Time Criticality Challenge in the Presence of Parallelised Execution

Luis Miguel Pinho[1], Eduardo Quiñones[2], Marko Bertogna[3], Luca Benini[4], Jorge Pereira Carlos[5], Claudio Scordino[6], Michele Ramponi[7] *

[1]lmp@isep.ipp.pt, ISEP (Portugal)
[2]eduardo.quinones@bsc.es, BSC (Spain)
[3]marko.bertogna@unimore.it, University of Modena (Italy)
[4]luca.benini@unibo.it, ETH Zurich (Switzerland)
[5]jorge.pereirac@atosresearch.eu, Atos (Spain)
[6]claudio@evidence.eu.com, Evidence Srl (Italy)
[7]ramponi@activetechnologies.it, Active Technologies Srl (Italy)

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. On one side, new kinds of HPC applications are being required by markets needing huge amounts of information to be processed within a bounded amount of time. On the other side, EC systems are increasingly concerned with providing higher performance in real-time, challenging the performance capabilities of current architectures. The advent of next-generation many-core embedded platforms has the chance of intercepting this converging need for predictable 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. This convergence, however, raises the problem about how to guarantee timing requirements in presence of parallel execution. This paper presents a novel approach to address this challenge through the design of an integrated framework for the execution of workload-intensive applications with real-time requirements.

1 Motivation

High performance computing (HPC) has been for a long time the realm of a specific community within academia and specialised industries; in particular, those targeting demanding analytics and simulations applications that require processing of massive amounts of data. In a similar way, embedded computing (EC) has also focused mainly on specific systems with specialised and fixed functionalities and for which timing requirements were considered as much more important than performance requirements. However, with the ever-increasing availability of more powerful processing platforms, alongside affordable and scalable software solutions, both HPC and EC are extending to other sectors and application domains.

The demand for increased computational performance is currently widespread and is even more challenging when large amounts of data need to be processed, from multiple data sources, with *guaranteed processing response times*. As a result, in the last years a new type of applications has been challenging the performance capabilities of hardware platforms, by crossing the boundaries between the HPC and the embedded computing domains.

This is the case of real-time complex event processing (CEP) systems[21], a new area for HPC in which the data coming from multiple event streams is correlated in order to extract and provide meaningful information. Some interesting real-time CEP systems follow:

- In cyber-physical systems, ranging from automotive and aircrafts, to smart grids and traffic management, CEP systems are embedded in a physical environment and their behaviour obeys technical rules dictated by this environment.
- In the banking/financial markets, 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[34].

* This work has been financially supported by the European commission through the P-SOCRATES project (FP7-ICT-2013-10).

The underlying commonality of the real-time 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 those 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 bounds.

This paper presents a novel approach to address time-criticality and parallelisation challenges common to both high-performance and embedded computing domains, in which an integrated framework for executing workload-intensive applications with real-time requirements on top of next-generation commercial-off-the-shelf (COTS) platforms based on many-core accelerated architectures is envisioned. The framework, that will be developed within the P-SOCRATES FP7 project [1], addresses the problem from both the HPC and real-time computing domains, requiring investigation of new HPC techniques combined with new real-time methods to fulfil timing requirements. To do so, the main sources of indeterminism of HPC techniques will be identified, proposing efficient mapping and scheduling algorithms, along with the associated timing and schedulability analysis, to guarantee the real-time and performance requirements of the applications.

The paper is structured as follows. The next section introduces the trends which are common to the domains of HPC and EC. Afterwords, Section 3 presents the vision and challenges of the proposed real-time programming model, which builds upon high-performance programming models, augmented with dependencies and timing information. Section 4 then provides a brief summary of related work, whilst Section 5 summarizes the paper.

2 Trends in High-performance and Embedded Computing Domains

Until now, trends in high-performance and embedded computing domains have been running in opposite directions. On one side, HPC systems are traditionally designed to make the common case as fast as possible, without concerning themselves for the timing behaviour (in terms of execution time) of the not-so-often cases. As a result, the techniques developed for HPC are based on complex hardware and software structures that make any reliable timing bound almost impossible to derive. On the other side, 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 deterministically bounded response times, in order to guarantee that deadlines are met. For this reason, 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 computing domains from a hardware and software point of view.

