ETP represents a probability distribution function, see Equation 1, where p_i is the probability of the program/instruction/resource to take latency l_i, with $\sum_{i=1}^{k} p_i = 1$.

$$ETP = \{\vec{l}, \vec{p}\} = \{\{l_1, l_2, ..., l_k\}, \{p_1, p_2, ..., p_k\}\} \quad (1)$$

The *convolution* function, represented by $\otimes$, is used to combine ETPs, leading to a new ETP in which all possible pairs of execution times from the two ETPs are added and the probabilities multiplied [8]. For example, let $E_1 = \{(2, 101, 200), (0.1, 0.4, 0.5)\}$ and $E_2 = \{(2, 101), (0.6, 0.4)\}$ be two ETPs. Then their convolution is:

$$E_r = \{\{2{+}2,\ 2{+}101,\ 101{+}2,\ 101{+}101,\ 200{+}2,\ 200{+}101\},\ \{0.1{\times}0.6,\ 0.1{\times}0.4,\ 0.4{\times}0.6,\ 0.4{\times}0.4,\ 0.5{\times}0.6,\ 0.5{\times}0.4\}\}.$$

And given that $101 + 101 = 2 + 200$ we collapse duplicate pairs resulting in: $E_r = \{(4,\ 103,\ 202,\ 301),\ (0.06,\ 0.28,\ 0.46,\ 0.2)\}$.

PTA requirements on hardware design. MBPTA requires that the events under analysis, program execution times, can be modelled with *i.i.d.* random variables [8][1].

The observed execution times fulfil the i.i.d. properties if observations are independent across different runs and a probability can be attached to each potential execution time. The existence of an ETP for each dynamic instruction ensures that i.i.d properties are achieved at the level of end-to-end execution time observations [3]. ETPs, however, cannot be derived with standard (deterministic) processor architectures since events affecting execution time (e.g., bus access policy) on those architectures cannot be attached a probability of occurrence.

In a multicore, ETPs cannot be obtained for some deterministic bus policies such as fixed priority arbitration, while for others like round robin [23] ETPs can be derived.

Functional (causal) dependencies, such as those produced by data dependencies, do not break the i.i.d. behaviour, because it is not required that the probability distribution for an instruction is independent of the sequence of preceding instructions, but that the observed timing behaviour for each dynamic instruction across different runs is i.i.d [3], [18].

III. PTA IN MULTICORE SYSTEMS

One of the difficulties in the use of multicores in CRTES emanates from inter-task interferences when accessing shared resources. Inter-task interferences appear when two or more tasks sharing a hardware resource, access it simultaneously. An arbitration mechanism determines which contending task is granted access to the shared resource, which affects the execution time and the WCET of running tasks.

A. Single-core Probabilistically Analysable Hardware Designs

The basic principle to design probabilistically analysable hardware is to control the sources of execution time variation (i.e. jitter) [8][9]. Jitterless resources have a fixed latency, independent of the input request or of the previous history of requests accessing that resource. Jitterless resources (e.g. integer adders) are easy to model: its ETP has a single latency with probability 1. Resources with jitter, or *jittery resources* have a variable latency. Their latency depends on the execution history of the program or on the particular request sent to that resource. Jittery resources have a variable impact on the WCET estimate for a given program. Jittery resources are either (i) enforced to always respond on their worst-case latency, so their upper-bounded timing behaviour also becomes i.i.d., or (ii) redesigned so that their timing behaviour depends on random events.

Following this approach [17] proposes a PTA-compliant single-core pipelined architecture, see Figure 1. The architecture comprises fetch (F), decode (D), execute (E) and write-back (WB) stages. In between all stages there are latches or queues. Additionally, the WB stage comprises a write buffer in which stores are put until they are sent to cache. Loads are processed in program order in the execute stage. The accesses to the instruction and data caches happen in the fetch and execute stages respectively. This pipeline design is similar to LEON3/4 designs given that core operations have a fixed latency.

Data and instruction caches deploy random placement and random replacement policies [17] as needed to satisfy PTA requirements. A fixed-latency memory controller (mc) serves as the bridge between caches and memory.

