Reconciling Time Predictability and Performance in Future Computing Systems

Francisco J. Cazorla*§, Jaume Abella*, Enrico Mezzetti*, Carles Hernandez*, Tullio Vardanega†, Guillem Bernat‡
* Barcelona Supercomputing Center, Barcelona, Spain
† Universita Degli Studi di Padova, Italy
‡ Rapita Systems Ltd., United Kingdom
§ IIIA-CSIC, Spain

***Abstract*—The demand for guaranteed, hence *predictable*, performance in the real-time systems domain is projected to increase by several orders of magnitude in the next few years, while its weight in the mainstream market is on the rise. Satisfying this need in a cost-effective manner compels system architects to use high-performance hardware units, which however have disruptive effects on current timing verification practice. This paper presents low-overhead solutions for hardware design and timing analysis to help attain the desired level of predictable performance in all application domains with assurance needs, also contributing to the universal pursuit of performance guarantees.**

Keywords: C.3.d Real-time and embedded systems; C.0.d Modeling of computer architecture

I. Introduction

The '90s witnessed convergence between the high-performance processor market and low-power (embedded) systems, resulting in high-performance low-power design solutions, extensively used in mobile devices. The present era shows similar signs of convergence between high-performance low-power mainstream products and the real-time embedded market [7] in the quest for high guaranteed performance. On the one hand, mainstream devices increasingly incorporate software functionalities that take part in critical systems (e.g. health monitoring), and consequently inherit the sustained performance needs of the latter. On the other hand, modern real-time systems include critical and complex functions (e.g. decision-making in robotic applications, autonomous vehicle operation in automotive, railway and aerospace) that have steadily increasing high-performance needs.

The level of guaranteed (hence predictable) performance required to sustain the execution of those critical functions is therefore projected to rise to unprecedented highs. For example, ARM forecasts for the automotive domain[1] maintain that advanced driver assistance features will require a 100-fold increase in computing performance by 2024. Getting there at competitive costs will necessarily yield very aggressive, parallel and heterogeneous computer designs such as, e.g., NVIDIA's ISO26262-compliant Xavier processor. However, the more stateful resources are deeply embedded in high-performance processors, the more complex the problem of asserting performance guarantees that also hold in the worst case, the bigger the risk of incomplete or restricted information, and the more pessimistic the results. Complexity grows to reflect the inordinate increase in the state space, and impairs certification. Pessimism grows, with worthless results, owing to the need to treat missing information conservatively.

This trend challenges the ability of measurement-based timing analysis (MBTA) – the most common and accessible technique for industrial use – to deliver the sought assurance of performance predictability in the form of Worst-Case Execution Time (WCET) bounds. In particular, creating test scenarios in which the application runs under extreme execution conditions that upper-bound those that can occur at operation is a hard challenge. The low-level hardware resources that are highly contended in high-performance processors (e.g., buses, request queues and caches) make it inordinately difficult for the user to ascertain whether the execution conditions (e.g., contention load) observed during testing correspond to what can occur during operation. Hence, the question whether the execution-time observations made in the analysis are *representative* of the extreme situations that can arise at operation is very difficult to answer. This quandary severely undermines the reliability of the proffered results and withholds users from convincedly transitioning to high-performance hardware when assurance of predictable performance is required.

This paper lifts some of the impediments that prevent the more widespread adoption of high-performance processors in critical real-time systems and, symmetrically, of timing analysis methods in domains where performance predictability is becoming a first-order requirement. To this end, we present two complementary approaches to achieve low-cost high-confidence and tight performance predictions with MBTA, for real-time systems running on high-performance hardware.

[A] Hardware Designs that Provide Increased Observability. This approach augments the processor hardware with the capability of producing Performance Monitoring Counters (PMC) data to expose key indicators of the internal timing behavior of selected hardware resources. This wealth of information allows determining whether the execution conditions experienced during the test campaign represent the worst-case situation. The *increased observability* yielded by Performance Monitoring Units (PMUs) expressly designed to expose hard-to-predict variable timing behavior, combined with big-data

[1] https://www.arm.com/about/newsroom/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade.php

analysis methods to treat the PMC read-outs, increases the quality of MBTA results dramatically.

