The Challenge of Time-Predictability in Modern Many-Core Architectures*

Vincent Nélis[1], Patrick Meumeu Yomsi[1], Luís Miguel Pinho[1], José Carlos Fonseca[1], Marko Bertogna[2], Eduardo Quiñones[3], Roberto Vargas[3], and Andrea Marongiu[4]

1 **CISTER/INESC-TEC Research Center, Porto, Portugal**
`{nelis, pamyo, lmp, jcnfo}@isep.ipp.pt`
2 **University of Modena, Italy**
`marko.bertogna@unimore.it`
3 **Barcelona Supercomputing Center, Spain**
`{eduardo.quinones, rvargas}@bsc.es`
4 **IIS – ETH Zürich, Switzerland**
`a.marongiu@iis.ee.ethz.ch`

Abstract

The recent technological advancements and market trends are causing an interesting phenomenon towards the convergence of High-Performance Computing (HPC) and Embedded Computing (EC) domains. Many recent HPC applications require huge amounts of information to be processed within a bounded amount of time while EC systems are increasingly concerned with providing higher performance in real-time. The convergence of these two domains towards systems requiring both high performance and a predictable time-behavior challenges the capabilities of current hardware architectures. Fortunately, the advent of next-generation many-core embedded platforms has the chance of intercepting this converging need for predictability and high-performance, allowing HPC and EC applications to be executed on efficient and powerful heterogeneous architectures integrating general-purpose processors with many-core computing fabrics. However, addressing this mixed set of requirements is not without its own challenges and it is now of paramount importance to develop new techniques to exploit the massively parallel computation capabilities of many-core platforms in a predictable way.

1998 ACM Subject Classification C.3 Special-Purpose and Application-Based Systems

Keywords and phrases Time-Predictability, Many-Cores, Multi-Cores, Timing Analysis

Digital Object Identifier 10.4230/OASIcs.WCET.2014.63

1 Current Trends in Application Requirements

Nowadays, computing systems are subject to a wide continuum of requirements, spanning from high-performance computing (HPC) systems to real-time embedded computing (EC) systems. Sitting on one extremity of that spectrum, HPC systems have been for a long time the realm of a specific community within academia and specialized industries; in particular,

* This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project(s) FCOMP-01-0124-FEDER-037281 (CISTER), and by the European Union, under the Seventh Framework Programme (FP7/2007-2013), grant agreement n° 611016 (P-SOCRATES), and by EU project TACLe (ICT COST Action IC1202).

14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014).
Editor: Heiko Falk; pp. 63–72
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

those targeting demanding analytic- and simulation-oriented applications that require massive amounts of data to be processed. For HPC system designers, "the faster, the better" is the mantra. On the other side of the spectrum, EC systems have also focused on very specific systems; in particular those with pre-set and specialized functionalities for which timing requirements prevail over performance requirements. Historically, the key objective for designers of EC systems was to design highly predictable systems where the time taken by every computing operation is *upper-bounded and these upper-bounds are known at design time*; Being fast was secondary.

With the new generation of computing platforms and the ever-increasing demand for safer but more complex applications, the conceptual boundary that was pulling HPC and EC systems apart is getting thinner every day. HPC systems require more and more guarantees on the timing behavior of their applications while EC systems face an increasing demand for computational performance. As a result, these HPC and EC systems that used to be torn apart by orthogonal requirements are now converging towards a brand new category of systems that share both HPC and EC requirements. This is the case of real-time complex event processing (CEP) systems [6], a new area of computing systems that literally cross the boundaries between the HPC and the EC domains.

In these CEP systems, the data come from multiple event streams and is correlated in order to extract and provide meaningful information within a pre-defined time bound. In cyber-physical systems for instance, ranging from automotive and aircraft to smart grids and traffic management, CEP systems are embedded in a physical environment and their behavior obeys technical rules dictated by this environment. Another example is the banking/financial markets where CEP systems process large amounts of real-time stock information in order to detect time-dependent patterns, automatically triggering operations in a very specific and tight time-frame when some pre-defined patterns occur [12].

