pTNoC: Probabilistically Time-Analyzable Tree-Based NoC for Mixed-Criticality Systems

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

***Abstract*—The use of networks-on-chip (NoC) in real-time safety-critical multicore systems challenges deriving tight worst-case execution time (WCET) estimates. This is due to the complexities in tightly upper-bounding the contention in the access to the NoC among running tasks. Probabilistic Timing Analysis (PTA) is a powerful approach to derive WCET estimates on relatively complex processors. However, so far it has only been tested on small multicores comprising an on-chip bus as communication means, which intrinsically does not scale to high core counts. In this paper we propose *pTNoC*, a new tree-based NoC design compatible with PTA requirements and delivering scalability towards medium/large core counts. *pTNoC* provides tight WCET estimates by means of asymmetric bandwidth guarantees for mixed-criticality systems with negligible impact on average performance. Finally, our implementation results show the reduced area and power costs of the *pTNoC*.**

I. Introduction

Multicores are well accepted as one of the main design paradigms to increase performance in critical real-time embedded systems (CRTES). In the avionics domain, for instance, the size of on-board applications has grown by a rate of 10x every 10 years [10], which has motivated the use of multicores for avionics CRTES in some Airbus studies [18]. The goal is allowing the consolidation of multiple applications with mixed criticalities onto a single hardware platform. Similar observations can be made for the automotive domain, where modern cars can have software components consisting of up to 100 million lines of code, and each car contains up to 70 computers working together under complex conditions [7]. Timing verification of CRTES requires the derivation of worst-case execution time (WCET) estimates, which is a complex task on multicore platforms. Furthermore, mixed-criticality CRTES also require high average performance (e.g., autonomous driving, unmanned vehicles, infotainment, etc.).

Deriving tight WCET estimates for tasks running on a multicore has been addressed with different families of timing analysis techniques [31]. Measurement-based probabilistic timing analysis (MBPTA) [9] is a promising approach to handle multicore complexity. MBPTA has as main benefit reducing the burden on the end-user to carry out the analysis: during the *analysis time* the user collects execution time measurements on the target platform from which MBPTA derives probabilistic WCET (pWCET) estimates that upper-bound the execution time of the program during operation. This simplicity makes MBPTA, which has been already evaluated in avionics [30] case studies, appealing for industrial use.

To obtain such benefits, MBPTA requires, on the one hand, that the hardware-software elements causing variability in programs' execution time are controlled to obtain reliable pWCET estimates [6]. This is achieved by either making those resources to work on their worst-latency (making them jitterless), or injecting randomization in their timing behavior (making them have a probabilistic behavior). Both approaches enable the analysis of those resources with MBPTA [15]. Randomization further allows delivering tighter pWCET estimates [12]. On the other hand, MBPTA requires that the execution conditions under which observations are taken at *analysis time* are conveniently set to match or upper-bound those that may occur during operation [6]. Notably, time randomized architectures are not a fiction, but concrete FPGA prototypes based on commercial processors are undergoing [11].

Among multicore shared resources the network-on-chip (NoC) has prominent impact on programs' execution time and pWCET, as it connects cores to memory and/or shared cache levels. In the context of non time-randomized multicore architectures, also known as time-deterministic architectures, several NoC designs have been evaluated including meshes [25], rings [19] and buses [12]. Among existing NoC designs, only buses have been proven MBPTA-compliant [12] for different arbitration policies. However, bus scalability is limited since its latency increases rapidly with the number of cores [23]. Further, in the context of MBPTA, proposed arbitration policies offer homogeneous guarantees and performance across cores, which do not match the heterogeneous bandwidth requirements in future mixed-criticality multicore real-time systems [28].

This paper proposes *pTNoC*, a new tree NoC design that overcomes the limitations of homogeneously-arbitrated buses to manage the abundant traffic between cores and memory and/or shared caches. Trees are chosen since they can scale to higher-core counts, have been shown to work with time-

deterministic architectures [20] and are implemented in real processors such as the P2012 [5]. The challenge lies on making a tree NoC MBPTA-compliant while providing high average performance and heterogeneous – configurable – guaranteed bandwidth allocations under a low complexity and energy envelop. Overall the main contributions of this paper are as follows:

1) We analyze several MBPTA-compliant arbitration policies for tree NoCs, and show that trees scale better to high core counts than buses in terms of homogeneous performance guarantees.
2) We propose arbitration policies to enable fine-grain and flexible heterogeneous bandwidth assignments to better match the requirements of mixed-criticality systems. This is implemented through small changes in the router arbitration policy.
3) We provide a complete evaluation in terms of WCET estimates, average performance, power and area, and compare our *pTNoC* against the only MBPTA-compliant NoC so far, a shared bus [12], showing that our *pTNoC* outperforms the MBPTA-compliant bus in all metrics. Our evaluation shows that *pTNoC* achieves tighter WCET estimates – between 16% and 25% on average for different arbitration policies – and higher average performance – around 3% – than its bus counterpart. Moreover, the mixed-criticality-aware *pTNoC* design delivers lower WCET estimates for tasks with stringent time requirements – an extra 9% reduction – with negligible effect on the average performance of other tasks.

The rest of the paper is organized as follows. Section II provides some background on MBPTA and NoC arbitration policies. Section III describes our reference multicore architecture. Section IV presents our *pTNoC* tree-based NoC design. Section V describes how to adapt NoC designs for mixed-criticality CRTES. Results are presented in Section VI. Finally, Section VII presents the main conclusions of this study.