[B] Hardware Designs that Expose Execution-Time Variability. This approach allows the user to rest assured that the execution-time measurements taken during analysis do capture the full span of variability that can occur during operation. This assurance is obtained with hardware support that enables the user to control the time variability of key performance contributors (e.g. caches), at negligible costs for space and performance. This solution is complemented with a timing analysis method that uses probabilistic reasoning to predict the extreme variability of the program's WCET.

Solution **[A]** increases the observation power offered by the processor hardware, and lowers the effort intensity of discerning the spectrum of variability observed during timing analysis. Solution **[B]** changes the processor hardware to cause the critical sources of variability to span their full extent in manners that can be more easily and conclusively captured during analysis. Their combination enables the achievement of considerably higher levels of guaranteed performance in advanced processor platforms.

II. Common Principles

MBTA studies the system's timing in analysis scenarios, to determine upper-bounds to the worst-case execution-time behavior that may occur at operation. MBTA's challenge is to construct analysis-time scenarios that help compute WCET estimates that upper-bound operation-time behaviour. This evidently requires ensuring that all factors with bearing on the *execution conditions* that the program may incur during operation are duly considered in the analysis. In fact, the factors that originate from low-level hardware resources are far more difficult to get at for the user than those that proceed from the software. This paper addresses the former challenge, with solutions that entail simple changes to hardware design and MBTA methods, which achieve quality results without sacrificing performance. On those grounds, we maintain that the adoption of the following design principles yields an effective application of MBTA to high-performance systems.

Performance predictability as a design requirement. Factoring predictability as a first-class citizen in the hardware design up front strikes a much better cost/benefit ratio than addressing it later, when predictability-geared modifications may not prove practical, affordable or – worse even – sufficient. Having failed to consider predictability early in the design of some of its processors, ARM tried to reduce contention interference by allowing second-level (L2) cache space to be partitioned across cores, yet continuing to share the queues that stored the pending L2 cache misses. This design decision was flawed [11], as it let any single core clog and starve the others by hoarding those queues. Other architectures were conceived with predictability in mind from the outset (e.g., PRET [1], T-CREST [2]), yet at the cost of radical differences from COTS designs, thus hitting a most formidable obstacle to adoption by chip manufacturers.

Performance-preserving predictability solutions. Evidence suggests that hardware vendors would not trade high average performance for time predictability, as sacrificing the former hinders market penetration. Conceivably therefore, support for predictability can make it in hardware design only as long as it incurs affordable overhead for chip area, power budget, validation complexity, and harms neither programmability nor software portability. Accordingly, we advocate the use of predictability features, e.g. for cache placement/replacement and arbitration, that are configurable at will (e.g., by simple enabling/disabling) on account of specific traits of the application. This vision contrasts with approaches that seek lesser resource sharing and more task isolation by way of ad-hoc hardware designs, such as scratchpads, special caches, or NoC modifications that extend beyond arbitration and require additional signals and protocols in the router [2].

For the management and arbitration of shared resources, for example, we vouch for solutions that narrow the gap between worst-case and average performance, specifically without sacrificing the latter. Static XY-routing for NoCs and round-robin or random arbitrations combined with smart virtual channel allocation are good instances of that notion, as they ease capturing worst-case scenarios while also assuring a worst-case behavior not too far apart from the average one. Conversely, adaptive routing and dynamic virtual channel allocation solutions increase that gap, often earning only marginal improvement to the average case [10].

Renounce the absolute WCET. Blindly accounting for theoretical worst-case scenarios in complex heterogeneous platforms is likely to lead to exceedingly pessimistic estimates. For example, contending shared hardware resources to saturation may cause massive degradation of performance, and therefore lead to insanely high WCET values. A more sensible concept seeks to keep a high *overall* utilization, which yields good average performance, while avoiding saturation situations by design, thereby improving the worst case.

WCET analysis should leverage resource utilization information to renounce the one-and-only absolute upper-bound and concentrate instead on the (least pessimistic) conditions that do upper-bound those that can arise during operation. This notion is often referred to as seeking *partial time composability*, which aims to provide WCET estimates that relate to *specific utilization bounds* for shared resources. To secure the system as verified, those utilization bounds shall then be enforced during operation, by hardware or software means, to prevent applications from exceeding their assigned quota.

