1

Introduction

Luís Miguel Pinho[1], Eduardo Quiñones[2], Marko Bertogna[3], Andrea Marongiu[4], Vincent Nélis[1], Paolo Gai[5] and Juan Sancho[6]

[1]CISTER Research Centre, Polytechnic Institute of Porto, Portugal
[2]Barcelona Supercomputing Center (BSC), Spain
[3]University of Modena and Reggio Emilia, Italy
[4]Swiss Federal Institute of Technology in Zurich (ETHZ), Switzerland; and University of Bologna, Italy
[5]Evidence SRL, Italy
[6]ATOS, Spain

This chapter provides an overview of the book theme, motivating the need for high-performance and time-predictable embedded computing. It describes the challenges introduced by the need for time-predictability on the one hand, and high-performance on the other, discussing on a high level how these contradictory requirements can be simultaneously supported.

1.1 Introduction

High-performance computing has been for a long time the realm of a specific community within academia and specialized industries; in particular those targeting demanding analytics and simulations applications that require processing massive amounts of data. In a similar way, embedded computing has also focused mainly on specific systems with specialized 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 high-performance and embedded computing 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. Although many systems focus on performance and handling large volumes of streaming data (with throughput and latency requirements), many application domains require real-time behavior [1–6] and challenge the computing capability of current technologies. Some examples are:

- In cyber-physical systems, ranging from automotive and aircrafts, to smart grids and traffic management, computing systems are embedded in a physical environment and their behavior obeys the technical rules dictated by this environment. Typically, they have to cope with the timing requirements imposed by the embedding domain. In the Large Hadron Collider (LHC) in CERN, beam collisions occur every 25 ns, which produce up to 40 million events per second. All these events are pipelined with the objective of distinguishing between interesting and non-interesting events to reduce the number of events to be processed to a few hundreds [7]. Similarly, bridges are monitored in real-time [8] with information collected from more than 10,000 sensors processed every 8 ms, managing responses to natural disasters, maintaining bridge structure, and estimating the extent of structural fatigue. Another interesting application is in intelligent transportation systems, where systems are developed to allow for fuel consumption reduction of railway systems, managing throttle positions, elaborating big amounts of data and sensor information, such as train horsepower, weight, prevailing wind, weather, traffic, etc. [9].
- In the banking/financial markets, computing 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 timeframe when some pre-defined patterns occur. Automated algorithmic trading programs now buy and sell millions of dollars of shares time-sliced into orders separated by 1 ms. Reducing the latency by 1 ms can be worth up to $100 million a year to a leading trading house. The aim is to cut microseconds off the latency in which these systems can reach to momentary variations in share prices [10].
- In industry, computing systems monitor business processes based on the capability to understand and process real-time sensor data from the factory-floor and throughout the whole value chain, with Radio Frequency Identification (RFID) components in order to optimize both the production and logistics processes [11].

The underlying commonality of the systems described above is that they are time-critical (whether business-critical or mission-critical, it is necessary to fulfill specific timing requirements) 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 meeting 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 analyses are needed to be able to make arguments of correctness, e.g., performing the required computations within well-specified bounds.

1.1.1 The Convergence of 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, high-performance computing (HPC) systems are traditionally designed to make the common case as fast as possible, without concerning themselves with the timing behavior (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.

In the last years, the above design choices are being questioned by the irruption of multi-core processors in both computing markets. 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 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 the *multi-core* system level an important challenge already faced by HPC designers at *multi-processor* system level: parallelization.

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 [12], as they allow scheduling multiple application services on the same processor, hence maximizing the hardware utilization while reducing cost, size, weight, and power requirements. However, real-time embedded applications with time-criticality requirements are still executed on simple architectures that are able to guarantee a predictable execution pattern while avoiding the appearance of timing anomalies [13]. 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 are described in the next chapter.

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. Many-core computing fabrics are being integrated with general-purpose multi-core processors to provide a heterogeneous architectural harness that eases the integration of previously hardwired 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. Similarly, in the real-time embedded domain, the Kalray Multi-Purpose Processor Array (MPPA), which includes clusters of quad-core CPUs coupled with many-core computing clusters. In both cases, the many-core fabric acts as a programmable accelerator. More recently, the Field-Programmable Gate Array (FPGA) has been used as a flexible accelerator fabric, complementing the above.