II. Background and Related Work

A. Measurement-Based Probabilistic Timing Analysis

MBPTA [9] provides a distribution of WCET or probabilistic WCET (pWCET) curve. Each value on the pWCET curve, see Figure 1, is associated with the residual risk – expressed as a cutoff probability – with which one instance of the program is not proven to stay below that pWCET value. The cutoff probability is selected in accordance with safety standards in the application domain (e.g., DO-178B/C [24] for avionics). For instance, Figure 1 shows a pWCET curve in which a cutoff probability of 10^{-14} and the corresponding pWCET estimate are selected.

MBPTA builds on Extreme Value Theory (EVT) [16], a statistical method for approximating the tail of distributions. In particular, EVT is applied to the execution time measurements obtained in the tests performed in the *analysis stage*. MBPTA

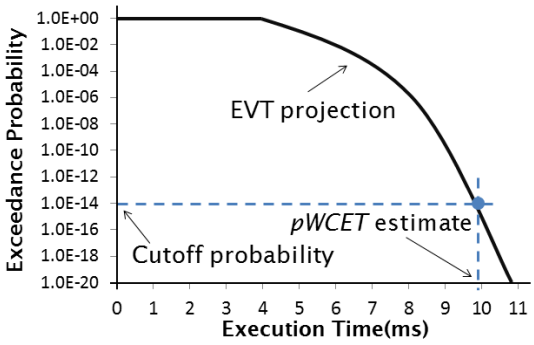


Figure 1: EVT projection (i.e. probabilistic WCET)

requires execution time measurements to upper-bound the impact that jittery hardware resources, such as caches or the NoC, can incur once the system is *deployed* [6], [15]. Such upper-bounding can be performed deterministically or probabilistically. An example of deterministic upper-bounding would be enforcing a variable-latency functional unit to operate always at its highest latency at *analysis time*. An example of probabilistic upper-bounding would be a randomly arbitrated bus [12] that is made to arbitrate across all cores at *analysis time* so that its probabilistic latency distribution at analysis time upper-bounds that during operation. By doing so, the jitter of those resources during operation is accounted for in the measurements obtained at analysis time.

We refer the interested reader to the original work on MBPTA for details on the timing analysis process including properties needed from the hardware/software platform [9], [15], the measurement collection [6], and detection and solution of anomalies [3]. MBPTA [9] has already been positively assessed for complex avionics [29], [30] case studies following a methodology close to industrial practice and MBPTA-compliant bus-based multicore designs have already been included [11] in a FPGA implementation of the NGMP processor for the space domain [8].

B. Arbitration Policies for Shared Resources

Three arbitration policies have been shown to meet the requirements of MBPTA:

- **Round-robin (RR)**. Arbitration policies such as *round-robin* are deployed in time-deterministic systems. In the context of MBPTA, when the time alignment of the requests w.r.t. the round-robin can be any – which is the least restrictive case from the user point of view – one needs to assume that each request will experience the maximum arbitration latency [12], which can be enforced at *analysis time* with the worst-case mode [21]. For the case of round-robin the highest latency a request can suffer due to the access to the bus is [21]: $(N_c - 1) \times L$, where N_c is the number of cores (contenders) and L the bus latency. This corresponds to the case in which a request from one core has to wait for one request from each other contenders to complete.

- **Lottery (LOT)**. The arbiter grants access to the different contenders randomly on each arbitration [17]. To be MBPTA-compliant, arbitration is always performed across all contenders (N_c) and only the one chosen can access the bus in that slot regardless of whether it has any pending request [12]. Therefore, large arbitration times occur with decreasing probabilities.
- **Random permutations (RP)**. Lottery arbitration may lead to theoretically infinite contention delays since there exists a non-null probability a contender waits long time to get granted access to the bus. With random-permutations [12] in each arbitration window, comprising one slot per contender, the order of the slots is randomly generated. For instance, with 3 contenders we could have the following arbitration windows (random permutations of 1,2,3): <2,1,3>, <1,2,3>, <3,1,2>, etc. This policy bounds the largest number of slots a contender may wait to get the bus (unlike lottery arbitration).

III. Reference Multicore

We consider a processor and memory architecture in which core-to-core communication is carried out with explicit messages managed by the real-time operating system (RTOS) through memory [4]. When a bus is shared across a large number of cores, the bandwidth provided to each core is reduced. In such an architecture there are two types of communications: N_c-to-1 to allow cores accessing L2 and 1-to-N_c to allow L2 responding core requests. The most suitable NoC for this type of communications is a tree since it is specifically suited to N_c-to-1 and 1-to-N_c communications. Further, a tree NoC has been used in time-deterministic real processors [5] [20]. Our view is that tree NoCs can also be made MBPTA compliant, so that larger multicores can be used in the context of MBPTA. Other designs such as meshes would fit much better designs with N_c-to-N_c communication needs and require larger area and power than a tree so we did not try to make them MBPTA compliant.

In our reference architecture memory-level parallelism is exploited across tasks running in different cores. The L2 cache can process up to one request per core in parallel, which implicitly defines the maximum throughput required for the NoC as well as the size of the buffers to use in the tree.

