On the Convergence of Mainstream and Mission-Critical Markets

Sylvain Girbal[†], Miquel Moretó[⋆,⊙,‡], Arnaud Grasset[†], Jaume Abella[‡], Eduardo Quiñones[‡], Francisco J. Cazorla[‡,Φ], Sami Yehia[Ψ,1]

[†]Thales Research & Technology, France
[⊙]International Computer Science Institute, Berkeley, USA
[Φ]Spanish National Research Council (IIIA-CSIC), Spain
[⋆]Universitat Politecnica de Catalunya, Spain
[‡]Barcelona Supercomputing Center, Spain
[Ψ]Intel Corporation, USA

ABSTRACT

The computing market has been dominated during the last two decades by the well-known convergence of the high-performance computing market and the mobile market. In this paper we witness a new type of convergence between the mission-critical market (such as avionic or automotive) and the mainstream consumer electronics market. Such convergence is fuelled by the common needs of both markets for more reliability, support for mission-critical functionalities and the challenge of harnessing the unsustainable increases in safety margins to guarantee either correctness or timing. In this position paper, we present a description of this new convergence, as well as the main challenges and opportunities that it brings to computing industry.

Categories and Subject Descriptors

B.8 [**Performance and Reliability**]: Miscellaneous; D.2.8 [**Metrics**]: Performance measures

General Terms

Measurement, Performance, Reliability

Keywords

Mission Critical, High Performance, Quality of Service

1. INTRODUCTION

The computing market has been dominated during the last two decades by the convergence between high-performance (HP) computing market pursuing for power efficiency and the embedded/mobile market pursuing for more performance and functionalities. This evolution has lead to: 1) several open standards to control the balance between low power and high performance in current processors, 2) several APIs to control the low-power hardware features from the software layers (e.g. Operating System, OS) and 3) processor core designs that can be easily retargeted to provide different performance-power design points depending on the target market.

In this position paper we present our witnessed new type of convergence between the mission critical market (such as the avionic, automotive, healthcare and robotic) and the mainstream consumer electronics market. Such convergence is fuelled by the increasing requirements of the mission-critical market for performance and functionalities and the growing needs of the consumer electronics market (such as mobile phones) to embed more critical functionalities and interact with other critical systems (such as cars and health monitoring systems). At the same time, we are confronted with the challenge of harnessing the unsustainable increases in safety margins to guarantee either correctness or timing (worst-case execution time margins). We also show that this new convergence brings opportunities and challenges in the way hardware and software have to be designed to deal with mission-critical requirements.

In Section 2 we show the main motivation behind the new convergence after revisiting the past convergence between low power and high-performance markets. Section 3 shows challenges and opportunities brought by this new convergence, while Section 4 presents novel timing analysis initiatives. We conclude in Section 5.

2. CONVERGENCE OF MAINSTREAM AND MISSION-CRITICAL MARKETS

We witness a new type of convergence between the the mainstream and the mission-critical (MS-MC) markets resulting from the previous convergence between the mobile and high-performance markets, as shown in Figure 1.

2.1 The Past Convergence: Low Power and High Performance Markets

In the 1990s, the mobile market was a niche market guided only by low-power constraints with very low performance and functionality requirements. Three factors motivated the convergence between mobile market and the mainstream market, see Figure 1. First, the increase in performance and functionality requirements of the mobile market made low-power processors including high-performance features. Second, in the 90's and 00's processors increased their performance at the expense of a rapid increase in power dissipation. For instance, Intel processors increased power from

[1] This work was done while Sami Yehia was a Research Engineer at Thales Research and Technology.

DAC '13, May 29 - June 07 2013, Austin, TX, USA.

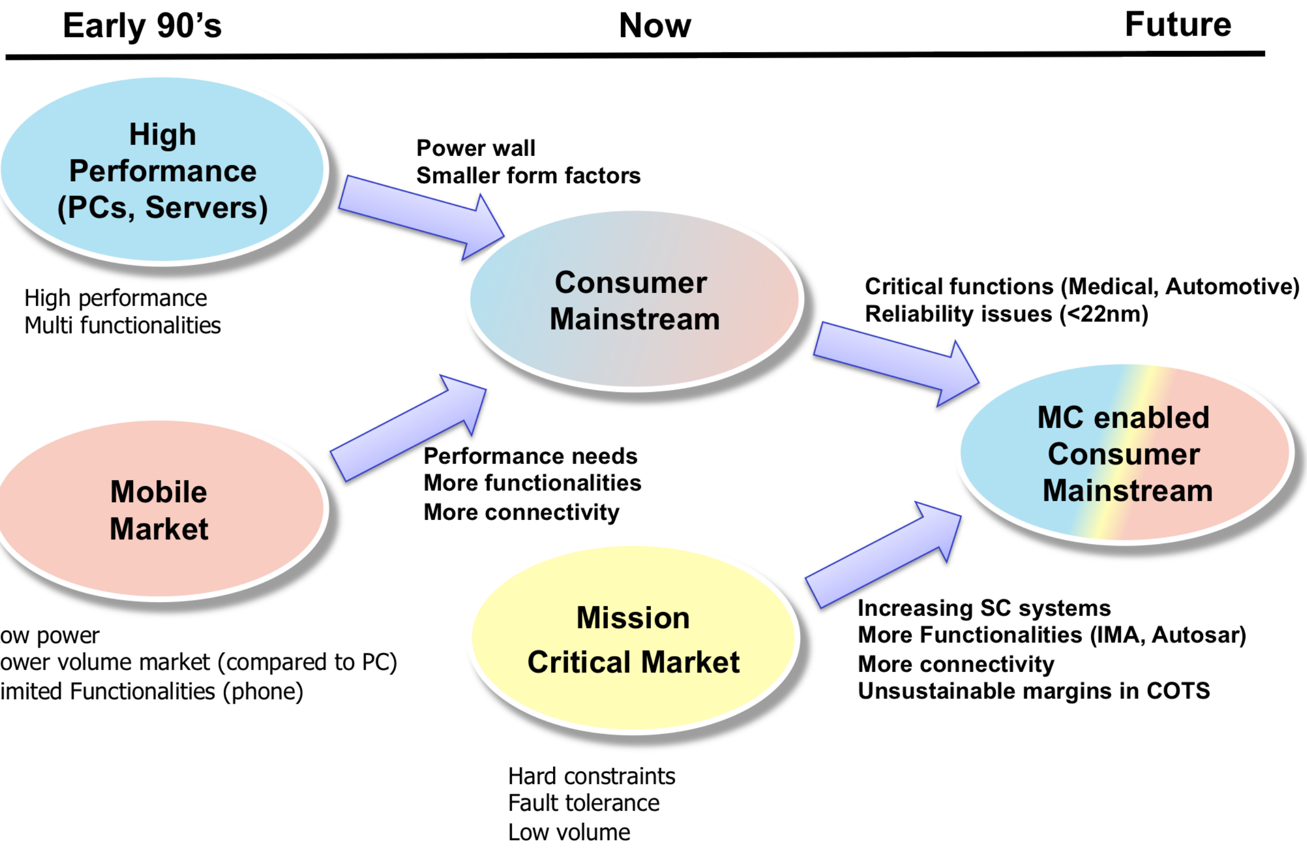

Figure 1: The past convergence between mobile and high-performance markets as a reference to explain our devised new convergence between mainstream and mission-critical markets.