In this current trend, challenges that were previously specific to each computing domain, start to be common to both domains (including energy-efficiency, parallelization, compilation, and software programming) and are magnified by 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*.

1.1.2 Parallelization Challenge

Needless to say that many 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 have many-core embedded processors, such as the MPPA, been designed to be used in the HPC domain, nor have HPC techniques been designed to apply embedded technology. On the other hand, real-time methods to determine the timing behavior of an embedded system are not prepared to be directly applied to the HPC domain and these platforms, leading to a number of significant challenges.

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 [14], the currently dominant open standard for parallel programming of heterogeneous systems, or CUDA [15], 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 much functionality as possible into a single chip. Hence, they inherently share out 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 limited flexibility, complicating the development of embedded systems. Commercial off-the-shelf (COTS) components based on many-core architectures are likely to dominate the embedded computing market in the near future, even if complemented with custom function-specific accelerators. 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. This book outlines how 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 and high-performance.

1.2 The P-SOCRATES Project

The work described in this book was performed in the scope of the European project P-SOCRATES (Parallel Software Framework for Time-Critical Many-core Systems)[1], funded under the FP7 framework program of the European Commission. The project, finished in December 2016, aimed to allow applications with high-performance and real-time requirements to fully exploit the huge performance opportunities brought by the most advanced COTS many-core embedded processors, whilst ensuring predictable performance of applications (Figure 1.1). The project consortium included Instituto Superior de Engenharia do Porto (coordinator), Portugal, the Barcelona Supercomputing Centre, Spain, the University of Modena and Reggio Emilia, Italy, the Swiss Federal Institute of Technology Zurich, Switzerland, Evidence SRL, Italy, Active Technologies SRL, Italy and ATOS, Spain.

P-SOCRATES focused on combining techniques from different domains: the newest high-performance software techniques for exploiting task parallelism, the most advanced mapping and scheduling methodologies and timing

[1] htttp://www.p-socrates.eu

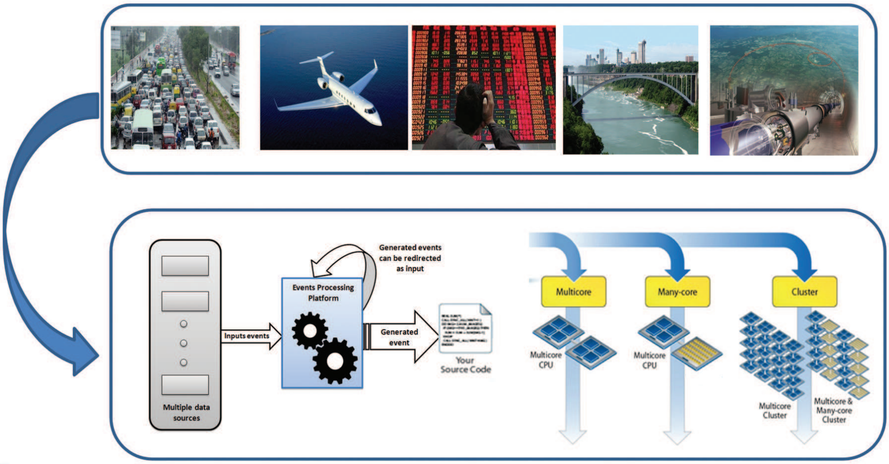

Figure 1.1 P-SOCRATES Global perspective.

and schedulability analysis techniques used in real-time embedded systems, and the low-energy many-core platforms of the embedded domain. This allowed taking important steps towards the convergence of HPC and real-time and embedded domains (Figure 1.2), providing predictable performance to HPC systems and increasing performance of real-time embedded systems.