In general, for a cache with S sets and W ways, starting from an empty cache state and given the sequence $< A_i, B_1, ..., B_k, A_j >$, where A_i and A_j correspond to accesses to the same cache line and no B_l (where $1 \leq l \leq k$) accesses the cache line where A_j is, the miss probability of A_j is as [17]:

$$P_{miss_{A_j}}(S,W) = \left(1 - \left(\frac{W-1}{W}\right)^{\sum_{l=1}^{l=k} P_{miss_{B_l}}}\right) \cdot \left(1 - \left(\frac{S-1}{S}\right)^{k}\right) \quad (2)$$

Note that the equation becomes an approximation when B_l accesses repeat and/or the initial cache state is not empty. However, this is irrelevant for MBPTA, since what really matters is that each access has a probability of hit/miss rather than the particular value of that probability.

The hit probability is used to compute the ETP of each cache access as follows where lat_{hit} and lat_{miss} are the cache hit and miss latency respectively:

$$ETP_{cache} = \{\{lat_{hit}, lat_{miss}\}, \{P_{hit_{A_j}}(S,W), P_{miss_{A_j}}(S,W)\}\} \quad (3)$$

On the event of a miss in the instruction or data cache, the missing instruction generates an access to the shared bus hierarchy. Next, we present several bus architectures and their corresponding probabilistic analyses. The resulting ETP for each bus should be composed with the miss part of the ETP_{cache}. For instance, if the ETP of the bus is $ETP_{bus} = \{\{l_1, l_2, l_3\}, \{p_1, p_2, p_3\}\}$, the resulting ETP of composing cache and bus effects would be:

$$ETP_{\substack{cache\\+bus}} = \{\{lat_{hit}, lat_{miss} + l_1, lat_{miss} + l_2, lat_{miss} + l_3\}, \{P_{hit}, P_{miss} \cdot p_1, P_{miss} \cdot p_2, P_{miss} \cdot p_3\}\} \quad (4)$$

[1]Two random variables are said to be independent if they describe two events such that the occurrence of one event does not have any impact on the occurrence of the other event. Two random variables are said to be identically distributed if they have the same probability distribution function.

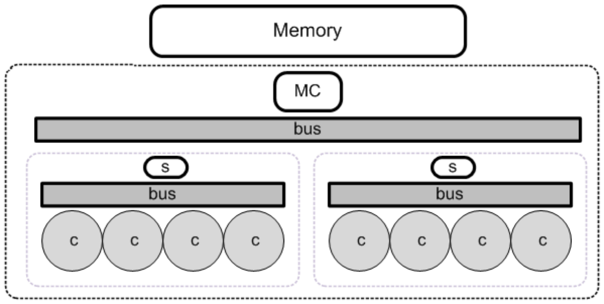

Fig. 2. Baseline multicore architecture considered in this paper. 'c' stands for core, 's' for switch and 'mc' for memory controller.

B. Bus Designs for Time-Probabilistic Multicores

Historically, clustered architectures have been considered in computer architecture. At the processor core level, execution pipelines are split into clusters (e.g. the IBM POWER7 [15]) to decrease hardware cost while efficiently exploiting instruction-level parallelism. At the chip level, processor implementations of many-core architectures (e.g. ARM Cortex A15 MPCore [5]) may group several cores into clusters as a means to reduce implementation costs. Besides, clusters enable voltage and frequency scaling as well as power-gating at the granularity of several cores, since having those mechanisms on a per-core basis has high hardware cost [21][1].

Following this philosophy, we use a baseline clustered architecture as shown in Figure 2, where cores are equipped with private data and instruction caches. We evaluate our designs in several setups in which we vary the number of clusters and the number of cores per cluster.

Inter-task interferences cause that the execution time of a task depends on the accesses of other tasks to shared resources, e.g. the bus. Taking into account the effect that any instruction of any task may have on any other instruction of any other task is infeasible. This would simply make the usage of the probabilistic approach intractable. To break this dependence we design our multicore such that we make that *the worst effect that one task can incur on the execution of any other task due to inter-task interferences can be probabilistically bounded.* This makes our design *time composable*, meaning that the pWCET estimate obtained for a given task is independent of the tasks that may run concurrently in the processor.