15W (Pentium [10]) to 115W (Pentium4 [11]) in less than 10 years. Limitations in heat dissipation as well as the increase in the energy cost led to a U-turn in high-performance processor design. Low-power features were increasingly incorporated to provide the highest performance under a given power envelope, e.g. Intel released Pentium M, right after Pentium 4, that dissipated up to 27W. And third, processors need several years and astronomical costs to be designed, verified and validated. Hence, reusing designs across domains drastically decreases costs.

After several years of convergence, several standards have been defined to balance power and performance in all high-performance processors. For instance, the Advanced Configuration and Power Interface (ACPI) [15] defines several low-power states as well as means to allow the OS to control the current state, so that different tradeoffs between performance and power can be used dynamically. This enables the same processor design being used in mobile and HP markets despite their different constraints.

Recently, market convergence has extended also towards the lowest performance/power range of the mobile segment. Processors initially designed for handheld devices such as smartphones are widely used in high-performance segments such as tablet and netbook segments. For instance, some servers have been released soon based on, for instance, the ARM11 and Intel Atom processors [12] given that they can provide higher performance/density (e.g., within the volume and power envelope of a particular server).

2.2 Mission-Critical Application Market

The mission-critical system designers have traditionally relied on low performance components to address the diverse, but limited in number, service requirements of mission-critical applications. In recent years this design approach is challenged, and found to be limited and cost-ineffective, by the growing need for more functionalities and performance, while still ensuring stringent reliability and safety requirements as well as tight hard real-time constraints. We observe an upcoming increase - unprecedented in diversity and level - in performance requirements in the avionic, automotive and medical domains, between others. For instance:

- In the avionic domain future generation of navigation systems (four-dimension trajectory and N-Fly zones management systems [7]) in aircrafts, 5th generation cockpit (with the possible introduction of single-crew capable airliners [5]), and collision avoidance are all functionalities with increasing performance requirements. This can be inferred from Figure 2, where code size is used as a proxy for the demand for computational power [6].
- In the automotive domain, future driver assistance systems for vehicles requires a supercomputer (*high-performance computing*) level of performance for processing data from cameras, radar, LIDAR and other sensors to detect and decide about warnings and in critical situations autonomous breaking or steering. Both

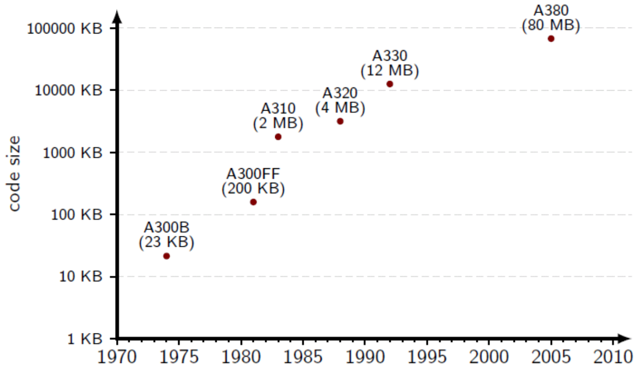

Figure 2: Growing of embedded software size in avionics

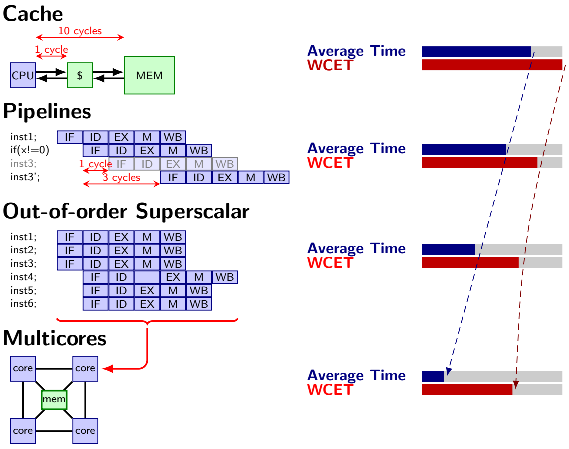

Figure 3: Evolution of average execution time and WCET under different processor setups with increasing high-performance features

domains are overarched by stringent requirements of lower fuel consumption, CO_2 emissions, with high-speed real time processing and fault-tolerance.

- In the medical market, the microcontrollers used in the implantable devices, pacemakers, infusion pumps and digital hearing aids have also significant fault-tolerance, low-power and high-performance requirements.

These required levels of performance can only realistically be realised by deploying high-performance hardware acceleration features such as caches or multiple cores per chip and smaller technology nodes (i.e. smaller transistor sizes). These technology advances enable (i) reducing energy consumption and temperature, and (ii) integrating more hardware functionalities per chip, which in turn enables running more system functionalities per chip, thus reducing overall SWaP (system size, weight and power consumption) costs.

However, because of the increasing complexity of those features, analysis of the worst-case scenarios for mission-critical applications is extremely difficult, especially with the emergence of multicores in the embedded market. One typical example in the time domain, is the calculation of worst-case execution time (WCET) [16]. The extra complexity of emerging hardware complicates the analysis timing behavior of mission-critical systems in two ways.

First, there is an increasing gap between the actual WCET and the average best effort average performance, as depicted in Figure 3. The use of complex hardware such as caches, pipelining, superscalar execution and multicores, reduces much more average performance than WCET. For instance, in the case of a multicore with a hardware shared resource, the execution time of a task is much smaller when no other cores use that shared resource. If the tasks running on the other cores, introduce high load in the shared resource, each task is going to suffer a significant slowdown due to these inter-task interferences. While on average this effect may not be significant, the actual WCET of the task is much more affected.