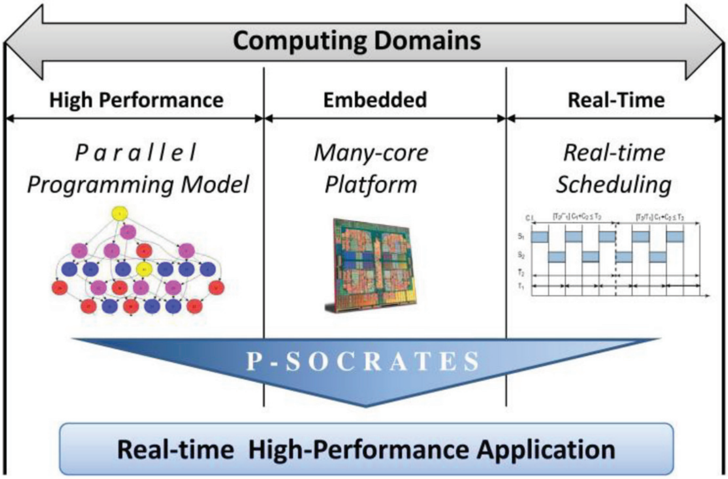

Figure 1.2 P-SOCRATES combines high-performance parallel programming models, high-end embedded many-core platforms and real-time systems technology.

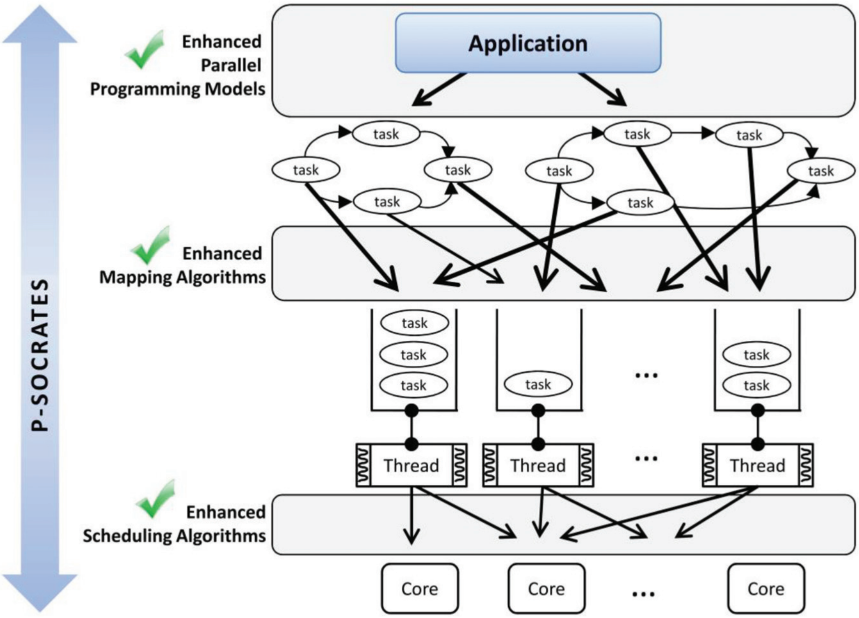
P-SOCRATES
Enhanced Parallel Programming Models
Application
task
Enhanced Mapping Algorithms
Thread
Enhanced Scheduling Algorithms
Core
...

Figure 1.3 Vertical stack of application decomposition.

P-SOCRATES developed a complete and coherent software system stack, able to bridge the gap between the application design with both high-performance and real-time requirements, and the hardware platform, a many-core embedded processor. The project provided a *new framework* to combine real-time embedded mapping and scheduling techniques with high-performance parallel programming models and associated tools, able to express parallelization of applications. The programming model used was based on the state-of-the-art OpenMP specification.

The software stack (shown in Figure 1.3) is able to extract a task-dependency graph from the application, statically or dynamically mapping these tasks to the threads of the operating system, which then dynamically schedules them on the many-core platform.

1.3 Challenges Addressed in This Book

1.3.1 Compiler Analysis of Parallel Programs

In order to enable predictable parallel performance to be analyzed, it is required that the application parallel graph is known, with control- and data-flow information needed for the analysis of the timing behavior of the parallel program. The extraction of this information should be as automatic as possible, to release the programmer from the burden of needing to understand the exact hardware details.

Chapter 3 addresses this challenge by presenting how this information can be obtained from the OpenMP tasking model, and how this information can be used to derive the timing properties of an application parallelized using this model.