In this paper we consider 8- and 16-core multicores. The benefits of the tree NoC diminish for smaller multicores (e.g., 4 cores) for which buses are competitive. Larger multicores (e.g., $>$ 16 cores) may fit better a clustered architecture where each cluster has its own local memory as it has been shown in [20]. Clustering provides good scalability, isolation across clusters (which is good to limit interferences) and low power. Also, with more than 16 cores memory bandwidth becomes the bottleneck, so clustered designs with local memory controllers per cluster are convenient in that case. Note also that CRTES industry has only certified single-core and few dual-core systems, so providing solutions up to 16 cores already provides scalability beyond near-future industry requirements.

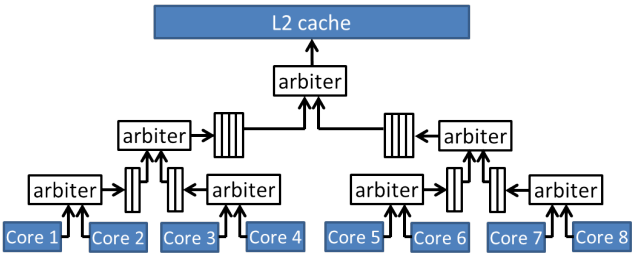


Figure 2: N_c-to-1 binary tree NoC for an 8-core setup

IV. Single-Criticality pTNoC

Figure 2 shows a tree NoC to connect 8 cores to the L2. Arbiters at the bottom level, i.e. those closer to the cores, arbitrate requests from each of the two cores they are connected to. The 2^{nd} lowest level may need to buffer up to 2 requests per link (one from each of the cores below); the 3^{rd} level up to 4 requests per link, and so on. As shown, the tree allows the use of small radix routers. Each router arbitrates among the two incoming links every cycle allowing at most one request to proceed to the next level of the tree. In the last stage, the tree can deliver up to one request per cycle, which is the speed at which L2 can accept requests. Typically, the L2 requires some cycles to serve a request, but does it in a pipelined way so a new request can be accepted every cycle. Finally, requests are served through a 1-to-N_c pipelined tree where only routing is needed (no arbitration is required).

Symmetrical arbitration policies provide homogenous bandwidth assignments to the tasks running in the different cores. We analyze how the policies used for MBPTA-compliant buses [12] apply to the tree NoC. In the following explanation we assume that all links have requests ready to be arbitrated. Later in this section we describe how contention must be modelled for pWCET estimation purposes.

- With RR each arbiter in the tree selects each of its incoming links alternatively in a deterministic manner. Given N_c cores, a request may be delayed by up to $N_c - 1$ requests from the other cores either because they are in front of it in a queue in the same link or because they are arbitrated first from another link in any router.
- LOT makes each arbiter select randomly one of the two links. As for RR, the latency for traversing the tree includes arbitration and queueing time. However, differently to RR, arbitration delay can be arbitrarily long with decreasing probabilities. The probability of waiting 0 cycles due to arbitration in one arbiter is $1/2$, 1 cycle $1/4$, 2 cycles $1/8$, and so on and so forth. In general, the arbitration delay in an arbiter is C cycles with probability $1/2^{C+1}$.
- RP is implemented by making each arbiter produce a random permutation of two elements, 0 and 1, every

two cycles, where 0 (1) indicates that the left (right) link is granted access. Thus, arbitration delay in one arbiter is 0 cycles with probability $1/2$, 1 cycle $3/8$ and 2 cycles $1/8$ [12].

Overall, any arbitration policy valid for the bus can also be adapted for the *pTNoC* and, as we show later, *pTNoC* outperforms the bus regardless of the arbitration policy used.

A. Factoring in NoC contention

pWCET estimates need to be time-composable, i.e. they should not vary regardless of the tasks running in the other cores. Time composable pWCET estimates greatly simplify incremental qualification and, by extension, help dealing with the increased timing verification and validation costs of CRTES. This is achieved by allowing each system component to be subject to formal timing validation in isolation and independently from other components.

In time-deterministic architectures measurement-based techniques rely on some hardware support, such as the *worst-case mode* [21]. In [21], each task is run in isolation at *analysis time* and hardware makes each request to experience the maximum, upper-bound, delay (UBD) that it may suffer during operation due to contention. In the case of the NoC this implies modeling the worst-case traversal time [20].

We use the *worst-case mode* for time-deterministic architectures and for time-randomized architectures when deploying round-robin arbitration since contenders could issue requests systematically with specific time intervals that lead always to the worst contention. Instead, for LOT and RP arbitration we create a *probabilistic worst-case mode* where the task under analysis runs in isolation in one core. In all other cores dummy requests are generated so that those cores always have one request in flight in the *pTNoC*, matching the maximum load a core can put on the tree. This is the worst case because, even if contenders issue requests systematically with specific time intervals, randomized arbitration policies produce random delays on those requests and thus, change – randomly – those time intervals. Thus, the worst scenario is the one we consider, with maximum load. To that end, dummy requests are simply discarded by the L2 cache, that immediately notifies the corresponding core so that it can issue a new dummy request in the next cycle. This creates a scenario with maximum contention where requests progress randomly through the *pTNoC*. Those requests suffer contention in the L2 due to its limited bandwidth. Overall, this produces the highest (probabilistic) contention that the task under analysis can suffer during operation.

V. Mixed-Criticality pTNoC

Safety-related functions are assigned a safety (assurance) level that defines the steps required in the design, verification and maintenance of the hardware/software components used by those functions, which inherit functions' safety level. Safety levels are described by specific standards in each domain. For instance, in the avionics domain safety standards such as DO-178B/C [24] classify components into 5 different levels, from Design-Assurance Level (DAL) DAL-A to DAL-E, where DAL-A stands for the most critical level.