A promising approach to derive performance predictions under a postulated contention load employs *micro-benchmarks* [5], small user-level programs designed to cause programmable contention on specific target resources (e.g. caches and buses). Running the application against the micro-benchmarks allows exploring the potential impact of different levels of contention, and facilitates the derivation of performance guarantees in those load scenarios.

Tracing and instrumentation support: Collecting hardware events (e.g. execution time) as needed for WCET estima-

tion should be done without affecting program behavior owing, e.g., to instrumentation instructions. While modern processors provide powerful tracing facilities, e.g. NXP's Nexus or ARM's Coresight, high-speed tracing technology must evolve to avert scalability issues as the core count increases.

III. Increased Observability

This approach extends PMUs with predictability-aware performance monitoring counters (PMCs). With this approach, sketched in Figure 2 with blue arrows, MBTA is fed with (possibly large volumes of) PMC read-outs. In that manner, not only MBTA can correlate PMC indicators with execution-time measurements, but also ascertain whether the execution conditions incurred at operation are indeed covered in the analysis scenarios, and thus better reason about representativeness.

A. Hardware Extensions: predictability-aware PMCs

Current processors embed sophisticated debug-and-statistical PMUs (aka DSU) that support a very large number of PMCs, hundreds in the IBM POWER7 or the NXP P4080. PMCs are normally used for profiling the application, improving average performance, or debugging. Hence, the information that they provide tends to ignore event types that are deemed not relevant to those purposes: this is often the case of the events that are critical to WCET estimation. For instance, typical PMCs report the number of requests sent to the bus and the bus usage by a given core, but do not report *how long* that core waits for the bus. Indeed, with existing PMCs, one can for example determine whether high bus contention may cause the load-store instructions to clog the load-store unit. Yet, knowing for how many cycles a program was stalled by clogging of the load-store unit is not sufficient for predictability analysis. Instead, one would need PMCs that break down the load-store unit stall time into the fraction of it due to the program's own memory activity and the fraction due to contention in the bus, cache, memory, etc. PMCs for the latter do not generally exist.

While predictability-aware PMCs are scarce in current processors, they are rather easy to implement as simple extensions of existing counters. As changes in the PMCs are very unlikely to affect average performance, it should not be difficult to persuade chip manufacturers of the importance of predictability-aware PMCs for performance predictability, and eventually favor their adoption.

B. Example: AMBA-Bus Delay Modelling (Side Column)

The Advanced Microcontroller Bus Architecture (AMBA) is one of the most used bus interfaces. Under the AMBA protocol, the arbitration process involves several hardware blocks (the arbiter and one or more masters) and several signals. Recently, a method has been devised to monitor contention in shared AMBA buses [8]. The method monitors existing signals of the AMBA bus, as shown in Figure 1: `HBUSREQ`, which each master sets to request the bus; and `HGRANT`, which the arbiter sets to identify the master that is granted access to the bus in each cycle. If forwarded these

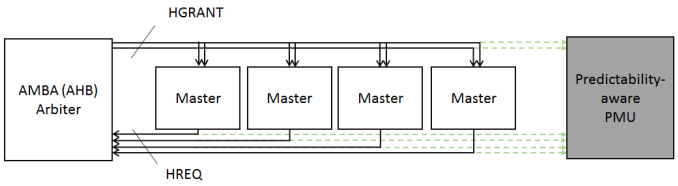

Fig. 1. PMU snooping of AMBA Signals (dotted lines) for contention delay monitoring. Example with four masters.

signals, the PMU can tell the number of cycles that a master i is held by master j, by simply counting how long `HBUSREQ` equals i *and* `HGRANT` equals j. The PMU can be equipped with a matrix of $N_m \times N_m$ counters, for N_m masters: entry $[i, j]$ then denotes the cycle count that master i is waiting for master j. This simple mechanism allows determining how long one task is delayed by another task, which is key to task consolidation, seeking assignments that cause the lowest contention, which in turn allows deriving WCET estimates in consolidated scenarios instead of pathological ones.