Overall, our bus designed so that every individual processor instruction can be characterised by a distinct ETP that has no dependence on any instruction of any other task. Next, we present the proposed bus designs, which allow deriving an ETP for each instruction independently of any other task.

C. Lottery Bus

We define an *arbitration round* or simply round as the number of processor cycles that each core needs to send any request to the bus. In this approach, on every round the arbiter selects one core to access the bus using a random policy. A similar bus was analysed in [19]. However, unlike [19], we assume that all bus contenders have always pending requests, although in a given cycle, only a subset of the contenders may have pending requests. This assumption makes our design time composable, upper-bounding inter-task interferences, and hence independent of the traffic generated by any other task running concurrently on the processor. Otherwise, ETP for memory operations of one task would depend on the traffic generated by other tasks, breaking time composability.

In a N core single-bus processor, the probability of a request to be selected in round $k+1$ is given by Formula 5, where the first element is the probability of the request not being granted access in the first k rounds and the second element the probability of being selected in the $k+1$ round.

$$p^{k}_{lotarb} = \left(1 - \frac{1}{N}\right)^{k} \cdot \frac{1}{N} \tag{5}$$

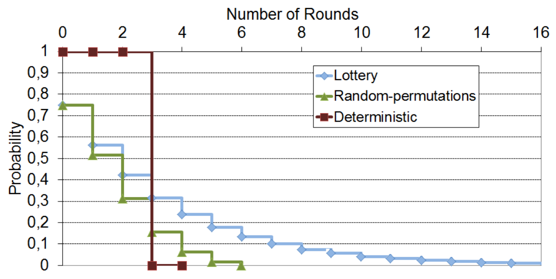

Fig. 3. Probability of not being granted access as a function of the number of arbitration rounds for a bus with 4 cores.

Blue diamonds in Figure 3 shows the probability of a bus request not to be granted access after k arbitration rounds for a bus shared by 4 cores. We observe that the larger the number of rounds in which a contender participates, the lower its probability not to be selected. There is a probability the contender not to be granted access after a large number of rounds. However, this probability decreases exponentially and more importantly, it is probabilistically computable.

If arbitration rounds have durations longer than 1 cycle, say L cycles, then a request may become ready in the middle of a round. In that case, the request has to wait until an arbitration round boundary before it can compete to get access to the bus.

Equation 6 shows the ETP for a bus access. (1) The first element convolved in the equation is the delay to align the request with the cycle at which the next round starts. A bus access request is initiated on a miss to the data or instruction cache. Given that the event 'cycle in which an access misses in the data or instruction cache' is a random event (and hence so is the cycle in which the access to the bus happens), the probability of a bus request to arrive in a particular cycle can be computed. In particular, every request to the bus may arrive in any cycle of the arbitration round $(0, ..., L-1)$ with a given probability p_{cyci} to arrive in cycle i with $0 \le i \le L-1$. Note that for the ETP it does not matter whether p_{cyci} follow any particular distribution as long as it is probabilistic.

(2) The second element convolved is the number of rounds that the request waits. Each round has L cycles and Equation 5 is used to compute the probability of a request to be selected in a given round.

(3) Finally, the last element convolved is the actual latency of the bus access request: it takes L cycles with 100% probability.

$$\begin{aligned} ETP_{bus_{lot}} \quad = \quad & \{\{0,1,...,L-1\},\{p_{cyc1},p_{cyc2},...,p_{cycL-1}\}\} \otimes \\ & \{\{0,L,2L,...\},\{p^{0}_{lotarb},p^{1}_{lotarb},p^{2}_{lotarb},...\}\} \otimes \\ & \{\{L\},\{1\}\} \end{aligned} \tag{6}$$

D. Randomised Permutations

An *arbitration window* or simply window refers to N consecutive bus arbitration rounds. Under this arbitration policy, in each window one round is randomly assigned to each of the N contenders. Each round has a duration of L cycles, the maximum bus cycles that any request may take. In this approach, at every window boundary the arbiter generates a random permutation for the N contenders (cores). This sequence determines the order in which contenders can use the bus.