2.1 Hardware Trends

In the last years, multi-core processors hit both computing markets [33]. The huge computational necessities to satisfy the performance requirements of HPC systems and the related exponential increments of power requirements (typically referred to as the *power-wall*) exceeded the technological limits of classic single-core architectures. For these reasons, the main hardware manufacturers are offering an increasing number of computing platforms integrating multiple cores within a chip, contributing to an unprecedented phenomenon sometimes referred to as *the multi-core revolution*. Multi-core processors provide a better energy efficiency and performance-per-cost ratio, while improving application performance by exploiting thread level parallelism (TLP). Applications are split into multiple tasks that run in parallel on different cores, extending to multi-core system level an important challenge already faced by HPC designers at multi-processor system level: *parallelisation*.

In the embedded systems domain, the necessity to develop more flexible and powerful systems (e.g., from fixed function phones to smart phones and tablets) have pushed the embedded market in the same direction. That is, multi-cores are increasingly considered as the solution to cope with performance and cost requirements [13], as they allow scheduling multiple application services on the same processor, hence

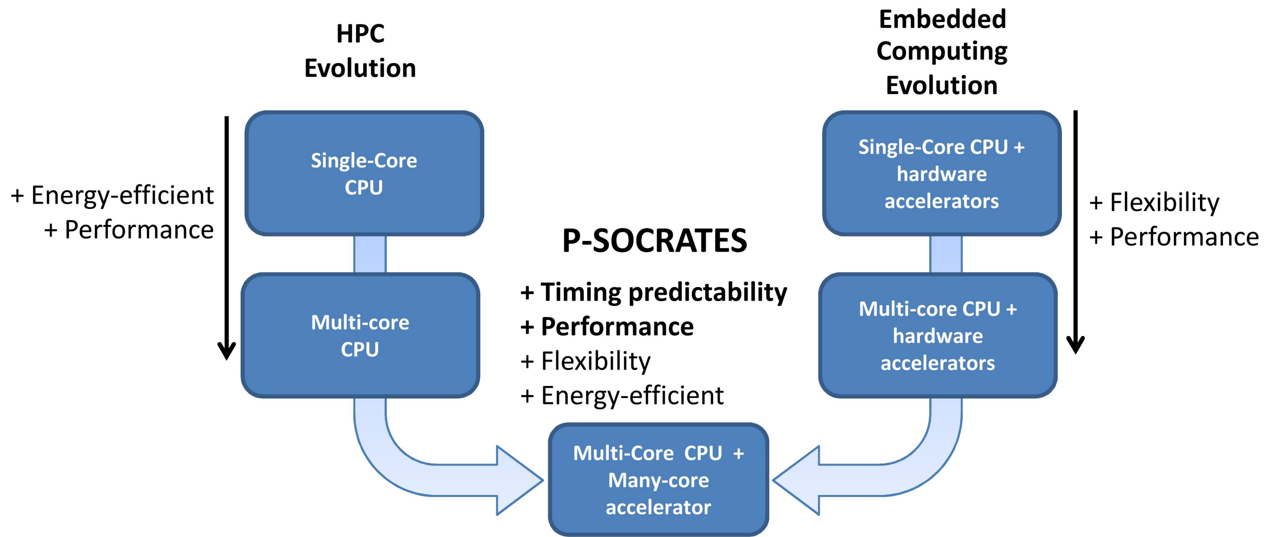

Fig. 1. Trend towards the integration of HPC and embedded computing platforms.

maximising the hardware utilisation while reducing cost, size, weight and power requirements. However, real-time embedded applications with time-critical requirements are still executed on simple architectures that are able to guarantee a predictable execution pattern while avoiding the appearance of timing anomalies [22]. This makes real-time embedded platforms still relying on either single-core or simple multi-core CPUs, integrated with fix-function hardware accelerators into the same chip: the so-called System-on-Chip (SoC).