C. MBTA Extensions: Big-data and Statistical Analysis

Predictability-aware PMUs allow MBTA to capture much richer information for observation runs than mere execution times. As MBTA usually requires numerous test runs, with abundant quantities of PMC values being read per run from the PMU, big volumes of (heterogeneous) data are likely to be collected, which makes a very clear case for big-data analysis extensions to MBTA.

Combining the predictability-preserving features outlined in Section II with the increased-observability capabilities presented in this section allows *attenuating* the jitter incurred by real-time software programs instead of striving to eradicate it with radical design changes (as pursued by other approaches). The residual jitter is exposed via PMCs to the timing analysis tool that can reason on the factors that cause it, and consequently advise the system engineer on how to attenuate them.

Big-data analysis can be used to extract the most sensitive information out of the observation measurements, telling what are relevant in them. Regression methods, extreme value analysis, data decomposition, and other statistical techniques can then be used to predict the incidence of individual factors of influence on the program under analysis. That approach would conceivably yield tight WCET estimates with quantifiable levels of confidence.

D. Extension to Static Timing Analysis

Static Timing Analysis (STA) [12] is designed to seek the assurance of a single-valued absolute WCET result, however pessimistic. STA, which has critical dependence on accurate and trustworthy knowledge on the processor internals, has proven its potential with hardware designed with predictability in mind [1], [2], and would certainly also benefit from the inclusion of predictability-aware PMCs. Yet, the need for open, documentary information is a serious vulnerability when the processor parts are subject to IP restrictions. Moreover, such

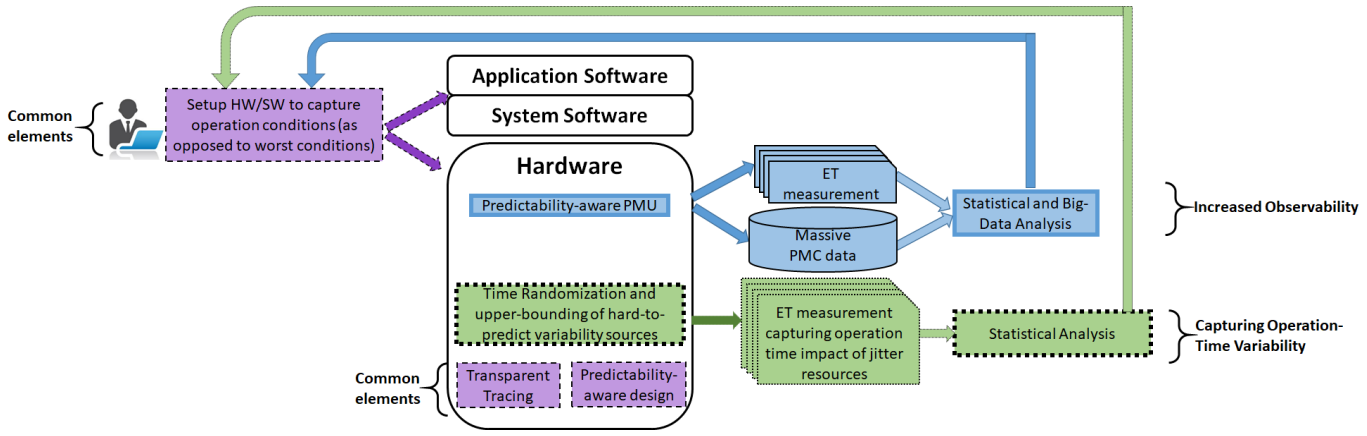

Fig. 2. Schematic view of the increased-observability approach (blue arrows) and the variability-capturing approach (green dotted arrows) for increased performance predictability. Common parts are shown in violet (dashed lines). 'ET' stands for execution time.

features as multi-level caches, decoupling buffers, manycores, NoCs and accelerators, which have been in use for years in mainstream products, have such an intricate internal behavior and tangled interactions with execution-time behavior that far exceed the capacity of state-of-the-art abstract modelling. The width of the state space to be comprehended and the quantity of state variables in it, form a *complexity wall* that cannot be overcome other than at the cost of massive loss of information, and increase of pessimism.