The underlying commonality of the systems described above is that they are time-critical (whether business-critical or mission-critical) and with high-performance requirements. In other words, for such systems, the correctness of the result is dependent on both performance and timing requirements, and the failure to meet either of them is critical to the functioning of the system. In this context, it is essential to guarantee the timing predictability of the performed computations, meaning that arguments and analysis are needed to be able to make arguments of correctness, e.g., performing the required computations within well-specified time bounds.

2 Trends in the High-performance and Embedded Computing Domains

Until now, trends in high-performance and embedded computing domains have been running in opposite directions. On the one hand, HPC systems are traditionally designed to make the common case as fast as possible, without concerning themselves for the timing behavior (in terms of execution time) of the not-so-often cases. The techniques developed for HPC are usually based on complex hardware and software structures that make any reliable time bound almost impossible to derive. On the other hand, real-time embedded systems are typically designed to provide energy-efficient and predictable solutions, without heavy performance requirements. Instead of *fast* response times, they aim at having *predictable* response times, in order to guarantee that deadlines are met in all possible execution scenarios. Hence these systems are typically based on simple hardware architectures, using fixed-function hardware accelerators that are strongly coupled with the application domain.

This section presents the evolution of both the HPC and the EC computing domains from a hardware and software point of view.

2.1 Hardware Trends

Owing to the immense computational capabilities needed to satisfy the performance requirements of HPC systems and because the resulting exponential increments of power requirements exceeded the technological limits of classic single-core architectures (typically referred to as the power-wall), multi-core processors have entered both computing markets in the last years [11]. The leading hardware manufacturers are now offering an increasing number of computing platforms that integrate multiple cores within a chip, which contributes to an unprecedented phenomenon sometimes referred to as the multi-core revolution.

Multi-core processors are much more energy-efficient and have a better performance-per-cost ratio than their single-core counterpart as they improve the application performance by exploiting thread-level parallelism (TLP). Applications are split into multiple tasks that run in parallel on different cores, which has for consequence to spread into the multi-core world an important challenge that was already faced by HPC designers at multi-processor system level: the parallelization. In the HPC domain, many-core platforms are seen as a highly scalable multi-core architecture that overcomes the limits of traditional multi-cores (such as the contention for memory bus for example) and considerably increases the degree of parallelization of the tasks that can be exploited.

In the EC domain, the necessity of developing more flexible and powerful systems have pushed the embedded market in the same direction. For instance, the mobile phone market evolved from selling cellphones with a limited number of well-defined functions to selling smart-phones and tablets with an unlimited access to a virtual store full of user-made applications. As newest applications are more and more greedy in term of performance, multi-core architecture have been increasingly considered as the solution to cope with the performance and cost requirements [3], because they allow multiple application services to be scheduled on the same processor, which maximizes the hardware utilization while reducing its cost, size, weight and power requirements. Unfortunately, most of multi-core architectures have been designed to provide increased performance rather than towards offering time-predictability to the application system and broadly speaking, these platforms failed to provide an appropriate execution environment to time-critical embedded applications. This is why those applications are still executed on simple architectures that are able to guarantee a predictable execution pattern while avoiding timing anomalies [7], which makes real-time embedded platforms still relying on either single-core or simple multi-core CPUs, integrated with fixed-function hardware accelerators into the same chip: the so-called System-on-Chip (SoC).