The needs for energy-efficiency (in the HPC domain) and for flexibility (in the embedded computing domain), 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, i.e., multi-core chips containing a high number of cores (tens to hundreds) in both domains. Examples of many-core architectures include the Tilera Tile CPUs [35] (shipping versions feature 64 cores) in the embedded domain and the Intel MIC [16] and Intel Xeon Phi [17] (features 60 cores) in the HPC domain.

Thus, the introduction of many-core systems has set up an interesting trend wherein both the HPC and the real-time embedded domain converge towards similar objectives and requirements. Many-core computing fabrics are being integrated together with general-purpose multi-core processors to provide a heterogeneous architectural harness that eases the integration of previously hard-wired accelerators into more flexible software solutions. In recent years, the HPC computing domain has seen the emergence of accelerated heterogeneous architectures, most notably multi-core processors integrated with General Purpose Graphic Processing Units (GPGPU), because GPGPUs are a flexible and programmable accelerator for data parallel computations [32, 37]. Similarly, in the real-time embedded domain, STMicroelectronics P2012/STHORM [7] processor, which includes a dual-core ARM-A9 CPU coupled with a many-core processor (the STHORM fabric); and the Kalray MPPA (Multi-Purpose Processor Array) [19], which includes four quad-core CPUs coupled with a many-core processor. In both cases, the many-core fabric acts as a processing accelerator. 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 domains (including energy-efficiency, parallelisation, compilation, software programming) and are magnified by the ubiquity of many-cores and heterogeneity across the whole computing spectrum. In that context, cross-fertilisation of expertise from both computing domains is mandatory. We foresee that the next step towards the integration of high-performance and embedded computing domains will be the use of many-core embedded processors such as the STMicroelectronics P2012/STHORM. Such processors will provide the required performance level, while still being energy-efficient and time predictable. An example towards this integration is provided by Mont-Blanc and Mont-Blanc2 FP7 projects [24], which is developing a new hybrid supercomputer based on energy-efficient embedded ARM CPUs coupled with high-performance NVIDIA GPU many-core processors.

However, 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 customised hardware designs and manually tuning applications by using specialised software parallel patterns [29]), a real cross-fertilisation 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. On one hand, neither many-core embedded processors, such as the P2012/STHORM, have been designed to be used in the HPC domain, nor HPC techniques have been designed to apply embedded technology. On the other hand, real-time methods to determine the timing behaviour of an embedded system are not prepared to be directly applied to the HPC domain and P2012-like platforms, leading to a number of significant challenges. Although customised processor designs could better fit real-time requirements [29], the design of specialised processors for each real-time system domain is unaffordable.

On one side, 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 [28], the currently dominant open standard for parallel programming of heterogeneous systems, or CUDA [26], 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. Hence, they inherently share as many resources as possible amongst the cores, which heavily impacts the ability to providing timing guarantees.

On the other side, the embedded computing domain world has always seen plenty of application specific accelerators with custom architectures, manually tuning applications to achieve predictable performance. Such types of solutions have a limited flexibility, complicating the development of embedded systems. However, commercial off-the-shelf (COTS) components based on many-core architectures are likely to dominate the embedded computing market in the near future. As a result, 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 and 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.

3 Towards a Real-time Parallel Programming Model

The use of parallel programming models is fundamental to exploit the performance out of parallel architectures and provide good programmability (and so productivity) of high-performance systems. Among the different models, OpenMP [2] has become one of the most used parallel programming models due to its simplicity and scalability in shared memory systems such as current many-core processors. OpenMP defines *task* annotations to represent independent units of work that can run concurrently. Recently, OpenMP has been extended with new directives, *in*, *out* and *inout*, that allow introducing asynchronous parallelism by defining dependencies among task.

```
void compute (int *A, int *B, int N) {
    for(int i=0; i<N; i++) {
#pragma omp task in(A[i-1]) inout(A[i]) out(B[i])
        foo(&A[i-1],&A[i],&B[i]);

#pragma omp task in(B[i-1]) inout(B[i])
        bar(&B[i-1],&B[i]);
    }
}
```

Fig. 2. OpenMP 4.0 code sample showing the data-flow dependencies among tasks.