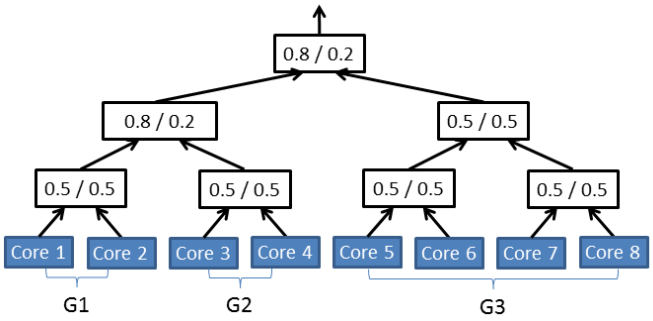


Figure 3: Heterogeneous guarantees in a tree NoC for an 8-core setup.

In the time domain, each safety-level requires providing a set of guarantees – which vary across levels – and sufficient evidence on the timing behavior of the tasks. For instance, the most critical real-time tasks (e.g., DAL-A in avionics or ASIL-D in automotive) may need strict performance guarantees provided in the form of a reliable WCET estimate.

Orthogonal to this, in each safety level, performance requirements may be heterogeneous depending on the type of application. While some critical real-time applications may need little (yet guaranteed) performance to operate on the little data read from some simple sensors, other critical real-time applications such as 3D Path Planning [27] for Unmanned Aerial Vehicles may require high performance to process large amounts of data. Likewise, low-criticality and non real-time applications may also have heterogeneous (average) performance needs. These heterogeneous performance needs translate into heterogeneous bandwidth allocation requirements when using the NoC. In this section we describe different approaches to deal with mixed criticalities, WCET, and average performance guarantees. Our solutions provide flexible means to satisfy the heterogeneous needs of mixed-criticality applications with a wide variety of performance requirements. For instance, let us assume we want to consolidate one DAL-A real-time task ($T1$) with commodious deadline, one DAL-B real-time task ($T2$) with a tight deadline, and six DAL-E (non-critical) tasks ($T3$ to $T8$) with no real-time constraints. In this scenario, our approach enables, for instance, allocating 20% of the *guaranteed* bandwidth to $T1$, 80% to $T2$, and remaining tasks are allowed to transmit requests opportunistically, but without timing guarantees.

A. Heterogeneous Bandwidth Assignments

Heterogeneous bandwidth allocation can be implemented by assigning different priorities to the requests of the different cores. In *pTNoC* we divide cores into several priority levels (layers). Priorities across layers may change as well as inside each layer, depending on the particular approach followed. We explore three such approaches.

(a) *Inter-layer priorities*. Each core is assigned a priority level with the requests from upper-layers prioritized over the requests of lower layers. Hence, guaranteed bandwidth can only be provided for the cores within the top layer. The bandwidth that the rest of the cores can enjoy is that left by higher-priority cores. For instance, in the avionics domain, DAL-E applications, which require only best-effort performance guarantees, are the only ones that could be assigned to a layer other than the top one.

Priorities can be implemented in each node of the tree keeping separate buffers for each priority level in those nodes where requests with different priorities may be arbitrated. The arbiter selects one request in this order: first high-priority requests from the selected link, followed by high-priority requests from the non-selected link, low-priority requests from the selected link and, finally, low-priority requests from the non-selected link.

(b) *Intra-layer priorities*. Cores in each priority layer can be provided heterogeneous bandwidth allocation, which can be implemented in different ways. In the case of RP we create permutations with as many slots as needed with high-priority links being assigned more slots. For instance, one could distribute bandwidth across links as shown in Figure 3 in which all cores are in the top priority layer: cores 1 and 2 (Group1 or G1) get the highest bandwidth: 0.32 (0.5 in the first arbiter and 0.8 in the second and the third). Analogously, cores 3 and 4 (G2) get 0.08 of the bandwidth each and cores 5 to 8 (G3) get 0.05 of the bandwidth each. In the case of a bus, the same approach can be implemented by increasing the number of slots for cores with higher bandwidth requirements.

(c) *Mixed-layer priorities*. The two previous approaches can be combined such that all requests from cores in higher-layer priorities are prioritized over the cores in lower-layers. And within each layer bandwidth can be allocated homogeneously or heterogeneously (intra-layer priorities).

Note that intra-layer priorities, despite they can be set to allocate the bandwidth in a non-homogeneous way, provide the same level of guarantees to all cores in the layer. In that respect, while intra-layer priorities only change the bandwidth allocation, inter-layer priorities change both bandwidth guarantees and allocation.

Overall, different degrees of bandwidth (from full bandwidth to no bandwidth) can be guaranteed to the different cores, thus enabling a flexible scheme to configure the NoC to the needs of tasks in different criticality levels.

B. Implementation Remarks

pTNoC provides special purpose control probabilistic bandwidth allocation (*pba*) registers, which can be written with standard instructions in most ISAs, e.g. *mfsr* (move from special register) and *mtsr* (move to special register). How these registers are set by the RTOS is explained with an illustrative example below.

In principle, each link of each arbiter needs as many queues as priority layers. One can restrict this to having only 2 priority layers to reduce hardware overheads, especially because only the highest priority layer has true guarantees and so it makes little sense having several low priority layers.