Second, the WCET estimates that can be provided by current timing analysis tools are becoming increasingly pessimistic with respect to the actual WCET. This is so because current analysis approaches, mainly static timing analysis, require acquiring sufficient knowledge of all factors of influence, at hardware and software level, on the timing behavior of the program. The addition of high-performance features, increases the complexity of acquiring the knowleged required to carry out the analysis. In order to appreciate this phenomenon, consider a cache model with a Least Recently Used (LRU) replacement policy. The accuracy in predicting the hit/miss outcome of a memory access depends on knowing the full sequence and addresses of the previous memory accesses made by the program up to the point of interest. This information is needed to build a complete and correct representation of the cache state so as to determine the hit/miss outcome of memory accesses with sufficient accuracy. Any reduction in the available knowledge, e.g. when the addresses of some memory accesses are unknown, leads to a rapid degradation of the tightness of the WCET estimation. In fact, partial knowledge can lead to results as inaccurate (i.e. inordinately pessimistic) as those obtained with no information at all. As a result, many of the analysis techniques in the literature rely on over-provisioning the architecture. At software levels all the hardware resources that cannot be analyzed, such as the cache are disabled. Aggressive architectural techniques such as branch prediction and data path forwarding are also disabled to ease computing WCET estimates [16]. Also, some manufacturers just turn off all but one core in a multicore platform if highly-critical system components exist.

It is also the case that the relatively low volume of the mission-critical market makes the development of specific processors meeting the specific requirements of mission-critical applications extremely costly because of the high Non-Recurring Engineering (NRE) costs.

2.3 Mainstream Consumer Electronics

The microprocessor systems in mainstream or consumer electronics market are also facing challenges. At the hardware level, semiconductor technology poses serious challenges in the design of future microprocessor systems. First the shrinking technology nodes (14nm in 2013-2014 according to Intel [3]) will make the processing elements subject to variability constraints, wear-out and soft errors such that the reliability of the basic computing elements will no longer be guaranteed. Thus, reliability will be a first-order design constraint similar to power and performance. Faults at the transistor and circuit levels will be more common than ever such that microprocessor and system designers will have to provide additional fault tolerance features at several design layers.

At the application level, we foresee an increasing demand in autonomy, decision-making and artificial intelligence. This demand will very likely be overarched by stringent requirements in term of safety, quality of service (QoS) and hard real time constraints. Examples of such applications are future domestic robots, healthcare applications [4], intelligent cars, augmented reality, human++ applications [14]. Also the increasing demands in connectivity will naturally lead to a situation where mobile devices directly connect with more mission-critical systems such as cars and medical devices. Hence, providing mobiles with mission-critical capabilities will enable a new type of applications for end users [9].

Our view is that most future applications will require some form of hard real-time behavior for at least part of their operation. For domestic robots, cars, planes, telesurgery, and Human++ implants, it is clearly necessary to impose limitations on the delay between sensing and giving the appropriate response. For parallel applications, it is important that all processes running in parallel have balanced execution time in order to maximally exploit the parallel resources of the platform, and limit the synchronization overhead. Especially on heterogeneous multicores, being able to accurately estimate execution times is crucial for performance optimization.

2.4 The Next Convergence

Overall, we observe a second challenging convergence trend, akin to the convergence between the computing PC market and mobile market during the last two decades (See Figure 1), between mainstream consumer electronics and mission-critical markets. On the one hand, the performance requirements of current and future mission-critical applications, as well as their connectivity with the less critical mobile electronic devices, make the use of low performance or over-provisioned architectures not a viable solution anymore. On the other hand, applications such as health care monitoring will start to hit the mainstream market and the increasing safety margins for reliability issues start to make the available transistors and available performance more and more difficult to exploit. One key common aspect in mission-critical and mainstream systems is that they are both relying on over-provisioning some of their resources in order to make some guarantees.

3. CHALLENGES OF THE MC AND MS MARKETS CONVERGENCE

Next, we attempt to pave the way toward achieving convergence between the mission-critical and the consumer electronic markets so as to reduce the high NRE cost of the mission-critical market and provide efficient mission-critical capabilities to the general purpose market.

To achieve such a convergence we need to enrich existing mainstream processing architectures with capabilities that provide non-functional guarantees to address the mission critical needs, without over-provisioning the processing architecture or diminishing its efficiency. In other words, we want to provide the user a (Quality of Service) QoS according to her or his functional and non-functional requirements.

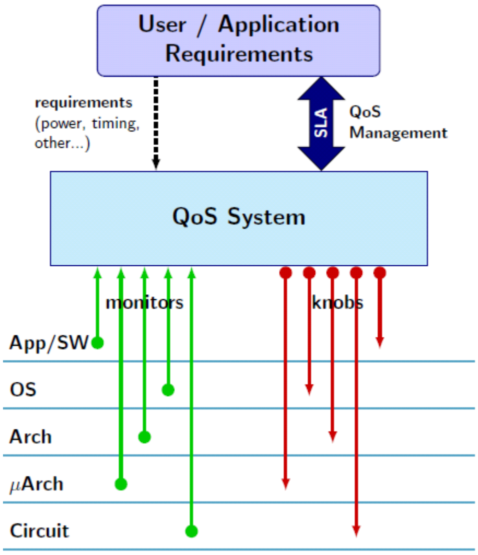
User / Application Requirements
requirements (power, timing, other...)
SLA
QoS Management
QoS System
monitors
knobs
App/SW
OS
Arch
μArch
Circuit

Figure 4: Integral QoS Approach

3.1 User Requirements

Because QoS objectives are often application specific, the envisioned QoS approach provides an efficient and general interface that can satisfy QoS objectives over a range of applications. It is required to define the necessary API and language support to express the requirements. A common factor for all type of applications is that the application requirements provided to the QoS system have to be architecture independent. This removes the need of the application programmer to provide a different set of metrics for every target architecture, guaranteeing performance and requirements portability. In particular our envisioned QoS system combines the requirements coming from mission-critical-like applications and general-purpose applications, possibly running them concurrently (mixed criticality systems). In the case of mission critical applications, we need much stronger guarantees and a pure best effort approach is not sufficient. Several approaches have previously advocated for the introduction of non-functional properties in programming languages, especially domain specific ones [8, 13]. Those languages consider timing characteristics, which are of paramount importance in the development of the application as well as the architecture.