The needs for time-predictability, energy-efficiency, and flexibility, coming along with Moore's law greedy demand for performance and the advancements in the semiconductor technology, have progressively paved the way for the introduction of many-core systems in both the HPC and EC domains. Examples of many-core architectures include the Tilera Tile CPUs [13] (shipping versions feature 64 cores) in the embedded domain and the Intel MIC [4] and Intel Xeon Phi [5] (featuring 60 cores) in the HPC domain. The introduction of many-core systems has set up an interesting trend wherein both the HPC and the real-time embedded domains converge towards similar objectives and requirements. Figure 1 shows the trend towards the integration of both domains. In this current trend, challenges that were previously specific to each computing domain start to be common to both (including energy-efficiency, parallelisation, compilation, software programming) and are magnified by

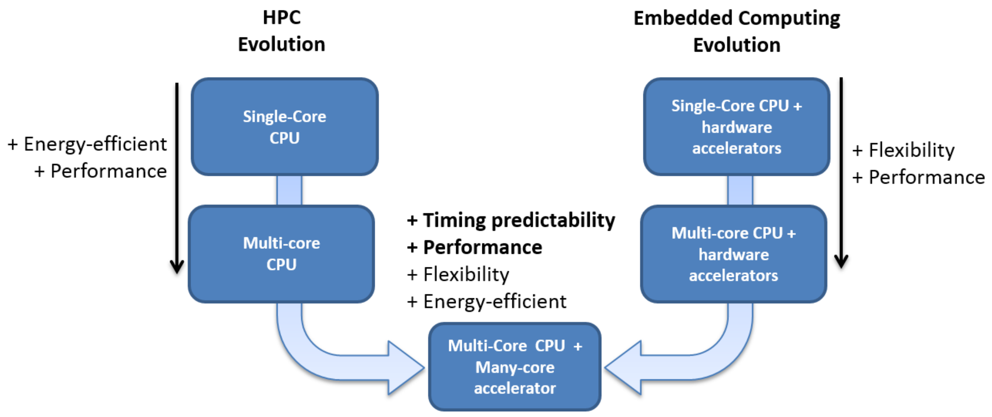

Figure 1 Trend towards the integration of HPC and embedded computing platforms.

the ubiquity of many-cores and heterogeneity across the whole computing spectrum. In that context, cross-fertilization of expertise from both computing domains is mandatory. In our opinion, there is still one fundamental requirement that has not yet been considered: time predictability as a mean to address the time criticality challenge when computation is parallelised to increase the performance. Although some research in the embedded computing domain has started investigating the use of parallel execution models (by using customized hardware designs and manually tuning applications by using specialized software parallel patterns [10]), a real cross-fertilization of expertise between HPC and embedded computing domains is still missing.

2.2 Software Trends

Industries with both high-performance and real-time requirements are eager to benefit from the immense computing capabilities offered by these new many-core embedded designs. However, these industries are also highly unprepared for shifting their earlier system designs to cope with this new technology, mainly because such a shift requires adapting the applications, operating systems, and programming models in order to exploit the capabilities of many-core embedded computing systems. Neither many-core embedded processors have been designed to be used in the HPC domain, nor HPC techniques have been designed to apply embedded technology. Furthermore, real-time methods that determine the timing behavior of an embedded system are not prepared to be directly applied to the HPC domain and many-core platforms, leading to a number of significant challenges. Although customized processor designs could better fit real-time requirements [10], the design of specialized processors for each real-time system domain is not a desired option for obvious financial reasons.

Different parallel programming models and multiprocessor operating systems have been proposed and are increasingly being adopted in today's HPC computing systems. In recent years, the emergence of accelerated heterogeneous architectures such as GPGPUs, have introduced parallel programming models such as OpenCL [9], the currently dominant open standard for parallel programming of heterogeneous systems, or CUDA [8], the dominant proprietary framework of NVIDIA. Unfortunately, they are not easily applicable to systems with real-time requirements since, by nature, many-core architectures are designed to integrate as many functionalities as possible into a single chip and thus they inherently share as many resources as possible amongst the cores, which heavily impacts the ability to provide timing guarantees.