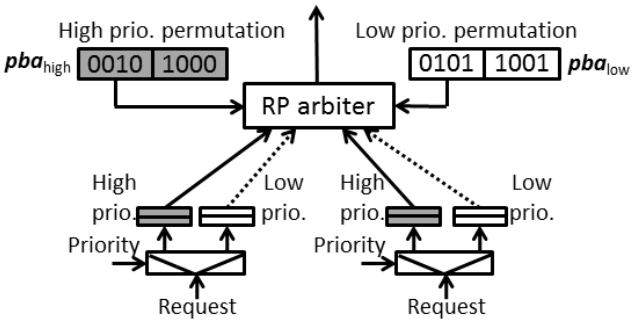


Figure 4: Example of an arbiter implementing inter- and intra-layer priorities.

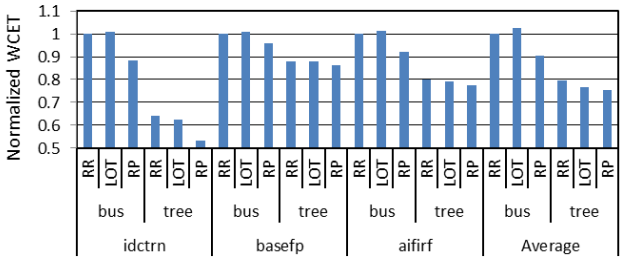


Figure 5: pWCET estimates for the 16-core bus and tree-based multicores normalized w.r.t. bus-RR.

Inter-layer priorities are implemented by prioritizing the requests in high-priority queues over the ones in low priority queues. Intra-layer priorities are implemented by allowing W-slot windows in the arbiters and using *pba* to determine which link (left (0) or right(1)) is granted access. For instance, let us assume $W = 4$-slot windows, which allows managing the bandwidth in 25% steps (e.g., 25% per slot) in each arbiter. In a 3-layer tree for an 8-core setup, this allows the bandwidth to range from 42.2% (75% in all arbiters, so 0.75^3) to 1.6% (0.25^3). To program the *pba* the arbiter creates 4-bit random permutations for each priority layer, see Figure 4. If only 2 priority layers are allowed, then 2 separate sets of *pba* registers are needed, one for each layer. In the example in Figure 4, for the high-priority traffic the left link is assigned 75% of the bandwidth, i.e. there are three 0's in each permutation out of four bits. For the low-priority traffic, bandwidth is divided evenly, i.e. the number of 0's and 1's is the same.

VI. Evaluation

We model a 8/16-core processor with pipelined in-order cores. Each core has separated first level instruction (I1) and data (D1) caches, a partitioned-across-cores L2 cache and main memory. I1 and D1 are 4KB, 8-way and 16B/line and

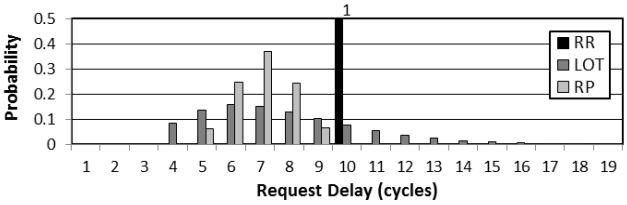


Figure 6: Request delay in a tree for an 8-core setup.

implement random placement and replacement policies [13]. The L2 is 256KB 8-way and also implements random placement and replacement policies. L2 is non-inclusive [14]. D1 uses write-through miss policy, I1 is read-only and L2 is write-back. Hit/miss latency is 1 cycle for I1/D1 and 3 cycles for L2. L2 is accessed either through *pTNoC* where each arbitration stage takes 1 cycle, or through a non-pipelined bus whose latency is 3 (4) cycles in the 8-core (16-core) case. The latency values used for the tree and the bus reflect the improved scalability of the tree where arbitration is performed on a distributed manner and the maximum link length is decreased [23]. For modeling this processor setup we used a simulator based on the SoCLib simulation framework [26]. For modelling the NoC we used gNoCsim [2], a powerful and configurable NoC simulator, which we integrated in SoCLib. As explained before, only the bus has been proven MBPTA-compliant so far. Moreover, other topologies (i.e. a mesh) do not provide any performance advantage for N_c-to-1 and 1-to-N_c communications and, as shown later, incur higher area and power overheads.

We make use of the proposal in [21] to make access to main memory be jitterless as needed for MBPTA. This further allows factoring out the variation of memory on pWCET estimates, making that NoC results can be better understood.

We use the EEMBC Autobench suite [22], which reflects current real-world demand of some automotive CRTES. pWCET estimation is performed with MBPTA for a cut-off probability of 10^{-13}. It is noted that similar trends are observed for other cut-off probabilities. For the purpose of measuring average performance we run each benchmark 1,000 times as part of a 8/16-benchmark workload composed of randomly selected EEMBC Automotive benchmarks. For instance, for the 8-core setup we included each benchmark 1,000 times in a list and randomly picked from there 8 benchmarks to build each workload until the list was empty.