1.3.2 Predictable Scheduling of Parallel Tasks on Many-core Systems

To be able to derive guarantees on the correct timing execution of parallel programs, it is required to provide appropriate mapping and scheduling algorithms of parallel computation in many-core platforms, together with deriving the associated offline analysis that enable determining if applications will meet their deadlines.

The challenge of real-time scheduling and schedulability analysis of parallel code is discussed in Chapter 4, which provides the substantial advances that the project has performed in the real-time scheduling and schedulability analysis of parallel graphs, using different scheduling models.

1.3.3 Methodology for Measurement-based Timing Analysis

The use of multi- and many-core platforms considerably challenges approaches for real-time timing analysis, required to determine worst-case execution time of the application code. In fact, the analysis of code execution time is considerably complex due to the interaction and conflicts between the multiple cores utilizing the same hardware resources (e.g., bus, memory, network).

Chapter 5 investigates the different available methods to perform this timing analysis in a many-core setting. After weighing the advantages and disadvantages of each technique, a new methodology is presented based on runtime measurements to derive worst-case estimates.

1.3.4 Optimized OpenMP Tasking Runtime System

The methodology presented in Chapters 3 to 5 of this book relies on the parallel computing abstraction provided by the OpenMP tasking model, and its conceptual similarities to the Direct Acyclic Graph (DAG) model, to achieve predictable task scheduling, requiring an efficient runtime support. However, a space- and performance-efficient design of a tasking run-time environment targeting a many-core system-on-chip is a challenging task, as embedded parallel applications typically exhibit very fine-grained parallelisms.

For that purpose, Chapter 6 presents the design and implementation of an OpenMP tasking run-time environment with very low time and space overheads, which is able to support the approach of the book.

1.3.5 Real-time Operating Systems

The run-time environment of Chapter 6 requires the underlying support of a Real-Time Operating System (RTOS) for many-core architectures. This operating system needs to both be able to execute multi-threaded applications in multiple cores, and also efficiently support a limited pre-emptive model, where threads are only pre-empted at the boundaries of OpenMP tasks.

Chapter 7 presents the re-design and re-implementation of the ERIKA Enterprise RTOS, aiming at an efficient execution on this kind of platforms. The new version of the RTOS allows us to share a single binary kernel image across several cores of the platform, reducing the overall memory consumption, and includes the new limited pre-emptive model.

1.4 The UpScale SDK

An outcome of the P-SOCRATES project was a complete and coherent software framework for applications with high-performance and real-time requirements in COTS many-core embedded processors. This software framework was publicly released under the brand of the UpScale SDK (Software Development Kit)[2]. The UpScale SDK includes the tools to manage the application compilation process, its timing analysis and its execution (Figure 1.4):

- *Compiler flow*. This flow has a twofold objective: (i) to guide the process to generate the binary that will execute on the many-core architecture and (ii) to generate the application DAG used for the timing analysis and run-time components.
- *Analysis flow*. This flow is in charge of deriving timing guarantees of the parallel execution considering execution time traces of the application running on the many-core platform and incorporated in the DAG. Timing

[2] http://www.upscale-sdk.com

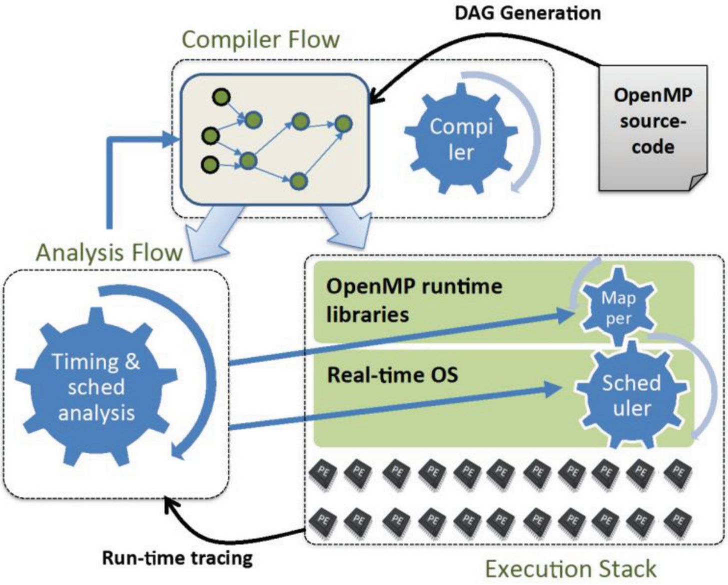

Figure 1.4 The UpScale SDK.

guarantees are derived by means of execution time bounds and a static scheduler or dynamic scheduler supported with response-time analysis.