The embedded computing domain world has always seen many application-specific accelerators with custom architectures on which applications are manually tuned to achieve predictable performance. Such types of solutions have a limited flexibility which complicates the development of embedded systems. However, we firmly believe that commercial-off-the-shelf (COTS) components based on many-core architectures are likely to dominate the embedded computing market in the near future. Assuming that embedded systems will evolve in this way, migrating real-time applications to many-core execution models with predictable performance requires a complete redesign of current software architectures. Real-time embedded application developers will therefore either need to adapt their programming practices and operating systems to future many-core components, or they will need to content themselves with stagnating execution speeds and reduced functionalities, relegated to niche markets using obsolete hardware components. This new trend in the manufacturing technology, alongside the industrial need for enhanced computing capabilities and flexible heterogeneous programming solutions of accelerators for predictable parallel computations, bring to the forefront important challenges for which solutions are urgently needed. To that end, we envision the necessity to bring together next-generation many-core accelerators from the embedded computing domain with the programmability of many-core accelerators from the HPC computing domain, supporting this with real-time methodologies to provide time-predictability. Time-predictability is an essential feature to allow system designers to model the timing behavior of the system through timing analysis techniques and then, based on these models, check that all its timing requirements are fulfilled.

3 Background on timing analysis techniques

What is it?

Timing analysis is any structured method or tool applied to the problem of obtaining information about the execution time of a program, a part of a program, or even any kind of computer operation such as a fetching a data in the cache or sending a packet over a network. The fundamental problem that timing analysis techniques have to deal with is the fact that the execution time of an operation is not a fixed constant, but rather varies across a range of possible execution times. Variations in the execution time of an operation occur due to variations in input data, as well as the characteristics and execution history of the software, the processor architecture, and the computer system in which the operation is executed.

What is it needed for?

Timing analysis is needed to assess that all the timing requirements of the system are fulfilled. In the EC domain, most of systems with real-time requirements require a reliable timing analysis to be efficiently designed and verified, in particular when the system is used to control safety critical components in application areas such as vehicles, aircraft, medical equipment, and industrial plants. In these application domains, in order for the whole system to be validated and assessed as safe, it is commonplace that only a subset of tasks has to fulfill strict timing requirements (i.e., they are required to complete their operations within specified time limits). That is, only few components of the entire system are "critical" and in need of precise timing analysis. An accurate timing analysis is consequently not always required as many components may be subject to real-time requirements but are in essence not critical. It is currently the case, for example, for most of modern applications that share HPC and real-time requirements.

Although the high criticality of some applications is beyond doubt, for many functions it is rather a business matter to evaluate whether the costs and consequences of a timing-related failure is worth the cost of the various mechanisms that must be implemented to prevent/handle this failure. In industrial systems, there is a continuum of criticality levels in the set of components of a real-time system. Depending on the criticality of each component an approximate or less accurate analysis might be acceptable. Real-time applications are commonly categorized as safety-critical (or life-critical), mission-critical, and non-critical. A failure or malfunction of a safety-critical application may result in death or serious injury to people, loss or severe damage to equipment or environmental harm, whereas a failure of a mission-critical application may result in a failure of the entire system, but without damaging it nor its embedding environment, and a failure of a non-critical application has no severe consequences. While safety- and mission-critical components must be certified with a very high level of confidence (through extremely accurate and thorough analyses), components that are less or not critical at all need only to maintain a "decent" average throughput and should be proven not to affect the execution behavior of the critical components.

How does timing analysis work?

The worst-case execution time of an operation depends not only on the intrinsic nature of the operation and its inner sub-operations, but also on external events that may occupy or lock a resource that the operation needs to access. For example, the worst-case traversal time of a packet throughout a network-on-chip does not only depend on intrinsic properties like the size of the data sent, the routing algorithm employed, or the capacity of the links between the source and the destination, but also on the current traffic on the network at the time the packet is sent. This means that, in order to provide a safe upper-bound on the execution time of an operation, timing analysis techniques must consider not only the nature of the operation and the characteristics of the executing environment, but also they must identify the worst "context" in which the operation can be performed. Owing to this influence of the context of execution, the body of knowledge developed in academia further sub-categorizes the timing analysis objectives and distinguishes between (i) the worst-case execution time (WCET) analysis and (ii) the interference analysis.

What is WCET analysis?