STA will continue to be the reference timing analysis solution for more restricted scenarios and lower-performance systems in which accurate abstract modelling is still possible. However, on the whole, STA can hardly be regarded as a practicable solution for the quest of performance predictability across all industrial domains. MBTA instead is bound to remain the most viable option in that regard.

IV. Capturing Operation-Time Variability

This approach strives to cause the full range of variation in the execution-time behavior that the application can incur at operation to occur, *without* the need for user intervention, in the measurement observations taken at analysis. Whereas the increased-observability solution enables measurement observations to capture the impact of jittery hardware resources, it requires further effort on the timing analysis side to claim that what was observed is fully representative of what can happen during operation. This approach, instead, selectively injects time randomization in jittery hardware resources to allow quantifying the *representativeness* of operation-time variations covered in measurement observations[2]. With those hardware modifications, MBTA can be soundly augmented with probabilistic reasoning, to allow quantifying the claim of representativeness, while its application procedure remains rather simple. This notion, which we further articulate in the sequel, is captured in the green-colored elements of Figure 2.

[2]As discussed in Section IV-A, this form of predictability-geared randomization can be implemented with negligible impact on average performance.

A. Hardware Extensions: Randomization and Upperbounding

To increase *representativeness*, a family of hardware designs has been recently proposed, which bases on two key properties: time randomization and time upper-bounding [9].

Upper-bounding. Hardware resources that exhibit low jitter (e.g. the floating-point unit) are modified so that they can be forced to work at their highest latency during analysis, causing measurement observations to upper-bound the operation-time jitter effect of their use.

Time randomization is used, at both analysis and operation, for hardware resources with high jitter, since upper-bounding their jitter, as done for resources with low jitter, would incur excessive pessimism. Randomization injected in those resources causes all extent of variability that they can incur to equally occur during analysis *and* operation, which in turn enables quantified claims of probabilistic coverage (i.e., representativeness) to be made. As execution time exhibits random variation, statistically sufficient measurement observations made during analysis suffice to represent the extreme timing behavior that may occur during operation, and probabilistic reasoning can be applied to MBTA. This is better illustrated in side column IV-B.

The feasibility of these hardware modifications has been demonstrated in an RTL-level FPGA prototype of a COTS multicore that is now in commercial offering[3].

B. Side Column: Handling Cache Jitter

The memory addresses at which the program code and data are located (the memory mapping) determine the cache sets to which they are assigned (the cache layout). Different memory mappings result in distinct cache layouts, with varying effect on the program's execution time.

With applications integrated out of parts developed by multiple suppliers, timing analysis – which presides to software dimensioning – must be performed as early as possible for each part, well before the final mappings are known. Hence,

[3]http://www.gaisler.com/index.php/products/processors/leon3

the challenge that caches present to MBTA is to assure that the program configurations used in the measurement observations do capture the cache layouts that may occur after final integration (which are of course unknown). MBTA addresses that challenge by seeking to single out the memory mappings that yield cache layouts that lead to higher execution times. If it manages to do so, then the cache impact is duly factored in the analysis, else the results may be optimistic. However, hoping to find the global worst-case cache layout before final software integration experimentally is not an option. Hence, no confidence can be had that the test campaigns performed at unit and integration level serendipitously find the worst-case cache layout for all software programs of interest. Moreover, even minor changes to the program code, which may happen across incremental development, (or rarer changes to link directives) can effect the memory layout, yielding execution times that may invalidate the prior findings of MBTA. These difficulties may degrade the confidence in the WCET estimates product of MBTA, below the thresholds of regulated domain practices, and therefore compel the affected industrial users to renounce caches altogether.

Cache randomization combines random replacement (which many architectures use) and random placement [6]. The latter breaks the dependence between memory location and cache placement. A random input value – changed across program runs at times that do not interfere with system operation – determines the (random) placement function. In that manner, the actual address in memory becomes irrelevant to cache placement and so, to execution time variation. Every single run yields a random cache mapping that corresponds to a random sample in the whole population of possible cache layouts. As randomization stays enabled during operation and therefore prevents systematic pathologies, this strategy allows making a sound probabilistic argument on the coverage of the cache-layout problem space attained in measurement observations. With cache placement, addresses are manipulated with a random seed[4] provided prior to program run and changed after it. A permutation network or hash logic (see the right picture in Figure 3) places each address in a random and independent set by changing the (random) seed across runs.