A. Homogeneous bandwidth setups

First we evaluate *pTNoC* under an homogeneous bandwidth setup in which all cores are provided the same level of guarantees, i.e. all cores are in the same priority layer.

pWCET estimates. We compare the pWCET estimates obtained with MBPTA for the 3 arbitration policies, namely RR, LOT and RP; for both the bus and the tree with 16 cores. Results, shown in Figure 5, are normalized w.r.t. the bus implementing RR arbitration, hence, the higher the value for an arbitration policy the worse. Due to space constraints we show average results across all benchmarks as well as results for few individual benchmarks corresponding to cases where differences are large (`idctrn`), small (`basefp`) and close to the average case (`aifirf`). As shown, the tree NoC always leads to tighter pWCET estimates than the bus regardless of the arbitration policy used (20% for RR, 25% for LOT, 16% for RP). This occurs because the tree has much higher guaranteed throughput due to its pipelined fashion. When comparing the pWCET across arbitration policies for the tree, we observe that RP is the best choice because its maximum and average latency to traverse the tree is equal or lower than that for the other policies.

Average performance during operation. While WCET is the most important metric in CRTES, we also evaluate average performance when real NoC traffic is experienced rather than worst-case traffic. Given that real traffic is low in the NoC, contention occurs seldom. Thus, tree NoC average execution time is only 3.4% lower than that for the bus. Differences across arbitration policies are negligible (well below 0.1% for both the bus and the tree).

Per-request tree delay. Figure 6 shows the per-request delay histogram to traverse the tree for the different arbitration policies at analysis time. Results correspond to the `a2time` benchmark, although all benchmarks show very similar distributions. As shown, for the RR policy its worst-case traversal time of the NoC is 10 cycles. LOT takes around 7.8 cycles on average, however, it may take any number of cycles with decreasing probability. For instance, a latency of 15 cycles is experienced around 1% of the times. Finally, RP leads to the lowest average latency (7 cycles) and it never experiences a latency higher than 10 cycles (it is 10 cycles 0.5% of the times), which is its worst case. This is the reason for RP to outperform the other policies for the tree.

B. Heterogeneous bandwidth setups

In order to study the case of mixed-criticality systems with heterogeneous performance requirements, we evaluate different setups:

1) Inter-layer priorities. First we evaluate a setup where only one core is in the high-priority layer (E1-prio) and the rest of the cores are in the low-priority layer; and also a setup in which two cores are in the high-priority layer (E2-prio), in particular cores 1 and 2, while the rest of the cores are in the low-priority layer.
2) Intra-layer priorities. All cores are in the high-priority layer with different intra-layer priorities.

pWCET 8-core setup. In this case for the latter setup we consider 3 groups of cores with priorities as shown in Figure 3: G1 (cores 1 and 2), G2 (cores 3 and 4) and G3 (cores 5, 6, 7 and 8), with G1 having more bandwidth than G2 and G2 more than G3. We obtained pWCET estimates for each benchmark in each group (G1-3) during *analysis* time as explained in Section IV-A.

Figure 7 shows the pWCET estimates for the tree NoC for each arbitration policy normalized w.r.t. the homogeneous

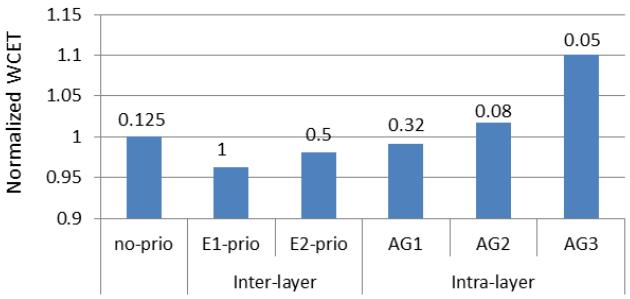


Figure 7: WCET estimates for the tree-based 8-core normalized w.r.t. non-priority RP case. Values on top of the columns show the guaranteed bandwidth in each case.

bandwidth (no-prio) case averaging values across all EEMBCs. The numbers on top of each column correspond to the (theoretical) guaranteed bandwidth in each case. I.e., in no-prio each of the 8 cores should get $1/8$ of the bandwidth and for the E1-prio and E2-prio case each core can get full and half of the guaranteed bandwidth respectively.

In the case of E1-prio and E2-prio results correspond to the benchmark(s) running with high priority. In both cases, we observe how our proposal effectively reduces the pWCET estimate for programs running in the core(s) in the top priority. For the case of intra-layer priority we observe that cores 1 and 2 in group G1 reduce their pWCET estimate by taking bandwidth from cores in groups G2 and G3, whose pWCET estimates are affected, specially for cores in G3. In general, the differences among bandwidth allocations are relatively small. The reason is that requests quickly get to the top arbiter since buffers in each link are large enough. Therefore, contention mostly occurs in that arbiter, thus creating relatively low variability across cores sharing the same link in the top arbiter. Moreover, intra-layer priorities in this top arbiter have a much larger impact than those in other arbiters due to the same reasons: requests mostly queue in the top layer.

pWCET 16-core setup. For the 16-core intra-layer setup we used a configuration similar to that for 8 cores as shown in Figure 8a, but modifying priorities in the top arbiter to illustrate its dominant effect. The delay histogram for requests in each priority group are shown in Figure 8b. pWCET estimates for this setup are shown in Figure 9. On top of each column we show the (theoretical) allocated bandwidth to each group.