Analogously to the fact that the arrival cycle of a request in a round follows a given probability distribution function, there is a probability defining the round in a window in which requests arrive. This is so since the accesses to the bus are initiated on random events: miss to the data or instruction caches. As explained before, MBPTA is not dependent on the particular distribution function this is as long as it is probabilistic. Without loss of generality and for the purpose of this explanation we assume that the probability function describing the arrival round of each request is uniform, that is, there is a probability $\frac{1}{N}$ a request to arrive in a particular round[2]. We identify two extreme cases:

- The shortest delay (0 cycles) occurs when a request becomes ready when the core it belongs to gets its round.
- The worst delay occurs when (i) a request belongs to a core that gets the first round in the current permutation and (ii) the last round in the next permutation, and (iii) the first round of the current permutation has just elapsed. In this case, the request should wait $2N-2$ rounds corresponding to the $N-1$ remaining rounds of the current permutation and the first $N-1$ rounds of the next one.

The probability of a request to wait k rounds to get access to the bus, with $0 \leq k \leq 2N-2$ is as follows:

$$p_{perarb}^{k} = \frac{max(N-k,0)}{N^2} + \sum_{i=max(1,N-k)}^{min(N-1,2N-k-1)} \frac{i}{N^3} \qquad (7)$$

The first addend in the equation is the probability of finding the appropriate round in the remaining part of the current permutation, whereas the second part stands for the probability of finding the appropriate round in the next permutation *if and only if* such round was not found in the current permutation. Green triangles in Figure 3 show the probability of a bus request not to be granted access after k arbitration rounds. We observe that after $2N-2$ rounds, the probability not to be selected is 0.

The ETP of the random permutation bus, shown in Equation 8, is obtained as the convolution of three components, as for the lottery bus. The first component in the convolution is the time and associated probability to align the request to the start of the next round; the second one stands for the waiting rounds until the request is granted access, and the final one stands for the actual bus access latency of the request.

$$\begin{aligned} ETP_{bus_{per}} \; = \; & \{\{0,1,...,L-1\}\{p_{cyc1},p_{cyc2},...,p_{cycL-1}\}\} \otimes \\ & \{\{0,L,...,k\cdot L,...,(2N{-}2)\cdot L\}, \\ & \quad \{p_{perarb}^{0},p_{perarb}^{1},...,p_{perarb}^{k},...,p_{perarb}^{2N-2}\}\} \otimes \\ & \{\{L\},\{1\}\} \end{aligned} \qquad (8)$$

E. Deterministic Bus

Alternatively to time-randomised buses, a deterministic bus deploying round-robin policy could be used [14]. This requires that a task is assumed always to suffer the worst latency when accessing the bus [23], with probability of 1. As a result, the ETP still remains as a safe upper-bound at deployment and is analogous to that of randomised buses except for the second term, which is now fixed: $\{\{(N-1)\cdot L\},\{1\}\}$ (see Figure 3).

F. Hierarchical Buses

Based on the ETPs derived for simple buses we can derive the ETP for a hierarchical bus network. In our bus setup, on the one hand, we have an intra-cluster bus ("ibus") per cluster, which is accessed by N_{co} contenders (cores) and has an access latency of L_i cycles. Each cluster has a switch that connects the intra-cluster bus to the inter-cluster bus ("ebus"). The simple switch adds a fix latency to the end-to-end latency of the interconnection network, S. The ebus is connected to each of the N_{cl} clusters and has a latency L_e. Usually, $L_e \geq L_i$ as the ebus is longer than the ibus as the ebus connects distant clusters.

[2]For our processor setup, our results show that the arrival of bus requests across rounds in the window is uniform.

All three resources of the hierarchical bus are accessed serially: ibus, switch and ebus. The ETPs of each resource can be easily composed as shown in Equation 9. Note that the bus ETPs are characterised by parameters L, latency, and N, number of contenders, as described Sections III-C and III-D.

$$ETP_{hbus} = ETP_{ibus} \otimes ETP_{switch} \otimes ETP_{ebus} \qquad (9)$$

$$\begin{aligned} ETP_{ibus} &= ETP(L_i, N_{co})_{bus} \\ ETP_{switch} &= \{\{S\},\{1\}\} \\ ETP_{ebus} &= ETP(L_e, N_{cl})_{bus} \end{aligned}$$