Historically QoS standards have been widely used in networked systems in order to ensure high-quality performance of data-flow, especially for real-time multimedia applications, without reactively expanding or over-provisioning the networks. The advent of multicore, reliability issues and the demands of mission-critical applications make the adoption of QoS and resource reservation mechanisms in multiprocessors systems the only way to efficiently use these systems and meet these mission critical demands.

3.2 QoS Service-Level Agreements

Establishing a Service-Level Agreement (SLA) between the application and the OS is the key to offering a guaranteed QoS to the user, especially for mission critical systems. For example, a user may want to guarantee the completion of a task within a specified deadline. In such a situation, the application will attempt to establish an SLA with the

OS before starting the task. The OS will accordingly allocate and block the necessary resources needed to guarantee the completion of the task within the deadline. In the case where the OS is unable to allocate the necessary resources to offer such guarantee, the application will be notified before starting the task and can trigger a backup mechanism. The advantage of such a scheme is that the designer can analyze its application based on a reasonable assumption of available resources, environmental conditions and performance degradation (due to wear-out effects) and not on the assumption of the worst-case scenario where no resources are available for the task or the assumption of an over-provisioned system that will guarantee the availabilities of resources.

3.3 Example of non-Functional Requirement: Specifying Timing QoS Requirements

In mission-critical systems, temporal aspects of their behavior are part of their specification. That is, their correctness depends on the correctness of their functional behavior and also its timing behavior: the time at which the results are produced. This is so because mission-critical systems interact with their environment, which also changes with time.

The software component of real-time functions are usually implemented as an endless loop that makes some data sensing, processing and actuation over an actuator. In each iteration of the loop, a new instance of the program/task is generated. These instances are called jobs. Jobs have a deadline prior to which its execution must end.

Timing properties of mission-critical systems are usually specified by means of the concept of *deadline misses*. In particular, the percentage of deadline misses stands for the percentage of instances that end after their deadline. Depending on the particular system, the system *value function* may be sharp or progressive. Sharp functions are those where the success is an “all or nothing” function. Conversely, progressive functions have a decreasing value after the deadline.

We follow the approach presented by Bernat et al. [2] for specifying the QoS time requirements of systems that can tolerate occasional losses of deadlines, such as mission-critical. QoS time requirements cannot be adequately specified with a single parameter, for example, with the percentage of deadline misses. This is so because the percentage of missed deadlines is an average measure and does not allow to determine the temporal behavior of the missed deadlines. For instance, for a task with 10% deadline misses, we lack the information about whether the task misses one deadline every ten, ten consecutive deadlines every 100, etc. This temporal frequency of misses is critical, as some real-time systems are more sensitive to consecutive deadline misses.

Therefore, it is necessary to provide metrics to express the QoS timing requirements of a system exact distribution of missed deadlines. This could be in the form of hit/miss patterns, e.g. (1 1 1 0 1 1 1 0) where ’1’ represents hit and ’0’ miss; or by specifying the minimum number of consecutive hit deadlines between two missed deadlines [2]. This can be further combined with progressive value functions.

3.4 Challenges and Opportunities at SW level

At software level, a promising research area are QoS API, Middleware and architecture designs, that allow applications or users to be provided a guaranteed level of service by efficiently using the available resources, according to the above-mentioned SLA, and without over-provisioning the architecture to achieve the required level of service. This represents a major departure and different philosophy from previous design approaches that in general address QoS with narrow scope that is neither cost-effective nor scalable. We advocate for holistic approaches across computing layers in which QoS becomes part of the design approach at each computing layer. Similarly to Instruction Set Architectures (ISA) that provide the abstract interface between low level programming and the hardware implementation, interfaces at each layer must be able to express QoS requirements. QoS compliant designs will need to provide means for bidirectional communication for monitors to observe and knobs to control the system, as shown in Figure 4. Some primitive notions of QoS principles are present in existing systems (such as load balancing, dynamic voltage scaling, etc. [1]) but are not pervasive enough and not meant to provide QoS. In other words we advocate for defining the necessary services required for meeting non-functional requirements (or properties), the necessary micro-architectural, architectural, middleware and system support to efficiently implement these services, how these services are expressed in the application and how they are deployed over all layers down to the hardware.

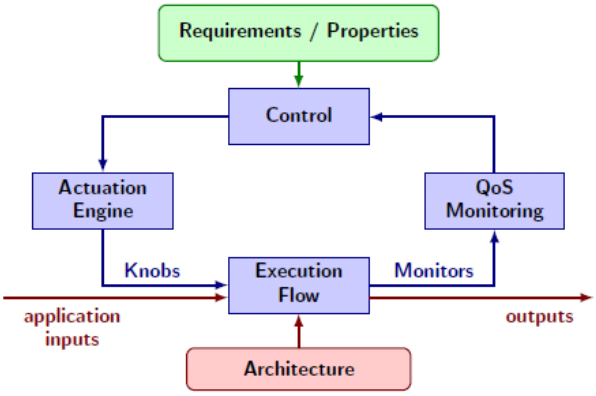

Figure 5: Detailed view of the Integral QoS Approach

3.5 Challenges and Opportunities at HW level

Monitors are used to collect the different parameters and state of the system periodically and verify that the tasks progress according to the requirements, as illustrated in Figure 5. The concept of monitors generalizes the standard concept of Performance Monitoring Counters (PMCs) in several aspects. Monitors provide much richer information than PMCs (e.g. information about the interaction between tasks), cover several metrics (not just performance) and provide feedback that allows deducing part of the future behavior of the running applications.

Depending on the behavior of the system, the knobs regulate the different tasks and system behavior to ensure a guaranteed QoS. Monitors and knobs can exist at different layers of the system. For example hardware counters at the architecture levels can be used to monitor cache misses, job queues at the OS level can be used to monitor the system load, and temperature sensors at the circuit level can be used to monitor the temperature. Similarly knobs can consist of migrating tasks (OS level), shutting down some cores for power saving (architecture level) or performing some DVFS actions at the circuit level.

4. TIMING ANALYSIS INITIATIVES

As mentioned in Section 2, there is an increasing performance demand in the mission-critical market, such as the avionic, automotive or the medical domain. However, the benefits that the additional computational power of advanced hardware/software features can potentially provide cannot be fully realized unless accompanied by analysis techniques that enable their use. In particular, in this section we focus on providing some approaches to analyze the timing behavior of future mission-critical/mainstream systems.

Several studies work in that direction by proposing different techniques to analysis the time behavior of applications running in a Commercial Off-The-Shelf (COTS) multicore processor having a varying degree of hardware shared resources. Annex I describes several of the main software-only solutions proposed in that line.