This approach frees MBTA users from the obligation to model or control the whole space of execution-time variations, and allows measurement observations to be collected in a much more black-box manner.

C. MBTA Extension: Probabilistic Reasoning on Predictability

Time randomization coupled with the application of probabilistic reasoning to timing analysis changes the WCET estimate concept from a single fully-assured limit value into a probability distribution that represents the maximum residual probability at which a WCET threshold can be exceeded.

[4]Pseudo random number generators exist that deliver repetition-free series long enough to prevent potential correlation of events in statistically significant spans. The produced numbers can be used to generate random placements and replacements in caches or for random arbitration policies.

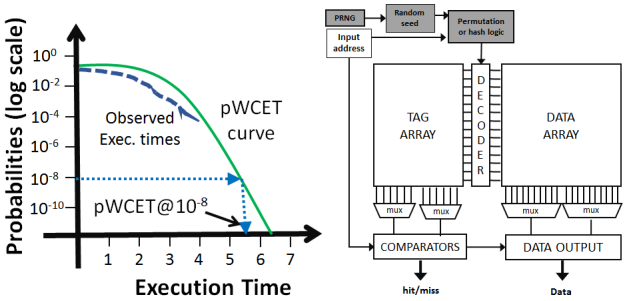

Fig. 3. Example of pWCET estimation (left) and a randomized cache (right)

Such mutation is not unspeakable for a twofold reason. First, no current timing analysis technique can provide absolute guarantees that the computed WCET bound will never be exceeded, simply because all methods ultimately depend on precarious inputs, whether worst-case scenarios for use with MBTA, or precise hardware timing models for STA, or assured flow facts for both. Second, the quality of those analysis inputs (and their effect on reliability and tightness of results) cannot be quantified in general, which resorts users to qualitative judgment. Probabilistic reasoning instead allows quantification, much like what it does to model the appearance of certain types of hardware faults in electronic components.

Within a probabilistic mindset, MBTA can use measurement observations to derive *probabilistic* WCET (pWCET) estimates, so that any given (extreme) execution-time bound is attached a probability of exceedance, which represents the highest probability that it can be topped (see left part of Figure 3). In practice, the probability of exceedance is set so low that the risk of overrunning the pWCET can be deemed irrelevant by the applicable safety regulations. The Extreme Value Theory and other methods that predict the tail of a probability distribution are fit for this use, and only need appropriate tailoring to the pWCET problem case. Interestingly, MBTA can use those methods in a black-box manner as the hardware modifications described in this paper ensure by construction that the measurement observations taken at analysis are representative of the execution-time distribution that can occur at operation. In fact, this approach holds even in the face of IP restrictions on hardware details as long as IP owners design their components following the principles needed to attain representativeness, namely time upper-bounding and randomization, as described in [9]. Still, there is no practical way for IP users to verify whether those principles hold, so the degree of confidence on the pWCET estimates depends on whether IP owners effectively adhere to those principles.

A measurement-based probabilistic timing analysis process starts by collecting observations that implicitly factor in the potential timing variability. Subsequently, specialized statistical tests are applied to assure that probabilistic methods (e.g., Extreme Value Theory) can be used for pWCET estimation. Those tests may check independence and identical dis-

tribution across measurements, and the quality and sufficiency of the data to deliver a prediction. At that point, a pWCET estimation can be obtained. See [3] for more details.