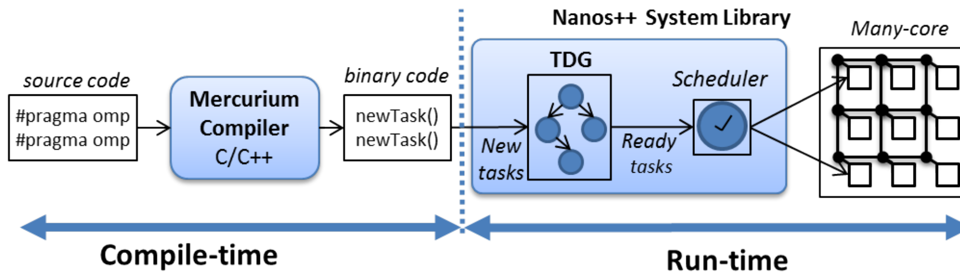

Fig. 3. OmpSs parallel programming framework.

Figure 2 shows a source code example using the dependency annotations. Each instance of task *foo* depends on data generated in previous loop iterations — i.e., *inout(A[i-1])*. Similarly, the task *bar* depends on *foo* outcome — i.e., *out(B[i])*.

This extension increases the freedom of task scheduling: tasks are scheduled for execution as soon as all depend tasks finished and there are available processor resources.

OmpSs [12] is a parallel programming framework compatible with OpenMP 4.0 whose effectiveness has been widely demonstrated in the HPC domain [10]. In OmpSs, the data-dependencies annotations are interpreted by a compiler, Mercurium, that emits calls to the runtime system Nanos++. Nanos++ is a parallel run-time system that dynamically generates the *task dependency graph* (TDG) at run-time. Each time a new task is created its *in* and *out* dependencies are matched against those of existing tasks. If a dependency, either read-after-write, write-after-write or write-after-read, is found, the task becomes a successor of the corresponding tasks. Tasks are scheduled for execution as soon as all their predecessor in the graph have finished and there are available processor resources. Figure 3 shows the complete system stack of OmpSs.

Current parallel frameworks base scheduling decisions on information available at run-time — i.e., the task dependency graph and processor resources availability (see Figure 3) — which makes it difficult to provide real-time guarantees. The reason is that the way tasks use shared processor resources determines the interferences that different tasks will suffer when accessing them, affecting the overall execution time of the application. A different usage of processor resources will result in a different execution.

In order to provide real-time guarantees without suffering any performance degradation, it is required to statically identify at design time which run-time configuration is needed, so the usage of shared processor resources is fixed and time guarantees can be provided. Therefore, it is of paramount importance to recover, at design time, relevant information to fix the usage of processor resources and so provide timing guarantees. Next sections provide our vision of how to address this challenge.

3.1 Vision

One of the main objectives when developing new time-criticality frameworks is to facilitate the transformation of current applications, designed to be executed on single-core platforms, to parallel multi-threaded

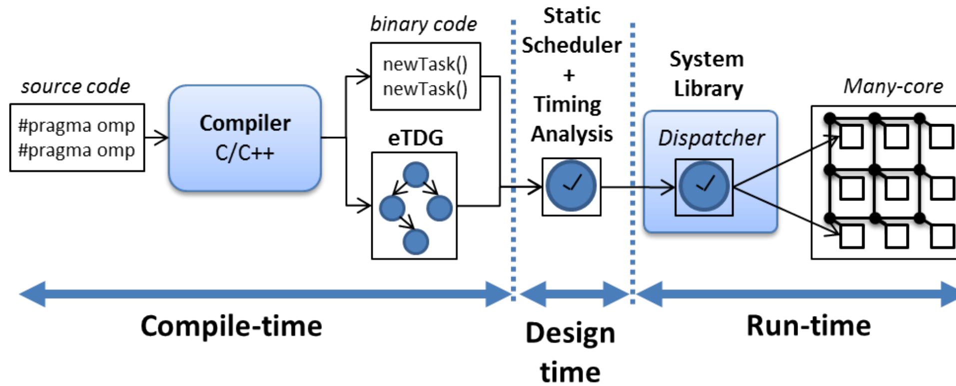

Fig. 4. Envisioned real-time parallel programming framework to provide timing guarantees.

applications, as well as to facilitate the design of new parallel applications. IT companies develop applications facing three important challenges:

- *Productivity.* Applications must be developed in a given time frame to accomplish time-to-market requirements.
- *Flexibility.* Applications must be easily adapted to multiple platforms.
- *Performance.* Applications must exploit all performance opportunities of the platform in which they run.