IV. Results

A. Experimental Setup

We use a cycle-accurate PowerPC simulator [27] to model the probabilistically analysable processor. Cores are as presented in Section II: 4-stage, In-order pipelined cores with a memory hierarchy composed of separated instruction and data caches. The size of each cache is 4-KB with 64-byte line size and 4-way associativity. The latency of the fetch stage depends on whether the access hits or misses in the instruction cache: a hit has 1-cycle latency and a miss has variable latency to access to memory. After the decode stage, memory operations access the data cache so they can last 1 cycle or a variable latency to access memory in case of a miss. The remaining operations have a fixed execution latency (e.g. integer additions take 1 cycle). The bus latency is $L = 8$ cycles for all buses. For the memory controller we use the solution proposed in [22] which upperbounds the effect of inter-task interferences on the requests of a core to the memory controller.

We use different multicore setups featuring the hierarchical bus architecture presented in Section III-F, varying the number of cores from 4 to 16, distributed in clusters of 4 or 8 cores. Each configuration is represented with a pair of numbers: the first one denotes the number of cores per cluster and the second one the number of clusters. For instance, an 8 core configuration with 2 clusters of 4 cores each will be a *4x2* configuration.

Following the method in [9] we carried out up to 1,000 runs per program and used EVT to extract pWCET estimates for each of the EEMBC Autobench benchmarks suite [24] that mimic some real-world automotive critical applications' behaviour. We used: *a2time*, *aifirf*, *cacheb*, *canrdr*, *puwmod*, *rspeed*, *tblook* and *ttsprk*.

B. Hardware Overhead

Lottery bus: The lottery arbitration simply requires $\lceil \log_2 N \rceil$ bits to select which contender is granted access in each round. Given that the number of contenders, N, is typically a power-of-two, using *exactly* $\log_2 N$ bits produced by a pseudo-random number generator (PRNG) is enough to select the particular contender that is granted access. Note that efficient PRNGs already exist in real processors implementing, for instance, random-replacement policies in cache [13], [28].

Random permutations can also be implemented with very low cost. A *randperm* register with $N \cdot \log_2 N$ bits is needed to store the N identifiers of $\log_2 N$ bits for each of the contenders. In order to generate a random permutation, $N-1$ random bits, called *randbits* generated by a PRNG are used. We swap the identifiers in the *randperm* register in a hierarchical way based on the values of *randbits*. For instance, if $N = 4$, we need 3 random bits. The first bit in *randbits* determines whether the first and second identifiers in *randperm* are swapped (1) or not

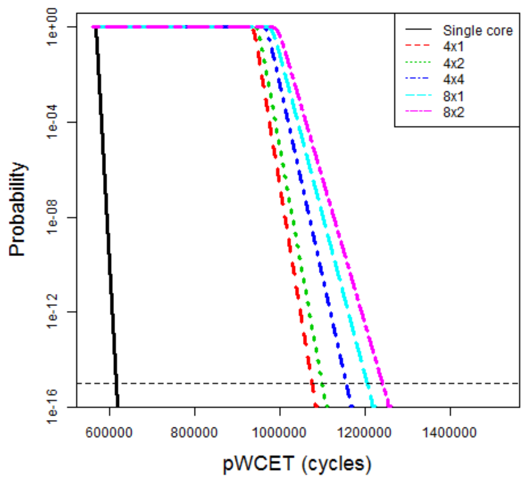

Fig. 4. pWCET estimates for `canrdr` under different multicore setups.

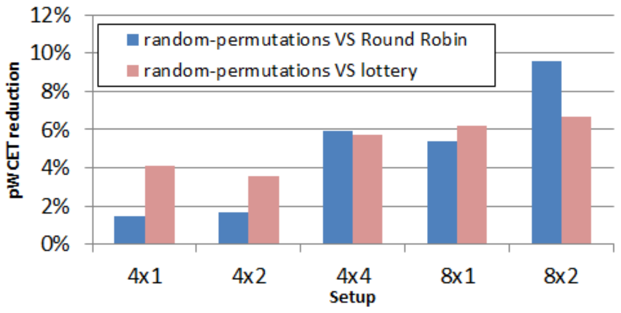

Fig. 5. pWCET reduction when using randomised permutations arbitration in the buses with respect to using lottery and deterministic arbitration respectively.