In addition to changing timing analysis techniques, new advanced hardware/software can be delivered with a twofold objective: (i) Improving performance and (ii) Enabling simple means to provide QoS guarantees. In that direction, Probabilistic Timing Analysis (PTA) has emerged as an attractive solution to respond to the demand for trustworthy timing analysis in the critical real-time embedded system domain in the face of more performance aggressive processors. Compared to conventional static timing analysis, PTA requires much less knowledge about the program and the processor, which allows users to obtain high performance and timing guarantees with little effort. This is a significant advantage considering that (1) market competition drives programs to become increasingly complex and (2) processor manufacturers increasingly use black-box IP components. In Annex II, we provide more details about PTA techniques and their advantages for the MS-MC convergence.

5. CONCLUSIONS

We envision a new type of convergence between the mission-critical market and the mainstream consumer electronics market: both markets have common needs for increased support for mission-critical functionalities together with an increase in safety margins to guarantee either correctness or timing. To reach these goals future computer designs should implement an integral QoS approach in which QoS becomes part of the design approach at each computing layer. The hardware has to provide features that enable software analyzing and controlling processor internal resource allocation through intelligent knobs and sensors. These hardware features could be enabled from the processor, depending on the criticality and performance needs of the target market, so that the same processor design can be used for different mission-critical requirements. The software stack should be able to provide interfaces that allow user QoS requirements to reach the appropriate layer as well as the proper knobs to provide non-functional guarantees to address the mission-critical needs.

We have shown how QoS requirements should be specified and measured using metrics beyond percentage of deadline misses to better assess the non-functional requirements of a MS-MC system.

Finally, providing guarantees on the timing behavior, as a representative of the non-functional requirements of the system, will also introduce changes aimed at controlling the interaction between software components. The probabilistic approach offers an interesting alternative path to simplify the analysis of the timing behavior of complex MS-MC systems.

Acknowledgements

M. Moretó, J. Abella, F. J. Cazorla and E. Quiñones have been partially supported by the PROARTIS FP7 European Project under grant agreement number 249100, the Spanish Ministry of Science and Innovation under grant TIN2012-34557 and the HiPEAC Network of Excellence. M. Moretó is supported by a Fulbright/MEC Fellowship. E. Quiñones has also been supported by the Spanish Ministry of Science and Innovation under the grant Juan de la Cierva JCI-2009-05455. The authors thank all the anonymous reviewers for their constructive comments and suggestions.

6. REFERENCES

[1] A. Bartolini et al. A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. In *GLSVLSI*, 2010.

[2] G. Bernat, A. Burns, and A. Llamoso. Weakly hard real-time systems. *IEEE Trans. Comput.*, 2001.

[3] M. Bohr. Silicon technology leadership for the mobility era, September 2012. Intel Developer Forum.

[4] S. Cantrill. Computers in patient care: the promise and the challenge. *Commun. ACM*, 53(9):42–47, 2010.

[5] A. Doyle. Thales outlines thinking on single-crew cockpits. *Flight-globale*, July 2010.

[6] G. Edelin. Embedded Systems at THALES: the Artemis challenges for an industrial group. In *ARTIST Summer School in Europe*, 2009.

[7] European Organisation For The Safety of Air Navigation. Study report on avionics systems for the time frame 2007, 2011 and 2020. *Eurocontrol*, 2004.

[8] T. Henzinger, B. Horowitz, and C. Kirsch. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91:84–99, 2003.

[9] M. Holzbock et al. Evolution of aeronautical communications for personal and multimedia services. *IEEE Communications*, 2003.

[10] Intel Corporation. Datasheet: Intel pentium processor 75/90/100/120/133/150/166/200, 1997.

[11] Intel Corporation. Datasheet: Intel pentium 4 processors 570/571, 560/561, 550/551, 540/541, 530/531 and 520/521 supporting hyper-threading technology, 2005.

[12] Intel Newsroom. Chip shot: Intel Atom processors codenamed ”Centerton” to power first HP's extreme low-energy production servers, 2012.

[13] E. A. Lee. Computing needs time. *Commun. ACM*, 52(5):70–79, 2009.

[14] J. Penders et al. Human++: Emerging technology for body area networks. *VLSI-SoC: Research Trends in VLSI and Systems on Chip*, pages 377–397, 2008.

[15] White Paper. Intel® Turbo Boost Technology in Intel® Core$^{\text{TM}}$ Microarchitecture (Nehalem)Based Processors, 2008.

[16] R. Wilhelm et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3), 2008.

APPENDIX

A. TIMING ANALYSIS OF COTS MULTICORE PROCESSORS

The mission-critical domain industries have been using Commercial Off-The-Shelf (COTS) architectures to reduce the non-recurring engineering costs (NRE) and time-to-market (TTM) [1], while trying to deal with the variability in runtime to satisfy real-time constraints.

The recent shift to multicore in the embedded COTS market should effectively allow the industry to cope with the exponential increase in performance needs. However this shift worsens the runtime variability problem as contentions on shared hardware resources bring new variability sources.

A.1 Timing Analysis Techniques

A common practice to guarantee the deadlines of a safety-critical application is to determine the application Worst-Case Execution Time (WCET). This WCET computation usually relies on analysis tools based on static program analysis tools based on extensive program analysis [17, 14], detailed hardware model, as well as measurement techniques through execution or simulation [6]. However, those WCET analysis tools are not currently able to determine the exact WCET: they are providing an upper bound, introducing some safety margins as depicted in Figure 6.

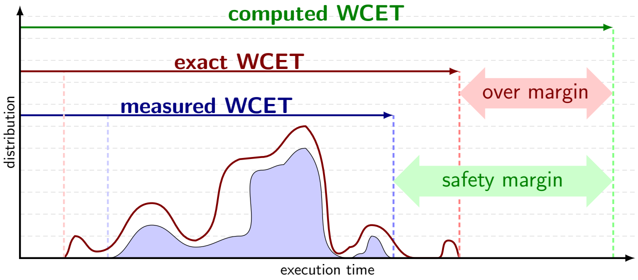

Figure 6: WCET estimation and the over-estimation problem

A measurement-based approach for single-threaded architectures computes the WCET estimate by multiplying the longest observed execution time by a safety margin, usually provided by an expert with understanding of both the target hardware architecture and the reference applications [11]. This method has been successfully used in the past to determine WCET estimate of applications running on single-threaded processors with a moderate difference between measured and computed WCET.