Therefore, it is fundamental to develop methodologies to easily extract parallelism from current and future applications without significantly changing the development processes, to let industry reuse its current test cases. One of the targets is to allow application providers to reuse their modelling techniques, taking full advantage of the computing power unleashed by the newest many-core platforms. Such transformation must be translated into less development time and potentially faster, cheaper time-to-market.

Our proposal is to *extend parallelism annotations*, which are extracted by the compiler, to identify portions of the application (tasks) that can run in parallel as well as *relevant information to derive the impact on execution time due to sharing resources when tasks communicate*. To do so, new compiler techniques must be developed to generate an *extended task dependency graph* (eTDG), containing relevant information required by the mapping and scheduling tool and the timing analysis method to allocate tasks to the different processor resources, guaranteeing that the real-time constraints of the application are accomplished. In other words, in order to provide timing guarantees, there is a necessity to fix the usage of shared processor resources.

Figure 4 shows the envisioned real-time parallel programming framework in which relevant information for task scheduling and timing analysis is recovered at compile- and design-time to fix the usage of processor resources.

3.2 Research Challenges

The envisioned approach presents multiple research challenges at compile-time and at design-time. This section summarizes the most important ones.

At compile-time, if all information is recovered, one could potentially provide tight execution bounds. Unfortunately, not all information can be recovered at compile time as there is information only available at run-time. This is the case, for instance, of data dependencies based on pointers, variable values or loop boundaries. In Figure 2, if the number of iterations (N) is not known, we cannot determine how many task instances of *foo* and *bar* will be executed and so the eTDG cannot be generated. Similarly, in Figure 5, if i and j values are not known at compile-time, it is not possible to determine if a data-dependency among tasks *produce* and *consume* exists.

In case the data-dependency cannot be solved or loop boundaries are not known at compile-time, it is required to consider *conservative approaches* in order to guarantee the functional correctness of the program.

```
void compute (int *A, int i, int j) {
#pragma omp task inout(A[i])
    produce(&A[i]);

#pragma omp task inout(A[j])
    consume(&A[j]);
    }
}
```

Fig. 5. The value of i and j must be known to determine the dependency among tasks *producer* and *consumer*.

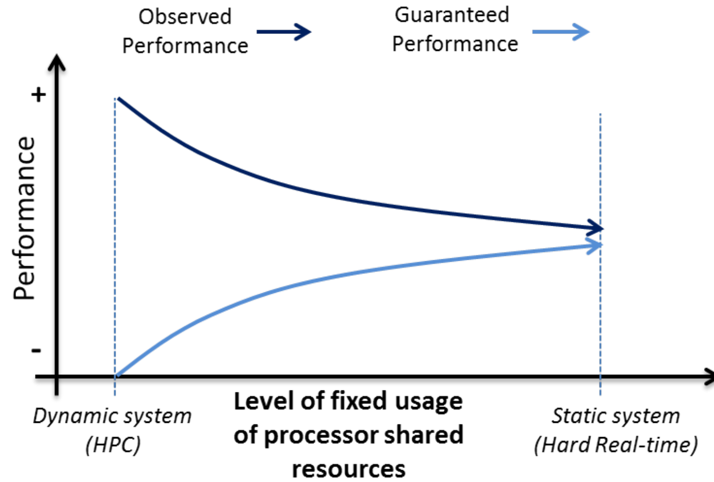

Fig. 6. Extracting information at compile-time increases guaranteed performance at the price of reducing the average performance due to conservative decisions.

Thus, if there is an unknown data-dependency, the construction of the eTDG must consider that this data-dependency exists. Similarly, if a loop boundary is unknown, it is required to determine an upper bound of the maximum number of loop iterations [3]. Needless to say that following a conservative approach will affect the average performance of the application. That is, false data-dependencies in the eTDG will force tasks to be executed sequentially. Similarly, assuming loop boundaries with higher number of iterations will make the eTDG to contain a higher number of task instances than the ones actually created, over-dimensioning the system due to tasks that are never executed.