(0). The second random bit determines whether the third and fourth identifiers in *randperm* are swapped. The third random bit determines whether the first pair of identifiers is swapped with the second pair. If the current state of *randperm* is '00-01-10-11' (so contenders order is 0, 1, 2, 3) and *randbits* is '101', the new permutation will be '10-11-01-00' (the first and second ids were swapped, and the first and second pair were also swapped), so the new contenders order will be 2, 3, 1, 0. It can be seen that the probability of a particular contender to reach any particular position in the permutation is exactly $\frac{1}{N}$ regardless of its position in the previous permutation since $\log_2 N$ random bits will determine its new position. It can be also seen that all permutations cannot be generated. For instance, in the example before contenders 2 and 3 cannot be in different halves of the permutation. However, this is irrelevant because any particular contender occupies each position in the permutation with the same probability and the order of contenders in the remaining positions has no effect on the current contender. For instance, in the example before it is irrelevant for contender 3 whether the contender in the first position is 0, 1 or 2.

C. Fulfilling the i.i.d Properties

Our bus designs guarantee that observed execution times fulfil the properties required by MBPTA given that we are able to derive ETPs. We contrast this empirically by analysing whether execution times of EEMBC benchmarks are i.i.d.

In order to test independence we use the Wald-Wolfowitz independence test [7]. We use a 5% significance level (a typical value for this type of tests), which means that absolute values obtained with this test must be below 1.96 to prove independence. For identical distribution, we use the two-sample Kolmogorov-Smirnov i.d. test [6]. For 5% significance, the outcome of the test should be above the threshold (0.05) to indicate i.d. Both tests were passed for all benchmarks when running each benchmark 1,000 times.

TABLE I
PWCET ESTIMATE FOR 10^{-15} FOR EACH EEMBC UNDER ALL SETUPS W.R.T THEIR PWCET ESTIMATE IN SINGLE-CORE.

benchmark/cores	4x1	4x2	4x4	8x1	8x2
a2time	3,31	3,35	3,73	3,59	3,72
aifirf	3,29	3,75	3,76	3,72	3,90
cacheb	3,25	3,49	3,63	3,55	3,81
canrdr	1,74	1,78	1,87	1,95	2,01
puwmod	1,80	1,80	1,79	1,91	1,92
rspeed	1,67	1,70	1,74	1,75	1,79
tblook	3,39	3,44	3,50	3,52	3,60
ttsprk	2,07	2,10	2,14	2,11	2,21

D. Comparison of Arbitration Policies

Section III-D shows that the ETP derived for the random-permutation arbitration is better than for the lottery and deterministic arbitration, i.e. the area below the random-permutation curve in Figure 3 is smaller than for lottery and deterministic arbitrations.

Lottery and deterministic arbitration need $N-1$ rounds on average to grant access to the shared resources, where N stands for the number of contenders. Conversely, random-permutation arbitration needs around $\frac{N}{2}$ rounds. For instance, for 4, 8 and 16 contenders lottery and deterministic arbitration need 3, 7 and 15 rounds on average, whereas random-permutation needs 1.8, 4.2 and 8.8 rounds on average.

This translates into lower pWCET estimates for the random-permutation arbitration, as shown in Figure 5. We consider an exceedance probability of 10^{-15} per run, which is low enough to meet highest safety levels according with avionics standards [29][2]. We observe that pWCET reductions range between 3.5% and 6.7% w.r.t. lottery arbitration and between 1.5% and 9.6% w.r.t. deterministic arbitration. pWCET reductions w.r.t. lottery arbitration are higher because its impact on average delay is similar to that of deterministic delay, but lottery arbitration introduces more variability (recall that with lottery arbitration there is a probability contenders not being granted access after a large number of rounds). Hence, although the best case of lottery arbitration is better than that of deterministic one, its worst case is worse than that of deterministic arbitration, thus leading to worse pWCET estimates.

E. MBPTA: EVT Projections

In this section, we provide pWCET estimates obtained with MBPTA [9] for EEMBC benchmarks under several multicore setups. In all setups we deploy random-permutation arbitration in buses, since, as it has been shown in Section IV-D, outperforms the other policies. Note that MBPTA has been used so far *only* on top of single-core architectures.