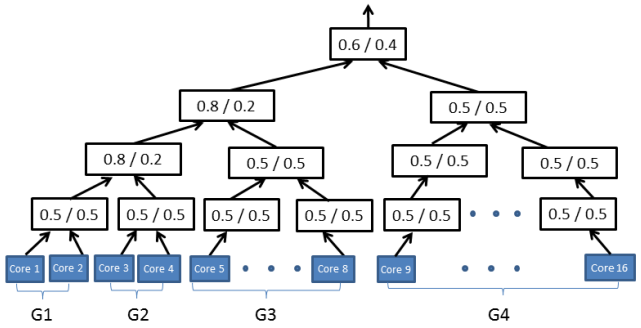
Trends are analogous to those in the 8-core setup. For instance, E1-prio provides the best pWCET estimate for the core with high priority, and it is followed by E2-prio for the 2 cores with high priority. Also, the groups of cores with highest G1 and lowest G4 have higher and lower pWCET estimates than the case of homogeneous priorities respectively. This correlates with the fact that the higher the bandwidth, the lower the pWCET estimate. Similar trends are observed for G2 and G3. The delay histogram in Figure 8b already shows that requests in G1 experience the lowest delay across groups and lower than no-prio, thus tasks in G1 obtain lower pWCET estimates than those for other cases. G2 and G3 obtain slightly worse pWCET estimates, as expected based on the delay histogram. No-prio performs a bit worse and G4 is the worst case among those, which also matches with the expectations based on the delay histogram.

It can also be observed that G2 and G3, despite having lower bandwidth than no-prio, have lower pWCET estimates. As explained before, this occurs because requests reach the top arbiter experiencing relatively low contention, but most contention occurs in the top arbiter where requests are queued until sent to L2. Therefore, bandwidth allocation in the top arbiter has much higher influence than that in the other arbiters. Since cores in G2 and G3 have higher bandwidth in the top arbiter than cores in no-prio, their pWCET is lower. Thus, pWCET differences in the 16-core setup are lower than in the 8-core case because priorities in the top arbiter are similar across links (0.6 vs 0.4) in the 16-core case, and unbalanced (0.8 vs 0.2) in the 8-core case. Part of our future work includes investigating other bandwidth management schemes with higher controllability.

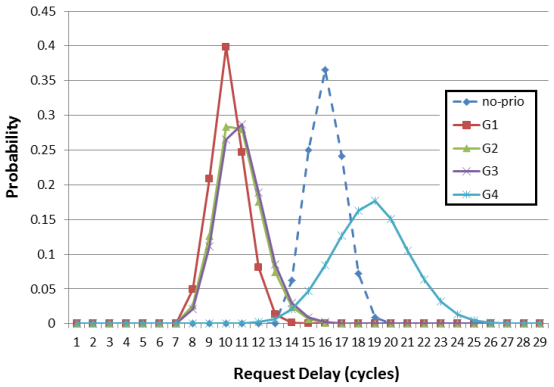
Average performance. We have also evaluated average performance when running under no-prio, inter- and intra-layer priority configurations. Figure 10 shows the average performance of all tasks in the workload for the 16-core setup. *pTNoC* effectively provides both, heterogeneous bandwidth guarantees and allocation, with very reduced impact on average performance. For the inter-layer configurations the slowdown is less than 5.5%. Differences come from the case where several programs with high memory bandwidth requirements fall in the same workload, some of them being in the top priority cores (thus creating plenty of high-priority traffic) and others starving systematically in the low priority cores. For the intra-layer configuration giving higher bandwidth to some cores negligibly improves average performance across all cores w.r.t. the no-prio setup. Average performance is just 2% better for G1 w.r.t. no-prio, 0.5% worse for G4 w.r.t. no-prio and 0.4% better across all groups.

C. Implementation and Energy Results

We have implemented the proposed tree design using the 45nm technology open source Nangate library [1] with Synopsys DC. We have used M1-M4 metallization layers to perform the Place&Route with Cadence Encounter of the different arbitration nodes. To determine the placement of the different arbitration nodes we have followed the approach proposed in [23] to minimize link length. Table I summarizes the implementation results for 8- and 16-core trees using two priority layers. Results for a bus implementation are also shown for comparison purposes. Area and delay results in the table are normalized w.r.t. an 8-core mesh NoC. As shown, tree NoCs require lower area than the 2D mesh network because the tree NoC implements only the resources required to perform all-to-one communication. However, as expected, the area required for implementing the tree is larger than



(a) 16-core Mixed criticality setup



(b) Request delay for a 16-core tree

Figure 8: Heterogeneous bandwidth guarantees in a 16-core tree NoC.

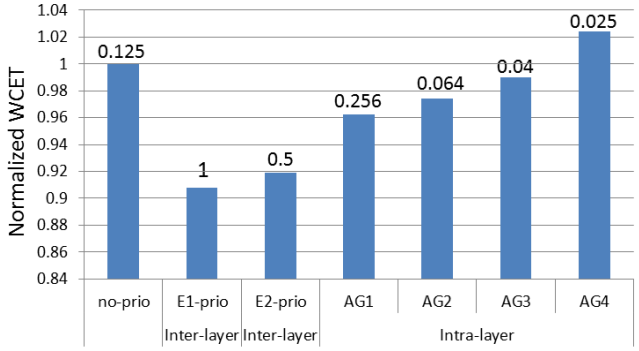


Figure 9: pWCET estimates for the 16-core setup for different arbitration policies normalized w.r.t. non-priority RP case.