- *Execution stack*. These two components are in charge of orchestrating the parallel execution of the application in a time-predictable manner, based on the DAG.

1.5 Summary

Providing high performance while meeting predictability requirements of real-time applications is a challenging task, which requires new techniques and tools at most if not all levels of the design flow and execution stack. This book presents the work which was done within the P-SOCRATES project to address these challenges, presenting solutions for deriving control- and data-flow graph of OpenMP parallel programs using the tasking model, algorithms for mapping and scheduling the OpenMP tasks into many-core platforms, and methods to perform both timing and schedulability analysis. The book also describes solutions for the runtime execution stack for real-time parallel computation, both at the level of the OpenMP runtime, as well as within real-time operating systems.

References

[1] Magid, Y., Adi, A., Barnea, M., Botzer, D., Rabinovich, E., "Application generation framework for real-time complex event processing," *32nd Annual IEEE International Computer Software and Applications (COMPSAC)*, 2008.

[2] Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N., "Stream reasoning and complex event processing in ETALIS," *Semantic Web 1*, 2009, IOS Press, pp. 1–5.

[3] Luckham, D. C., "*Event Processing for Business: Organizing the Real-Time Enterprise,*" John Wiley and Sons, 2011.

[4] Palmer, M., "*Real-Time Big Data and the 11 Principles of Modern Surveillance Systems*," http://streambase.typepad.com/streambase_stream_process/2011/07/in-his-tabbforum-article-dave-tolladay-eloquently-argues-that-real-time-surveillance-is-crucial-in-todays-high-frequency-t.html, last accessed February 2018.

[5] Twentyman, J., "*Sensory Perception,*" http://www.information-age.com/technology/information-management/1248733/sensory-perception, last accessed February 2018.

[6] Klein, R., Xie, J., and Usov, A., "Complex events and actions to control cyber-physical systems." In *Proceedings of the 5th ACM International Conference on Distributed Event-Based System (DEBS)*, 2011.

[7] Shapiro, M., "Supersymmetry, extra dimensions and the origin of mass: exploring the nature of the universe using petaScale data analysis," *Google TechTalk*, June 18, 2007.

[8] "*NTT DATA: Staying Ahead of the IT Services Curve With Real-Time Analytics*," https://www.sap.com/sea/documents/2012/10/66e7c78d-357c-0010-82c7-eda71af511fa.html, last accessed February 2018.

[9] "*SAP Enters Complex-event Processing Market*," http://www.cio.com.au/article/377688/sap_enters_complex-event_processing_market/, last accessed February 2018.

[10] Tieman, R., "Algo trading: the dog that bit its master", *Financial Times*, March 2008.

[11] Karim, L., Boulmakoul, A., Lbath, A., "Near real-time big data analytics for NFC-enabled logistics trajectories," *2016 3rd International Conference on Logistics Operations Management (GOL),* Fez, 2016, pp. 1–7.

[12] Ungerer, T., et. al. "MERASA: Multi-core execution of hard real-time applications supporting analysability," In *the IEEE Micro 2010, Special Issue on European Multicore Processing Projects*, Vol. 30, No. 5, October 2010.

[13] Lundqvist, T., Stenstrom, P., "Timing anomalies in dynamically scheduled microprocessors." In *IEEE Real-Time Systems Symposium*, 1999.

[14] "*OpenCL (Open Computing Language)*", http://www.khronos.org/opencl, last accessed February 2018.

[15] *NVIDIA*, https://developer.nvidia.com/cuda-zone, last accessed February 2018.