A.2 Analysis Technique Limitations

A direct extension to the measurement-based analysis presented in Figure 6 to multicore COTS processors would consist of running several reference applications simultaneously on the same processor, and monitoring the execution time for each application in the workload.

During the last decades, despite all the improvements in the WCET estimation domain [8, 3], the over-estimation remained mostly constant as the predictability of the architecture decreased [17], making the use of WCET analysis tools difficult for real industrial programs running on multicore COTS architectures [7, 11] for the following reasons: (1) WCET analysis of real industrial programs with a vast number of possible execution paths is a challenging task. (2) The implementation of accurate hardware models for new architectures requires a significant effort and a detailed description of the hardware, which is not always available. (3) Possible interference on shared hardware resources among co-running tasks significantly increases the complexity of timing analysis, forcing it to have a full knowledge of co-running tasks at software level, and detailed resource contention models at hardware level.

A.3 Quantifying Variability on Multicores

Before proposing alternative timing analysis techniques coping with multicore COTS, it is important to quantify the runtime variability for such systems.

To perform this study, we used the Freescale P4080DS platform detailed in Table 1, a complex 8-core architecture, with private L1 instruction & data caches and a private L2 unified cache. The eight cores are connected through a proprietary CoreNet Fabric to two shared 1MB L3 caches, each connected to a DDR memory controller.

Core	8 Power Architecture e500mc at 1.5GHz
Pipeline	7-stage pipeline, superscalar, out-of-order
Distributed L1 caches	32kb, 8-way associative, 64-byte line, PLRU
Distributed L2 cache	128kb, 8-way associative, 64-byte line, PLRU
Shared L3 cache (x2)	1MB, 32-way associative, 64-byte line, PLRU
Memory controller	Two DDR memory controllers
Interconnect	CoreNet Coherency Fabric

Table 1: Freescale P4080 specifications

Figure 7 illustrates the runtime variation for some applications of the MiBench benchmark suite [4] during 1000 successive execution iterations on the P4080 platform. Each application runs standalone on the barebone platform, the other cores being idle to minimize variability sources.

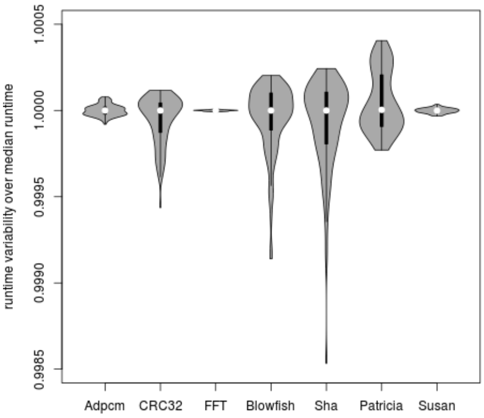

Figure 7: Variability of mibench applications running standalone on the 8-core P4080 platform

Each violin plot from Figure 7 is showing the runtime distribution around the median runtime for this benchmark. The width of a violin plot for a specific runtime is proportional to the population ratio with this particular runtime. If Figure 7 shows quite different variability schemes, the variation remains very small (below 1%).

Figure 8 quantifies the impact of co-running benchmarks. Each mibench application is now run concurrently with 2 benchmarks dedicated at stressing the interconnect resource.

Each violin plot of Figure 8 illustrates the runtime variation of each application around the previously computed

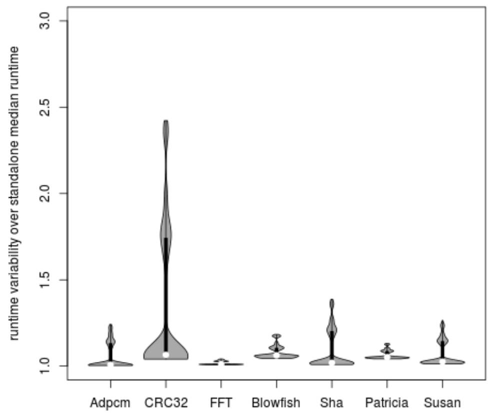

Figure 8: Variability of mibench applications running with 2 co-running benchmarks

standalone median. If the average variation remains quite low, the impact on the worse case is much more significant with an average variation of x1.34 (+35%) and a maximum variation of x2.42 (+140%) to be compared with the previous variation of 1%.

A similar study [12] from EADS shows on the same architecture that using actual WCET analysis techniques forces the industry to multiply the WCET value by a value close to the number of cores being used. This is of the same order of magnitude than the overall potential performance gain, and will provide no performance benefits over a single-core deployment. Therefore, particular care should be taken for WCET analysis techniques to scale for multicore systems.

A.4 Hardware Alternatives

Above mentioned limitations have motivated studies that analyze if changes in hardware can facilitate the effective timing analysis for multicore architectures. Several hardware proposals [13, 5, 16, 10, 9] aim at easing the computation of composable WCET bounds of tasks running on multithreaded architectures. However as COTS architecture are dominated by the consumer electronic market which does not actually share such predictability concerns, it is very unlikely that such costly mechanism to be integrated by general chip providers.

A.5 Modeling Alternatives

In order to provide timing guarantees for future mission-critical systems running on multicore COTS, new timing analysis mechanisms have to be developed. Recently a new set of approaches based on resource stressing benchmarks [15, 2] have been proposed, where the target application is tested against the resource stressing benchmarks to 1) identify which hardware resources are effectively shared, and to 2) identify the shared hardware resources each target application is sensitive to.

By combining this information for co-running benchmarks, we will be able to identify the benchmarks that will run smoothly together avoiding worst-case contentions. Such a scheme will allow us to better predict and control the variability of execution time on multicore COTS architecture by selecting interference friendly mappings of co-runners.

A.6 Additional Appendix Authors

Petar Radojković, Barcelona Supercomputing Center, Spain.

Jingyi Bin, Thales Research & Technology, France and Fundamental Electronic Institute, France.

A.7 References

[1] T. G. Baker. Lessons learned integrating COTS into systems. In *Proceedings of the First International Conference on COTS-Based Software Systems*, ICCBSS '02, pages 21–30, 2002.

[2] J. Bin, A. Grasset, D. G. Perez, S. Girbal, P. Bonnot, and A. Merigot. Controlling execution time variability using cots in for safety critical systems. *ACACES*, page 191, 2012.

[3] C. Ferdinand, F. Martin, C. Cullmann, M. Schlickling, I. Stein, S. Thesing, and R. Heckmann. Program analysis and compilation, theory and practice. chapter New developments in WCET analysis, pages 12–52. 2007.