As an example, Figure 4 shows the EVT projections for `canrdr`. We observe that the higher the number of cores, the bigger the pWCET estimate, because bus contention increases. Also, configurations with many cores per cluster, *8x1* (1 cluster with 8 cores) and *8x2*, have higher pWCET than the other configurations with same number of cores, *4x2* and *4x4* respectively. Since latency in buses depends on the number of contenders and not on their specific traffic, it is better to reduce the maximum number of contenders in any bus. The big discrepancy between the single-core and multicore setups is due to the fact that with only 1 core there is low memory contention, which increases for the multicore case.

Table I shows pWCET estimates for each EEMBC under the different multicore setups with respect to the pWCET estimate obtained when each benchmark runs in a single-core setup. Although per-task pWCET estimates increase in a multicore, the fact that a larger number of tasks can be scheduled in the multicore, increases the guaranteed load that can be run

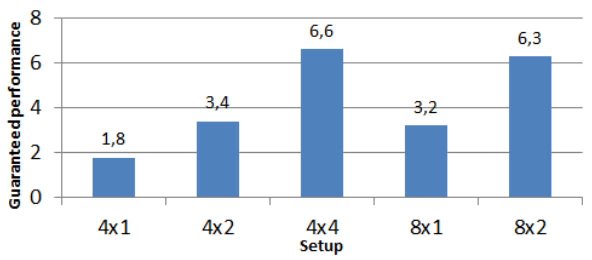

Fig. 6. Guaranteed performance when using randomised permutations arbitration in the buses with respect to the single-core case.

by the (multicore) CPU.: The *guaranteed IPC* (instructions per cycle) for a benchmark b under a given processor setup(s) and exceedance probability (e), $gIPC_{se}^{b}$, is given by the number of instructions b executes divided by its pWCET at that exceedance probability. The average guaranteed IPC per core under a given setup s is obtained as $gAvgIPC_{se} = \frac{\sum_{i=1}^{B} gIPC_{se}^{i}}{B}$ where B stands for the number of benchmarks. Finally, the guaranteed performance for a given setup s with N cores is $Total - gIPC_{se} = gAvgIPC_{se} \cdot N$.

Figure 6, shows $Total - gIPC_{se}$ with $s = \{4x1, 4x2, 4x4, 8x1, 8x2\}$ and $e = 10^{-15}$. We observe how $TotalgIPC_{se}$ grows by 1.8x in the 4-core setup (4x1), 3.4x and 3.2x in the 8-core setups (4x2 and 8x1 respectively) and by 6.6x and 6.3x in the 16-core setups (4x4 and 8x2 respectively), thus proving that multicores equipped with our time-randomised bus design provide much higher guaranteed performance than its single-core counterparts.

V. Related Work

Recent proposals enable the use of multicores in real-time systems [20][30]. A commonality of all these proposals is that they are meant to work in conjunction with static timing analysis or measurement-based timing analysis and not with probabilistic timing analysis as it is the focus of this paper.

At the processor core level, several proposals enable the execution of non-hard real-time tasks and hard real-time tasks, making the execution of the former transparent. That is, non-hard real-time tasks are prevented from affecting the execution time of hard real-time tasks [20]. At the chip level, several techniques deal with the communication bus by proposing means to upper-bound the effect (interferences) that one task running on one core can cause on the other running tasks [23][25][16].

Regarding probabilistic timing analysis, several techniques [8][9] have been proposed to cover PTA's static and measurement-based variants respectively. At hardware level, a PTA-compliant single-core architecture is presented in [17]. However, to the best of our knowledge, no multicore PTA-compliant architecture has been presented so far.

VI. Conclusions and Future Work

PTA enables deriving pWCET estimates of applications running on complex hardware in safety-critical real-time systems, while reducing the amount of information about the hardware and software to that end. Yet, PTA relies on some properties that existing multicore processors fail to provide. In this paper we have focused on one of the resources in which most inter-task interferences appear on a multicore: the bus. We have described several low-cost PTA-compliant bus designs that break the deterministic behaviour of the bus and the dependence of a given application execution time on the behaviour of co-hosted applications by means of randomised arbitration policies. Our results prove that the proposed designs (i) fulfil PTA requirements, (ii) can be analysed with existing PTA tools for single-core processors and (iii) provide improved guaranteed over single-core designs: 1.8x, 3.4x and 6.6x for 4, 8 and 16-core setups respectively.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the PROARTIS Project (grant agreement 249100) and the PROXIMA Project (grant agreement 611085). This work has also been supported by the European Space Agency under NPI Contract 40001102880, Spanish Ministry of Science and Innovation (grant TIN2012-34557) and the HiPEAC Network of Excellence.