The *WCET analysis* is the context-independent part of the timing analysis that focuses on deriving a safe upper-bound on the execution time of a program (or any piece of code). It assumes that the analyzed program runs in isolation and without interruption, i.e., there is no other user-tasks running concurrently with the analyzed task, interrupts from the operating system are disabled, and the analyzed task gets immediate access to a resource as soon as it needs to. Under these circumstances, the WCET of a program is defined as the longest execution time that will ever be observed when the program is run on the target hardware. It is the most critical measure for most real-time work. For example, as mentioned earlier, the WCET of tasks is a key component for the higher-level schedulability analysis, but in practice it is also used at a lower level analysis, e.g., to ensure that software interrupts will have sufficiently short reaction times, or to guarantee that operating system calls return to the user application within pre-defined time-bounds.

WCET analysis can be performed in a number of ways using different tools, but the main methodologies employed can be broadly classified in three categories: (1) static analysis techniques, (2) measurement-based analysis techniques, and (3) hybrid techniques. Broadly speaking, measurement-based techniques are suitable for software that are less time-critical and for which the average-case behavior (or a rough WCET estimate) is more meaningful or relevant than an accurate estimate. For example, systems where the worst-case scenario is extremely unlikely to occur and/or the system can afford to ignore it if it does occur. For highly time-critical software, where every possible execution scenario must be covered and handled, the WCET estimate must be as accurate as possible and static analysis or some type of hybrid method is therefore preferable.

What is interference analysis?

The *interference analysis* is the execution-context aware part of the timing analysis. It focuses on deriving safe upper-bounds on the extra execution time-penalty that the analyzed task may suffer during its run-time because of the interference with other tasks and with the system. It takes into account the context in which each operation is performed and identifies the worst-case interference scenario for the analysis. Typically, interference analysis will supplement the outputs of the WCET analysis by factoring in extra delays due to, for example, sporadic SW/HW interrupts or the interference from other tasks on the shared communication bus, network, or caches. Specifically, for every shared software and hardware components (such as the caches, the main memory, the shared data, etc.) and for each access to these resources that the analyzed task may request, the interference analysis techniques identify the worst initial "state" of the component and the worst-case scenario of interference (from the system and the other tasks) on that component that would induce the largest execution time for the analyzed access. These upper-bounds are then used to adjust the WCET estimates obtained from the WCET analysis techniques. It must be noted that both the WCET analysis techniques and the interference analysis techniques may analyze the same SW/HW resources, but their main focus and objectives are not the same. For example, some WCET analysis tools include cache analysis, during which the tool may substantially tighten the WCET estimate by taking into account that the requested data will not always have to be fetched from the main memory as it may have been loaded already and is thus available in the local cache. In contrast, interference analysis techniques also analyze the cache(s) but relax the assumption that the analyzed task is the only one that can use it, thus allowing tasks to evict cache lines from each other. Relaxing this assumption trivially causes the tasks to experience extra delays during their execution and the WCET estimates must therefore be augmented accordingly.

Interference between tasks and applications are typically reduced by ensuring a certain degree of "isolation" between those tasks and applications. Isolation can come in different flavors: tasks can be isolated in the time domain, the space domain, or both, and it can be symmetric or asymmetric. Isolation between system components also provides other advantages: it is fostered by system designers to avoid fault propagation for example, and when system timeliness is of concern, it helps provide two major features:

- **Time compositionality:** the timing properties of interest at system level can be determined from the timing properties of its constituent components.
- **Time Composability:** the timing properties determined for individual components in isolation should hold after the composition with other components.

Typically, these two properties (and in particular the time-composability property) are obtained by enforcing spatial and temporal isolation between software components at run-time.