[4] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Workload Characterization, 2001*, WWC '01, pages 3–14, 2001.

[5] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken. Compsoc: A template for composable and predictable multi-processor system on chips. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):2:1–2:24, Jan. 2009.

[6] R. Heckmann and C. Ferdinand. Verifying safety-critical timing and memory-usage properties of embedded software by abstract interpretation. In *Proceedings of the conference on Design, Automation and Test in Europe*, DATE'05, pages 618–619, 2005.

[7] R. Kirner and P. Puschner. Obstacles in worst-case execution time analysis. In *Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, ISORC '08, pages 333–339, 2008.

[8] R. Kirner, I. Wenzel, B. Rieder, and P. Puschner. Using measurements as a complement to static worst-case execution time analysis. In *Intelligent Systems at the Service of Mankind*, volume 2. Dec. 2005.

[9] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, page 91, 2003.

[10] B. Lickly, I. Liu, S. Kim, H. D. Patel, S. A. Edwards, and E. A. Lee. Predictable programming on a precision timed architecture. In *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, CASES '08, pages 137–146, 2008.

[11] E. Mezzetti and T. Vardanega. On the industrial fitness of wcet analysis. In *Proceedings of the 11th International Workshop on Worst Case Execution Time Analysis (WCET2011)*. 2011.

[12] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. *European Dependable Computing Conference*, pages 132–143, 2012.

[13] R. Obermaisser, H. Kopetz, and C. Paukovits. A cross-domain multi-processor system-on-a-chip for embedded real-time systems. *IEEE Trans. Industrial Informatics*, 6(4):548–567, 2010.

[14] P. Puschner and A. Burns. Guest editorial: A review of worst-case execution-time analysis. *Real-Time Systems*, 18(2/3):115–128, 2000.

[15] P. Radojkovic, S. Girbal, A. Grasset, E. Quiñones, S. Yehia, and F. J. Cazorla. On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments. *TACO*, 8(4):34, 2012.

[16] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quinones, M. Gerdes, M. Paolieri, J. Wolf, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro*, 30(5):66–75, 2010.

[17] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, T. Mitra, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, I. Puaut, R. Heckmann, F. Mueller, P. Puschner, J. Staschulat, and P. Stenström. The worst case execution time problem, overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, pages 36–53, May 2008.

B. PROBABILISTIC TIME-ANALYSABLE COMPUTING SYSTEMS

Aggressive hardware acceleration features like deep memory hierarchies and many-cores; and software like complex motor control algorithms that require sophisticated mathematics [2], are needed to respond to the increasing demand for more software functionality due to the ever-rising proportion of system value that is delivered in software.

More powerful hardware/software also offers the opportunity to schedule a larger number of applications, while the use of many-cores allows co-hosting several critical and non-critical applications on a common powerful platform, providing a better performance/Watt ratio than a single core solution with similar performance. Such a reduction in the number of processors will also offer tremendous scope for cost reduction.

B.1 Existing Timing Analysis Techniques

The benefits that the additional computational power of these advanced hardware/software features can potentially provide cannot be realized in the mission critical market unless accompanied by analysis techniques that enables their use. In particular, there must be guarantees on the achievable performance. However, the introduction of complex hardware and software (1) challenges current practice in the timing analysis of real-time embedded systems; and (2) produces abrupt performance variations due to rather small variations in the program or the execution environment.

Timing analysis is a complex process, mainly due to the fact that the variations in execution time of programs is caused by the characteristics of the software itself, as well as by the hardware platform upon which the program is to run and the complex interaction between both. As a result, all characteristics of software and hardware must be thoroughly understood in order to provide meaningful time guarantees.

Static timing analysis (STA) techniques rely on the construction of a cycle-accurate model of the system and a mathematical representation of the application code which makes it possible to determine the timing behavior on that model. STA approaches have the important limitation that they are expensive to carry out requiring exhaustive knowledge of all factors, both hardware and software, that determine the execution history of the program under analysis. For instance, to determine whether an access to a cache hits/misses in cache, the addresses of previous accesses have to be known. The introduction of multicore architectures and the software running on top of them will dramatically increase this cost. It is also the case that multicore architectures may be subject to intellectual property restrictions or have huge documentation, likely without the level of detail required to make a cycle-accurate model, making it altogether impossible to use STA.

Alternatively, measurement-based timing analysis (MBTA) techniques have been devised to sort out the limitations of STA. MBTA relies on extensive testing performed on the real system under analysis using stressful, high-coverage input data, recording the the longest observed execution time; and adding to it an engineering margin to make safety allowances for the unknown. However, the engineering margin is extremely difficult – if at all possible – to determine, especially when the system may exhibit discontinuous changes in timing due to inter-task interferences in shared resources in many-core processors or pathological cache access patterns. Moreover, uncontrolled interactions of applications with mixed criticality levels further challenge the WCET estimates obtained with MBTA.

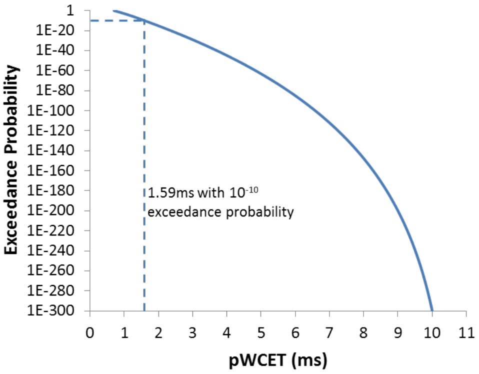

Figure 9: Synthetic example of a pWCET function

B.2 Probabilistic Timing Analysis (PTA)

New approaches have recently emerged to provide an alternative to those based on partitioning the resources. One of such approaches is the probabilistic approach [4][5][3][1]. PTA techniques enable probabilistic guarantees of timing correctness to be derived. For example, if the requirements placed on the reliability of a sub-system indicate that the probability of a timing failure must be less than 10^{-9} per hour of operation, then the PTA techniques aim to translate this reliability requirement into a probabilistic WCET (pWCET) for the sub-system. PTA effectively provides a continuum of WCETs for different confidence levels. Thus a sub-system may have a probability of 10^{-8} per hour of exceeding an execution time of 1.43ms, and probabilities of 10^{-9}, and 10^{-10} per hour of exceeding 1.51ms and 1.59ms, respectively (see Figure 9). The absolute WCET may be 10ms and can occur with a probability of 10^{-300} per hour.