D. Impact on Average Performance

Randomization is often perceived to negatively affect average performance. In actual fact, however, its penalty largely depends on the granularity level at which randomization is applied. Recent solutions have shown it to have marginal impact on performance and negligible cost on hardware. Some variants of random placement and replacement policies for caches have been shown to stay within 1% of the performance of deterministic caches [6] on multi-level cache hierarchies. The key factor in those proposals stems from the observation that randomization can be performed at relatively coarse granularity. Earlier proposals operated on cache lines (i.e., for a dozen bytes), which taxed average performance significantly. More recent designs [6] operate on cache way boundaries (i.e., for kilobytes), which preserves spatial and temporal locality, and therefore prizes average performance, and still suffices for probabilistic reasoning. For arbitration, instead, experiments on the BlueGene Torus Interconnection [4] have shown that randomization provides performance results very close to the state of the art. Ad-hoc policies can of course be developed specifically for given traffic profiles to achieve slightly better results, but in the absence of specialization, the average performance of randomized arbitration is competitive.

The evaluation of time-randomized multicores [6] with avionics, space and railway case studies has shown that pWCET estimates are typically within 20% of the actual performance on time-deterministic counterparts. Hence, bounds are lower than the usual practice on time-deterministic architectures of adding a 20% engineering margin on top of the highest observed execution time. Moreover, time-randomized architectures enable scientific reasoning and quantitative evidence supporting pWCET estimates, thus facilitating certification against safety-related standards.

V. Conclusions

The demand for unprecedented levels of predictable performance compels a rising proportion of application domains that exceed traditional real-time systems, to use high-performance hardware. This trend confronts MBTA with hard challenges. First, the absolute worst conditions for software and hardware sought for classic WCET determination can become so pathological that any upper bound on them is too high to be useful; much more sensible and relevant is to define an estimate of the WCET for the set of configuration states that the system can really exhibit. Hardware and software design as outlined in this paper can favour system configurations that lead to measurable and reasonable, hence usable, estimates. Second, accounting for the execution-time variability of hardware resources with bearing on timing at operation is becoming exceedingly difficult for the average user. This paper explores two approaches to address this problem: (1) hardware designs to provide control knobs that improve observability and allow MBTA to extend the representativeness of analysis-time observations to operation-time situations; (2) design means to cause the timing variability of jittery hardware that can occur at operation to also arise during analysis without the need for user control. The combination of those two approaches spares the need for exceedingly detailed timing models of the target hardware and therefore scales to complex processors affordably. Both solutions offer a solid baseline to support the quest for performance predictability across a wide range of application domains.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Carles Hernandez 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] Precision Timed Machines. http://chess.eecs.berkeley.edu/pret.

[2] Time-Predictable Multicore Architecture for Embedded Systems. http://www.t-crest.org/.

[3] J. Abella, M. Padilla, J. Del Castillo, and F.J. Cazorla. Measurement-based worst-case execution time estimation using the coefficient of variation. *ACM Trans. Des. Autom. Electron. Syst.*, 22(4):72:1–72:29, June 2017.

[4] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas. Blue gene/l torus interconnection network. *IBM Journal of Research and Development*, 49(2.3):265–276, March 2005.

[5] G. Fernandez, J. Jalle, J. Abella, E. Quiñones, T. Vardanega, and F.J. Cazorla. Increasing confidence on measurement-based contention bounds for real-time round-robin buses. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 125:1–125:6, New York, NY, USA, 2015. ACM.

[6] C. Hernandez, J. Abella, A. Gianarro, J. Andersson, and F.J. Cazorla. Random modulo: A new processor cache design for real-time critical systems. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2016.

[7] High-Performance Embedded Architecture and Compilation. HiPEAC vision. 2011, 2013, 2015 and 2017.

[8] J. Jalle, J. Abella, E. Quiñones, L. Fossati, M. Zulianello, and F.J. Cazorla. AHRB: A high-performance time-composable AMBA AHB bus. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*, 2014.

[9] L. Kosmidis, E. Quiones, J. Abella, T. Vardanega, C. Hernandez, A. Gianarro, I. Broster, and F.J. Cazorla. Fitting processor architectures for measurement-based probabilistic timing analysis. *Microprocessors and Microsystems*, 47, Part B:287 – 302, 2016.

[10] M. Panic, C. Hernández, J. Abella, A. Roca, E. Quiñones, and F.J. Cazorla. Improving performance guarantees in wormhole mesh noc designs. In *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pages 1485–1488, 2016.

[11] P.K. Valsan, H. Yun, and F. Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, April 11-14, 2016*, 2016.

[12] 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.