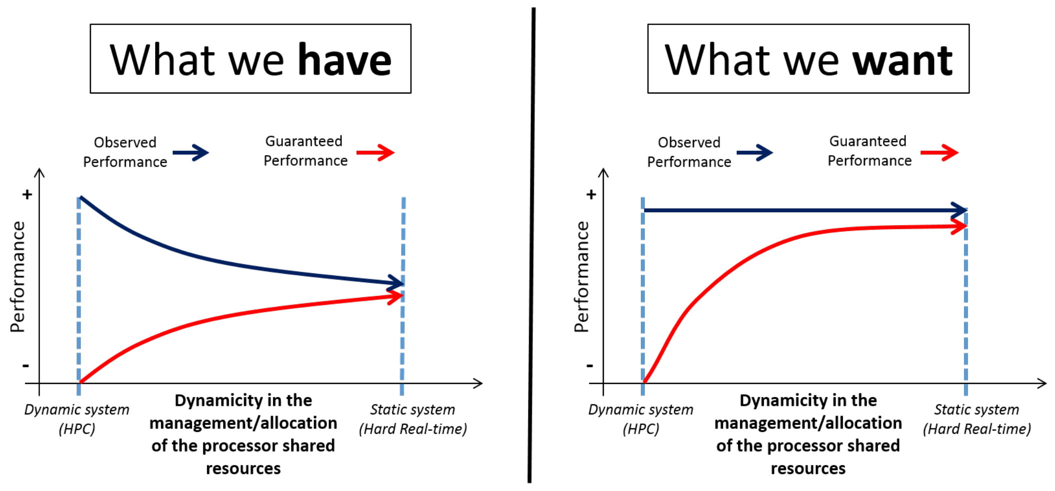

Figure 2 Performance degradation and guarantees improvement.

4 Glance at a few forthcoming challenges in ensuring time-predictability

The challenge of ensuring time-predictability for this new generation of systems with mixed requirements is twofold. On one side, the software solutions used in HPC systems must be adapted to be more predictable while preserving (as much as possible) their efficiency and on the other side, timing analysis techniques used to validate EC systems must be adapted to these new software solutions.

What does it imply to adapt the HPC software solutions?

It mostly implies reducing the dynamicity of all the mechanisms that are responsible for the management of the communication, memory, and computing resources. Instead of taking decisions on-the-fly based on the execution history and/or the current state of the system (as it is done in HPC systems), most of the decisions regarding the allocation of the resources among the tasks should ideally be taken before the run-time to enable a thorough offline analysis of the system timing behavior. Figure 2 (left side) illustrates the expected trends in the (observed) average performance and in the guaranteed performance when the dynamicity of the resource allocation schemes is reduced. Typically, as we shift the decision-taking process from the run-time to the design-time we limit the dynamicity of the system, which has for effect to decrease the observed performance as the system becomes less "flexible" while the guaranteed performance increases as the system becomes more predictable. The challenge here is to obtain high performance and tight guarantees of this high performance as depicted on the right-hand side of Figure 2 or, if this turns out not to be possible, one should at least find an appropriate trade-off between the observed performance and the guaranteed performance.

What are the changes needed at the programming model level?

In a nutshell, programming models need to be extended to provide detailed information about the code including for instance information on the control flow, timing properties, and functional and data dependencies between parts of the code. These annotations of the code could be used to extract an accurate and complete model of the application where all the dependencies between the functions (or any piece of code) are clearly documented. Together with the control flow information and the estimations of the worst-case execution time of each part of the code, this information could be used by timing analysis techniques to derive safe bounds (exact or probabilistic) on the overall execution time of the application.

To the best of our knowledge, the greatest effort in that direction is provided at Barcelona SuperComputing Center (BSC) where researchers have developed OmpSs, a programming model that integrates features from the StarSs programming model developed also at BSC into a single programming model. In particular, the objective of OmpSs is to extend OpenMP with new directives to support asynchronous parallelism and heterogeneity (devices like GPUs).[1] However, it can also be understood as new directives extending other accelerator based APIs like CUDA or OpenCL. The OmpSs environment is built on top of the Mercurium compiler and the Nanos++ run-time environment. More details about OmpSs and its objective can be found in [2].

Once we have a time-predictable setup, can we apply commercially-available WCET analysis tools?