Figure 6 shows the expected trends in the average and guaranteed performance when following conservative approaches. X-axis represents different levels of data recovered at compile-time, so the usage of processor resources can be fixed. As more information is extracted at compile-time, a more precise eTDG can be built and so higher guaranteed performance can be provided (light blue curve). However, due to the conservative approaches, the eTDG can differ from the TDG created at run-time, so that the average performance can be degraded (dark blue curve). Although counter-intuitive, it may happen that, in order to increase the guaranteed performance, a core is kept idle even when there is some pending workload to execute.

At *design-time*, it is necessary to provide the system with appropriate means to map the task dependency graphs to the underlying operating system threads (mapping), and dynamically schedule these threads to achieve both predictability and high-performance (scheduling). Although previous works [18, 14] have shown that run-time characterisation and management of locality has more potential than static locality analysis, dynamic information usage is a stopper to provide the timing guarantees for parallel applications on a many-core. Therefore, further research about how to allocate tasks to processor resources is needed, accounting for the impact of such allocation on other tasks due to interference when accessing shared resources. To that end, the programming model needs to be extended so that the responsibility for managing locality is shared among

the programmer and the mapping tool. This will allow providing timing guarantees to application customers while also providing maximum performance. Data annotations with in/out clauses provide a reasonable balance between the programmer and the system in managing locality [12], but further research is needed to minimise the interferences when accessing shared resources.

As a complement to the design of the scheduling algorithms, the real-time scheduling theory focuses on designing application, scheduler, and platform models and developing tools and techniques that together allow the timing behaviour of the entire system to be anticipated, modelled and analysed. Most of the analyses that can be found in the real-time scheduling literature assume that the system activities are functionally independent. An outgrowth of this assumption is to assume some of the key timing parameters to be constant, exact, and known at design time — e.g., the worst-case execution time of an activity is commonly assumed to be known at design time and invariant. However, when deployed on the same hardware architecture, activities that are co-scheduled on different cores share some low-level hardware resources, such as caches, communication buses and main memory. These shared resources inherently introduce functional dependencies between the activities as concurrent accesses to the same resource are not allowed; the timing behaviour of activities sharing the same resource is thus affected. Therefore, the existing techniques cannot be applied, but need to be augmented by further analyses to factor-in all the sources of contention due to shared resources. Preliminary results toward this direction have already been presented (e.g. [23, 11])

Moreover, these challenges need to be tackled in a holistic, integrated perspective. Our proposal is to construct the eTDG graph in synergy with the mapping and scheduling algorithms, with feedback from the timing and schedulability analysis. The strategy cannot be to search for all possible combinations in the whole design space. A guided process needs to be introduced, which is able to reason on the best mapping for a particular result.

4 Related Work

In what concerns programming models, the HPC world has seen a plethora of proposals for data or task parallelism (e.g. [4, 36, 15]). Furthermore, approaches such as [18] or [2, 12] also allow expressing dependencies among tasks, being the run-time system responsible of the dependencies to be satisfied before spawning dependent tasks. Task-based models can be dynamically managed by mapping the tasks to threads in a thread pool, e.g., using the popular Work-Stealing algorithm [8]. Yet, sources of non-determinism at run-time cause timing divergences among threads. Dynamic schedulers try to compensate by detecting them at run-time [9, 27] and either (i) "re-moulding" into more threads on-the-fly; (ii) boosting relative priorities, or (iii) adapting the mapping and number of allocated processing units. Since performance is the major goal, mapping strategies are mostly dynamic in nature, and, although being able to provide better average behaviour, they may allow for unpredictable unbounded delays.

General purpose computation on graphics processing units (GPGPUs) has also received a lot of attention, as it delivers high performance computing at rather low cost. GPUs are many-core computing fabrics that, integrated together with general-purpose processors, result in flexible and programmable accelerator for data parallel computations, programmed in frameworks such as OpenCL [28] or CUDA (Compute Unified Device Architecture) [26]. This approach also does not allow for supporting time-predictability and moreover is specific to SIMD operations.