References

[1] SCC external architecture specification (EAS). In *http://www.intel.es/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-architecture-brief.pdf. Revision 0.94.*
[2] Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. *ARP4761*, 2001.
[3] J. Abella et al. Measurement-based probabilistic timing analysis and i.i.d property. White Paper. 2013. http://www.proartis-project.eu/publications/MBPTA-white-paper.
[4] Aeroflex Gaisler. *Quad Core LEON4 SPARC V8 Processor - LEON4-NGMP-DRAFT - Data Sheet and Users Manual*, 2011.
[5] ARM Ltd. Cortex-A15 Processor. In *http://www.arm.com/products/processors/cortex-a/cortex-a15.php*.
[6] S. Boslaugh and P.A. Watters. *Statistics in a nutshell.* O'Reilly Media, Inc., 2008.
[7] J.V. Bradley. *Distribution-Free Statistical Tests*. Prentice-Hall, 1968.
[8] F.J. Cazorla et al. Proartis: Probabilistically analysable real-time systems. *ACM TECS*, 2012.
[9] L. Cucu et al. Measurement-based probabilistic timing analysis for multi-path programs. In *ECRTS*, 2012.
[10] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *the 22nd IEEE Real-Time Systems Symposium (RTSS01)*, 2001.
[11] Aniruddha Udipi et al. Towards scalable, energy-efficient, bus-based on-chip networks. In *HPCA*, 2010.
[12] R. Wilhelm et al. The worst-case execution time problem: overview of methods and survey of tools. *Trans. on Embedded Computing Systems*, 7(3):1–53, 2008.
[13] http://www.arm.com. *ARM Cortex-R4 processor manual.*
[14] J. Jalle, J. Abella, E. Quinones, L. Fossati, M. Zulianello, and F.J. Cazorla. Deconstructing bus access control policies for real-time multicores. In *SIES*, 2013.
[15] R. Kalla, B. Sinharoy, W.J. Starke, and M. Floyd. POWER7: IBM's next-generation server processor. In *IEEE Micro*, 2010.
[16] T. Kelter et al. Bus-aware multicore WCET analysis through TDMA offset bounds. In *ECRTS*, 2011.
[17] L. Kosmidis, J. Abella, E. Quinones, and F.J. Cazorla. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.
[18] L. Kosmidis, T. Vardanega, J. Abella, E. Quinones, and F.J. Cazorla. Applying measurement-based probabilistic timing analysis to buffer resources. In *WCET workshop*, 2013.
[19] K. Lahiri, A. Raghunathan, and G. Lakshminarayana. LOTTERYBUS: a new high-performance communication architecture for system-on-chip designs. DAC '01, pages 15–20, 2001.
[20] MERASA. *EU-FP7 Project: www.merasa.org*, 2007-2010.
[21] E. Musol. Hardware-based load balancing for massive multicore architectures implementing power gating. In *IEEE TCAD*, volume 29, 2010.
[22] M. Paolieri, E. Quinones, F.J. Cazorla, and M. Valero. *An Analyzable Memory Controller for Hard Real-Time CMPs* . Embedded System Letters (ESL), 2009.
[23] M. Paolieri et al. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, 2009.
[24] Jason Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.
[25] J. Rosen et al. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *RTSS*, 2007.
[26] E. Salminen et al. Benchmarking mesh and hierarchical bus networks in system-on-chip context. *J. Syst. Archit.*, 53(8), August 2007.
[27] SoCLib. -, 2003-2012. http://www.soclib.fr/trac/dev.
[28] http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53. *Leon3 Processor*. Areroflex Gaisler.
[29] F. Wartel et al. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *SIES*, 2013.
[30] R. Wilhelm et al. Designing predictable multicore architectures for avionics and automotive systems. In *RePP Workshop*, 2009.