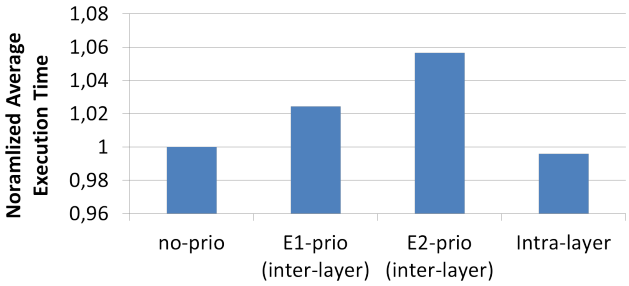


Figure 10: Average execution times for 16 cores for different arbitration policies.

that required for the bus. Critical path delay results shown in the table determine maximum achievable frequency of the different NoC designs. Maximum achievable frequency mainly depends on two factors: network size and arbitration inputs. For the bus and the tree, network size impacts operation frequency as it determines the maximum link length required in the floorplan. The number of arbitration inputs or contenders is given by the number of layers used to perform the arbitration. As shown in Table I *pTNoC* shows good scalability up to 16 nodes unlike the bus that shows a delay $1.67\times$ and $1.47\times$ worse than the *pTNoC* for 8 and 16 cores, respectively. Moreover, while the number of arbitration layers penalizes maximum achievable frequency, its impact can be contained if only two arbitration layers are used as proposed in this paper. Note that results shown in the table are for the case of RP. However, results for RR and LOT arbitration, also computed but not shown in the table, are roughly the same.

We have also computed energy values when executing several applications with the proposed NoC design. In particular, we have computed values for EEMBC benchmarks and for one synthetic benchmark. The synthetic benchmark is a corner case where high number of misses in L1 occur to create high contention in the NoC. Differences in energy measurements for regular benchmarks (EEMBC) are imperceptible while RP behaves slightly better (1.5%) in the case of the synthetic benchmark. The conclusions we extract from these measurements is that the very low implementation overhead of RP has negligible impact in energy values even in the least favorable scenario (low contention) while the reduction in the execution time provided by RP provides slightly lower energy values in the most favorable scenario (high contention).

VII. Conclusions

MBPTA has emerged recently as a powerful method to derive WCET estimates for critical tasks in safety-related systems. Multicore designs providing the properties needed by MBPTA have been presented in the literature, but they rely on a shared bus to communicate cores and memory, and do not fit well mixed criticalities.

We have presented a MBPTA-compliant tree NoC, *pTNoC*, that outperforms buses in the 8- and 16-core setups evaluated. *pTNoC* enables the realization of mixed criticalities on multicores by managing different guarantee levels and bandwidth assignments with minimum impact on average performance.

Table I: Results of the synthesis normalized w.r.t. 8-core mesh values. Absolute values are also provided for completeness.

	Area					
	8-core			**16-core**		
	mesh	tree	bus	mesh	tree	bus
Relative	1	0.54	0.38	2.30	1.55	1.24
Absolute ($10^4 \mu m^2$)	18.4	10.0	7.0	42.3	28.6	22.9

	Delay					
	8-core			**16-core**		
	mesh	tree	bus	mesh	tree	bus
Relative	1	0.90	1.33	1	0.96	1.61
Absolute (ns)	0.83	0.75	1.1	0.83	0.80	1.34

Our results for 16 cores show 16% to 25% average WCET reductions for our tree w.r.t. the bus for different arbitration policies. Further WCET reductions of up to 9% are obtained when using priorities for mixed criticalities. Also, our results show that *pTNoC* incurs low area and energy costs.

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] *The NanGate 45nm Open Cell Library. http://www.nangate.com.*

[2] *NaNoC design platform. http://www.nanoc-project.eu.*

[3] J. Abella et al. Heart of gold: Making the improbable happen to extend coverage in probabilistic timing analysis. In *ECRTS*, 2014.

[4] ARINC. *Specification 651: Design Guide for Integrated Modular Avionics*, 1997.

[5] L. Benini et al. P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In *DATE*, 2012.

[6] F. Cazorla et al. Upper-bounding program execution time with extreme value theory. In *WCET workshop*, 2013.

[7] R. Charette. This car runs on code. In *IEEE Spectrum online*, 2009.

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

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

[10] G. Edelin. Embedded systems at thales: the artemis challenges for an industrial group. In *Presentation at the ARTIST Summer School in Europe 2009*, 2009.

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

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

[13] L. Kosmidis et al. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.

[14] L. Kosmidis et al. Multi-level unified caches for probabilistically time analysable real-time systems. In *RTSS*, 2013.

[15] L. Kosmidis et al. Measurement-based probabilistic timing analysis and its impact on processor architecture. In *DSD*, 2014.

[16] S. Kotz and S. Nadarajah. *Extreme value distributions: theory and applications*. World Scientific, 2000.

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

[18] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *EDCC*, 2012.

[19] M. Panic et al. On-chip ring network designs for hard-real time systems. In *RTNS*, 2013.

[20] M. Panic et al. Parallel many-core avionics systems. In *EMSOFT*, 2014.

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

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

[23] A. Roca et al. Enabling high-performance crossbars through a floorplan-aware design. In *ICPP*, 2012.

[24] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[25] M. Schoeberl et al. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *NOCS*, 2012.

[26] SoCLib. -, 2003-2012. http://www.soclib.fr/trac/dev.

[27] T. Ungerer et al. parmerasa – multi-core execution of parallelised hard real-time applications supporting analysability. In *DSD*, 2013.

[28] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, 2007.

[29] F. Wartel et al. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *SIES*, 2013.

[30] F. Wartel et al. Timing analysis of an avionics case study on complex hardware/software platforms. In *DATE*, 2015.

[31] 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.