In the real-time community, scheduling techniques have been the subject of extensive research. Traditional techniques have been extended for the multiprocessor case and more recently for parallel execution (e.g.[20, 31, 5, 25]). After the majority of the works considering 1-to-1 mappings, where each parallel execution is mapped to a thread, new models are appearing with more complex mapping approaches. Different mappings of parallel tasks to threads can be done, mostly statically [31, 5] but also dynamically [25], in order to increase system utilisation whilst maintaining predictability. These strategies need however to be extended for exploiting the dependency graphs from compiler generated parallel task graphs.

The real-time community is also providing extensive research on timing and schedulability analysis. The objective of timing analysis is to compute tight bounds (or probabilistic profiles) on the time needed to perform an operation executed in isolation. Schedulability analysis is then used to check analytically, at design or run time, whether all the timing requirements of the system will be met.

Static approaches for timing analysis typically infer timing properties from mathematical models and logical abstractions (e.g., [3]), while measurement-based techniques exploit the results of extensive simulations

to derive worst-case estimations. Hybrid techniques combine features from both static and measurement-based approaches while avoiding (as much as possible) their respective pitfalls (e.g. [30]). In what concerns schedulability analysis, different tests have been proposed for various kinds of workload and platform models, and for different scheduling algorithms. In its simplest form, a schedulability test is just a mathematical condition such that, if the condition is satisfied, then the system is deemed schedulable, i.e., all the deadlines will always be met at run-time. Unfortunately, there are still many open problems and NP-hard issues in the schedulability analysis of multi-core systems [6]. Furthermore, timing and schedulability analysis cannot be taken in isolation from the mapping approach, since the mapping of tasks to particular cores clearly impacts the timing analysis.

5 Conclusions

There is currently an increasing interest in the convergence of High-Performance and Embedded Computing domains. Not only new high-performance applications are being required by markets needing huge amounts of information to be processed within a bounded amount of time, but also embedded systems are increasingly concerned with providing higher performance in real-time, challenging the performance capabilities of current architectures. Meeting this dual challenge can only be provided by next-generation many-core embedded platforms, guaranteeing that real-time high-performance applications can be executed on efficient and powerful heterogeneous architectures integrating general-purpose processors with many-core computing fabrics.

This paper proposed a novel approach to address time-criticality and parallelisation by an integrated framework for executing workload-intensive applications with real-time requirements on top of next-generation commercial-off-the-shelf (COTS) platforms based on many-core accelerated architectures. The framework, that will be developed within the P-SOCRATES FP7 project [1], addresses the problem from both the HPC and real-time computing domains, integrating the extraction of task dependency graphs with timing information from the applications code, with real-time mapping and scheduling algorithms, along with the associated timing and schedulability analysis.

References

1. P-SOCRATES (Parallel Software Framework for Time-Critical Many-core Systems) http://p-socrates.eu.
2. *OpenMP Architectural Review Board, OpenMP Application Program Interface, Version 4.0, http://www.openmp.org*, July 2013.
3. AbsInt Corp. *aiT WCET analyser, http://www.absint.com/ait/*.
4. Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurr. Comput. : Pract. Exper.*, 23(2):187–198, February 2011.
5. S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese. A generalized parallel task model for recurrent real-time processes. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 63–72, 2012.
6. Sanjoy Baruah and Kirk Pruhs. Open problems in real-time scheduling. *Journal of Scheduling*, 13(6):577–582, 2010.
7. Luca Benini, Eric Flamand, Didier Fuin, and Diego Melpignano. P2012: building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 983–987, San Jose, CA, USA, 2012. EDA Consortium.
8. Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, September 1999.
9. Carlos Boneti, Roberto Gioiosa, Francisco J. Cazorla, and Mateo Valero. A dynamic scheduler for balancing hpc applications. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 41:1–41:12, Piscataway, NJ, USA, 2008. IEEE Press.
10. Javier Bueno, Luis Martinell, Alejandro Duran, Montse Farreras, Xavier Martorell, Rosa M. Badia, Eduard Ayguade, and Jesús Labarta. Productive cluster programming with ompss. In *Proceedings of the 17th international conference on Parallel processing - Volume Part I*, Euro-Par'11, pages 555–566, Berlin, Heidelberg, 2011. Springer-Verlag.
11. 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 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, TRUSTCOM '11, pages 1068–1075, Washington, DC, USA, 2011. IEEE Computer Society.