The main idea of PTA techniques is that for future real-time systems, such probabilistic guarantees offer significant advantages over deterministic approaches which attempt to make absolute guarantees, severely limiting the use of advanced hardware features and inevitably offering significantly lower performance guarantees. PTA allows filling the gap between the execution times observed and the WCET attaching trustworthy probabilities to the execution times in between, thus enabling the (safe) use of lower execution time bounds. For instance, in the example before the effective utilization of the system is increased by a factor above 5x if the pWCET used for scheduling is as reliable as the hardware on which the program runs. Overall, using the absolute WCET that can occur with a probability of 10^{-300} per hour makes no sense if our system (e.g., an aircraft) is not resilient against higher probability catastrophic events such as an asteroid impact, which can occur with a probability higher than 10^{-30} per hour.

Under the probabilistic approach to time analysis, the timing behavior of certain hardware resources must be randomized, such as the cache placement and replacement [6]. Randomizing the timing behavior of hardware enables the use of PTA to analyze the timing behavior of the system. Randomization breaks many of the dependences between hardware and software components, simplifying the analysis. Techniques like *Extreme Value Theory* can be used to predict the timing behavior of applications [3].

PTA does not require randomizing the timing of all resources, but the timing behavior of resources can be characterized with a random variable, which is expressed as an *execution time profile* (ETP). In other words, each potential latency that a resource may take must have an associated probability. Those resources with constant latency, l, have a probability of 1 to have that latency and hence are also amenable to PTA. Consequently, PTA needs that any hardware resource has either (i) constant response time or (ii) different response times with an associated (true) probability each.

In order to treat a resource with PTA, the probability assigned to each latency in the timing vector of the ETP for that resource must be a *true* probability. The probability for a given latency is different from its frequency. This is best shown by an example. Consider a resource R_1 with $\overrightarrow{ETP}_1$= $\{\{t_1, t_2\}, \{0.5, 0.5\}\}$: each latency in the timing vector (t_1 and t_2) would have a true probability of occurrence of 0.5 if – in the implementation of that resource – on every request we flipped a coin and the request had latency t_1 if we saw heads and t_2 otherwise. In contrast, if for a deterministic stateful resource R_2 with the same potential latencies (t_1 and t_2) we *observed* that, for a given program, 50% of the requests take t_1 and 50% t_2, we would have a 50% observed frequency for each possible latency, but not a true 0.5 probability. This is so because for events that are strictly dependent on the history of execution, cumulative information on past events *cannot* be used to provide guarantees about the appearance of future events.

The example of the cache. Exploiting the execution history of the program, is one of the most common principles of processor design. A typifying example of this strategy is the cache. The use of caches is widespread in general-purpose processors because they dramatically improve average performance by exploiting locality (either temporal or spatial) in memory access patterns, reducing access times by several orders of magnitude. However, this strategy has an important downside: the execution time of programs, and their WCET, heavily depend on execution history, which challenges timing analysis. In particular, the combination of deterministic cache placement and replacement polices such as modulo placement and Least Recently Used (LRU) replacement make cache behavior to depend on the particular location of data and instructions in memory. A minor modification in such location may produce abrupt effects in performance by varying completely which cache lines compete for the space in each cache set.

Cache behavior imposes severe constraints on STA since all addresses must be known to provide tight WCET bounds. STA keeps track of the cache state on each access. However, dynamic memory, stack placement and operating system interaction among others make this process costly – if at all feasible. Any lack of information implies that accesses must be assumed to be missed and, even worse, assume that unknown accesses can occur in any cache set, thus evicting many cache lines from the guaranteed cache state tracked by STA. Alternatively, MBTA can run experiments to record the highest execution times. Unfortunately, there is no guarantee on whether those memory layouts leading to the WCET are included in the tests. This is a serious issue given that performance variation can be huge and those undesirable memory layouts may occur systematically once the system is deployed.

Conversely, PTA requires cache designs with random placement and replacement so that the timing behavior of caches does not depend on data and instructions location in memory. This has been proven to be feasible recently by means of either (i) hardware means such as parametric random placement [6] or (ii) software means such as stack and function placement randomization [7]. Those techniques make placement of objects in cache random so that each placement has a true probability of occurrence and hence, if measurement-based PTA (MBPTA) is used [3], observed execution times are relevant of those that will be observed in the system once deployed. Further, random placement and replacement have been proven to avoid abrupt performance variations, which facilitates providing QoS guarantees.

B.3 PTA benefits

The benefits of PTA can be summarized as follows: System reliability is expressed in terms of probabilities for hardware failures, memory failures, software faults and for the system as a whole. PTA techniques fit and extend this probabilistic approach to timing correctness so that probabilistic guarantees can be provided holistically for the different system components. In that sense, PTA is in match with the aging and reliability behavior of the hardware itself, which common wisdom accepts may fail with a given (though very low) probability. The first steps towards adapting PTA to work in the presence of faults at hardware level have been presented in [8].

PTA allows obtaining trustworthy and tight WCET estimates at low cost, which is attractive for the mission critical market, especially for the mainstream one, where cost constraints for QoS features are more severe.

Based on the fact that PTA provides probabilistic WCET estimates for any target exceedance probability, it can be used to obtain WCET estimates for applications with different criticality levels at low cost. In other words, a single (low cost) approach can be used to provide the timing guarantees needed across mixed criticalities.

Overall, we regard probabilistic timing analysis as a technically viable and economically attractive solution to respond to the demand for providing guaranteed timing analysis in future mission-critical/mainstream systems in the face of more performance aggressive software and hardware.

B.4 References

[1] F. Cazorla et al. PROARTIS: Probabilistically analysable real-time systems. Technical Report to appear in ACM TECS, 2013.

[2] M. Copeland. Implementing Complex Motor Control Algorithms with a Standard ARM® Processor. 2012.

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

[4] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *RTSS*, 2001.

[5] D. Griffin and A. Burns. Realism in Statistical Analysis of Worst Case Execution Times. In *WCET Workshop*, 2010.

[6] L. Kosmidis, J. Abella, E. Quinones, and F. Cazorla. A cache design for probabilistically analysable real-time systems. In *DATE*, 2013.

[7] L. Kosmidis, C. Curtsinger, E. Quinones, J. Abella, E. Berger, and F. Cazorla. Probabilistic timing analysis on conventional cache designs. In *DATE*, 2013.

[8] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quinones, and F. J. Cazorla. Degraded test mode for fault-aware probabilistic timing analysis. In *ECRTS*, 2013.