It is very unlikely that all the existing methods will be applicable to the next-gen applications that share HPC and real-time requirements, especially it is the case for static approaches. Although static approaches have proven to be very efficient for safety-critical embedded systems these next-gen applications are not (yet?) safety-critical even though they present real-time requirements, which means that they are not subject to the hard and fast programming rules that are idiosyncratic to the safety-critical domain. They typically use pointers, dynamic memory allocation, recursive functions, variable-length loops, etc., and sometimes these applications are implemented by third party companies that are not concerned at all by the validation of the overall system, i.e. the code is not annotated with timing-related information that could be helpful for the timing analysis like loop-bounds for instance. Because of the lack of strict programming rules and the lack of information related to timing aspects of the code, static approaches are likely to fail to provide tight upper-bounds on the execution time and we foresee a rising popularity of measurement-based and probabilistic approaches in a near future.

Furthermore, it must be noted that it is unreasonable (if not impossible) to perform an exhaustive testing of these next-gen applications. Besides the fact that the size and the complexity of the software are constantly increasing, forthcoming platforms may chose to continue to increase their performance by borrowing more and more techniques from the HPC domain, including advanced computer architecture features such as caches, pipelines, branch prediction, and out-of-order execution. These features increase the speed of execution on average, but also make the timing behavior much harder to predict by parsing the code, since the variation in execution time between fortuitous and worst cases increases. The problem was already central in single-core platforms but is now further exacerbated in a multi/many-core setting where low-level hardware resources like caches and communication medium are shared by several cores, thereby inducing situations in which several entities contend for accessing the same resource.

[1] Note that this objective has been achieved by now and the latest version of openMP already integrates the research results obtained at BSC in that domain.

And what about interference analysis techniques?

As introduced before, the existing WCET techniques cannot be applied as is and need to be augmented by further analyses to factor in all the extra delays due to contention for the shared resources. Although preliminary results have already been presented in that direction (see [1] for a list of potential sources of interference between tasks), most of the scientific papers on the subject present techniques to estimate the extra delay due the contention for a single shared resource. That is, the authors focus on one and only one source of interference at a time, such as the cache, the network, the memory bus, etc., while the challenge of making these analyses work together has almost never been studied and we firmly believe that interference analysis need to be tackled in a holistic, integrated perspective.

References

1 Dakshina Dasari, Bjorn Andersson, Vincent Nelis, Stefan M. Petters, Arvind Easwaran, and Jinkyu Lee. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In *Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM'11, pages 1068–1075, Washington, DC, USA, 2011. IEEE Computer Society.

2 Alejandro Duran, Eduard Ayguadé, Rosa M. Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. Ompss: A proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21:173–193, 2011-03-01 2011.

3 T. Ungerer et al. MERASA: Multi-core execution of hard real-time applications supporting analysability. In *IEEE Micro, Special Issue on European Multicore Processing Projects*, volume 30:5, pages 66–75. IEEE Computer Society, aug 2010.

4 Intel Corporation. *Intel Many Integrated Core (MIC) Architecture*, last access Nov 2013. `http://www.intel.com/content/www/us/en/architecture-and-technology/many-integratedcore/intel-many-integrated-core-architecture.html`.

5 Intel Corporation. *Intel Xeon Phi*, last access Nov 2013. `http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html`.

6 David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Longman Publishing Co., Inc., 2001.

7 T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In *Proc. of the 20th IEEE Real-Time Systems Symposium*, pages 12–21, 1999.

8 NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture, Version 2.0*, 2008.

9 OpenCL. *The open standard for parallel programming of heterogeneous systems*, 2013. `http://www.khronos.org/opencl/`.

10 parMERASA FP7 European Project – grant agreement 287519. *Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analysability*, 2011–2014. `http://www.parmerasa.eu`.

11 Sutter, Herb. *Welcome to the Jungle.* `http://herbsutter.com/welcome-to-the-jungle/`.

12 R. Tieman. Algo trading: the dog that bit its master. *Financial Times*, March, 2008.

13 Tilera Corporation. *Tile Processor, User Architecture Manual, release 2.4, DOC.NO. UG101*, May 2011.