12. 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, 2011-03-01 2011.
13. T. Ungerer et. al. Merasa: Multi-core execution of hard real-time applications supporting analysability. In *In the IEEE Micro 2010, Special Issue on European Multicore Processing Projects, Vol. 30, No. 5*, Oct. 2010.
14. Mike Houston, Ji-Young Park, Manman Ren, Timothy Knight, Kayvon Fatahalian, Alex Aiken, William Dally, and Pat Hanrahan. A portable runtime interface for multi-level memory hierarchies. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, PPoPP '08, pages 143–152, New York, NY, USA, 2008. ACM.
15. Intel Corp. *Array Building Blocks, http://software.intel.com/en-us/articles/intel-array-buildingblocks.*
16. Intel Corporation. *Intel Many Integrated Core (MIC) Architecture - http://www.intel.com/content/www/us/en/architecture-and-technology/many-integratedcore/ intel-many-integrated-core-architecture.html*, last access Nov 2013.
17. Intel Corporation. *Intel Xeon Phi - http://www.intel.com/content/www/us/en/ processors/xeon/xeon-phi-detail.html*, last access Nov 2013.
18. R.M. Badia J. Labarta J.M. Perez, P. Bellens. Cellss: Making it easier to program the cell broadband engine processor, 2007.
19. Kalray Corporation. *Kalray MPPA 256 - http://www.kalray.eu/products/mppa-manycore/*, last access Nov 2013.
20. K. Lakshmanan, S. Kato, and R. Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 259–268, 2010.
21. David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
22. T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In *Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE*, pages 12–21, 1999.
23. José Manuel Marinho, Stefan M. Petters, and Marko Bertogna. Extending fixed task-priority schedulability by interference limitation. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, RTNS '12, pages 191–200, New York, NY, USA, 2012. ACM.
24. Mont-Blanc FP7 European Project, grant agreement 288777. *http://www.montblanc-project.eu/*, 2011 - 2014.
25. L. Nogueira, J.C. Fonseca, C. Maia, and L.M. Pinho. Dynamic global scheduling of parallel real-time tasks. In *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*, pages 500–507, 2012.
26. NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture, Version 2.0*, 2008.
27. Stephen L Olivier, Allan K Porterfield, Kyle B Wheeler, Michael Spiegel, and Jan F Prins. Openmp task scheduling strategies for multicore numa systems. *Int. J. High Perform. Comput. Appl.*, 26(2):110–124, May 2012.
28. OpenCL. *The open standard for parallel programming of heterogeneous systems - http://www.khronos.org/opencl/*, 2013.
29. parMERASA FP7 European Project - grant agreement 287519. *Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analysability, http://www.parmerasa.eu*, 2011 - 2014.
30. Rapita Systems Corp. *RapiTime, http://www.rapitasystems.com.*
31. A. Saifullah, K. Agrawal, Chenyang Lu, and C. Gill. Multi-core real-time scheduling for generalized parallel task models. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 217–226, 2011.
32. Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27(3):18:1–18:15, August 2008.
33. Herb Sutter.
34. R. Tieman. Algo trading: the dog that bit its master. *Financial Times*, March, 2008.
35. Tilera Corporation. *Tile Processor, User Architecture Manual, release 2.4, DOC.NO. UG101*, May 2011.
36. Sandra Wienke, Paul Springer, Christian Terboven, and Dieter an Mey. Openacc: first experiences with real-world applications. In *Proceedings of the 18th international conference on Parallel Processing*, Euro-Par'12, pages 859–870, 2012.
37. Henry Wong, Anne Bracy, Ethan Schuchman, Tor M. Aamodt, Jamison D. Collins, Perry H. Wang, Gautham Chinya, Ankur Khandelwal Groen, Hong Jiang, and Hong Wang. Pangaea: a tightly-coupled ia32 heterogeneous chip multiprocessor. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 52–61, 2008.