Conference Paper

High-performance parallelisation of real-time applications

Luís Miguel Pinho*
Vincent Nélis*
Eduardo Quinoñes
Paolo Burgio
Andrea Marongiu
Paolo Gai
Juan Sancho

CISTER-TR-170203

High-performance parallelisation of real-time applications

Luís Miguel Pinho*, Vincent Nélis*, Eduardo Quinoñes, Paolo Burgio, Andrea Marongiu, Paolo Gai, Juan Sancho

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

This paper presents an overview of theP-SOCRATES methodology and tools, instantiated in theUpScale SDK (Software Development Kit) for the development oftime-predictable high-performance applications. The proposedmethodology was designed to provide an integrated SDK to fullyexploit the huge performance opportunities brought by the mostadvanced many-core processors, whilst ensuring a predictableperformance and maintaining (or even reducing) developmentcosts of applications. The paper also provides the performanceresults of the application of the SDK in relevant embedded usecases.

High-performance parallelisation of real-time applications

Luís Miguel Pinho, Vincent Nélis
School of Engineering of the Polytechnic of Porto (ISEP)
Portugal
{lmp,nelis}@isep.ipp.pt

Eduardo Quinoñes
Barcelona Supercomputing Centre
Barcelona, Spain
eduardo.quinones@bsc.es

Paolo Burgio
University of Modena
Italy
paolo.burgio@unimore.it

Andrea Marongiu
ETH Zurich
Switzerland
a.marongiu@iis.ee.ethz.ch

Paolo Gai
Evidence Srl
Italy
pj@evidence.eu.com

Juan Sancho
ATOS
Spain
juan.sancho@atos.net

***Abstract*—This paper presents an overview of the P-SOCRATES methodology and tools, instantiated in the UpScale SDK (Software Development Kit) for the development of time-predictable high-performance applications. The proposed methodology was designed to provide an integrated SDK to fully exploit the huge performance opportunities brought by the most advanced many-core processors, whilst ensuring a predictable performance and maintaining (or even reducing) development costs of applications. The paper also provides the performance results of the application of the SDK in relevant embedded use-cases.**

***Keywords*—high-performance; real-time; parallelisation**

I. Introduction

Nowadays, the prevalence of electronic and computing systems in our lives is so ubiquitous that it would not be far-fetched to state that we live in a cyber-physical world dominated by computer systems. All these systems demand for more and more computational performance to process large amounts of data from multiple data sources, and some of them with guaranteed processing response times. In other words, systems required to deliver their results within pre-defined (and sometimes extremely short) time bounds. Examples can be found for instance in intelligent transportation systems for fuel consumption reduction in cities or railway, or autonomous driving of vehicles.

The computer electronic devices on which these systems depend on are constantly required to become more and more powerful and reliable. In order to cope with such performance requirements, chip designers have started producing chips with dozens or hundreds of cores, interconnected with complex networks on chip. This radical shift in the chip design paved the way for parallel computing: rather than processing the data sequentially, the cooperation of multiple processing elements within the same chip allowed systems to be executed concurrently, in parallel.

Unfortunately, the parallelization of the computing activities brought upfront many challenges, because it affects the timing behavior of systems as well as the entire way people think and design applications. Therefore, although many-core processors are promising candidates to improve the responsiveness of these systems, the interactions that the different computing elements may have within the chip, can seriously affect the performance opportunities brought by parallel execution. Moreover, providing timing guarantees becomes harder, because the timing behavior of the system running within a many-core processor depends on non-explicit, unwanted interference on shared resources (e.g., caches, memory banks), most of the time not know by the system designer. This makes system analysts to be struggled trying to provide timing guarantees for such platforms. Finally, most of the optimization mechanisms buried deep inside the chip are geared only to increase average performance and execution speed rather than providing predictable time behavior.

P-SOCRATES (Parallel Software Framework for Time-Critical Many-core Systems) [1] was a European project, which has developed a novel methodology to facilitate the deployment of standardized parallel architectures for real-time applications. This methodology is implemented (based on existent models and components) to provide an integrated software development kit to fully exploit the huge performance opportunities brought by the most advanced many-core processors, whilst ensuring a predictable performance and maintaining (or reducing) development costs of applications.

The paper provides an overview of the P-SOCRATES methodology and tools (the UpScale Software Development Kit), as well as the results of its application on relevant embedded use-cases. This evaluation showcases that the integration of time-predictability in the design methodology and software stack is performed without average performance loss and with similar energy consumption.

II. The P-SOCRATES Methodology

P-SOCRATES considered a holistic approach [2] with a complete and coherent software system stack, able to bridge the gap between application design and hardware many-core platform (Figure 1). The project combined a parallel programming framework with real-time techniques and operating systems.

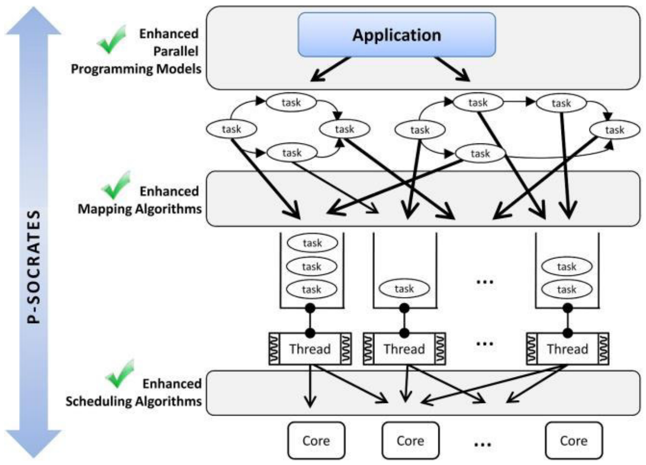
P-SOCRATES
Enhanced Parallel Programming Models
Application
Enhanced Mapping Algorithms
Enhanced Scheduling Algorithms
task
Thread
Core
...

Fig. 1. P-SOCRATES holistic approach

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 [3] 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. It also provides directives to define data dependencies between the tasks and more recently an offloading model where a part of the code can be offloaded to an accelerator.

The model of the project considers real-time applications with multiple real-time concurrent activities. Each one of these is a recurrent loop of real-time jobs. Each job starts executing in one core, and then offloads the computational intensive parts to a many-core accelerator cluster.

High-performance parallel frameworks (Figure 2) base scheduling decisions on information available at run-time – i.e., the task dependency graph and processor resources availability – 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.

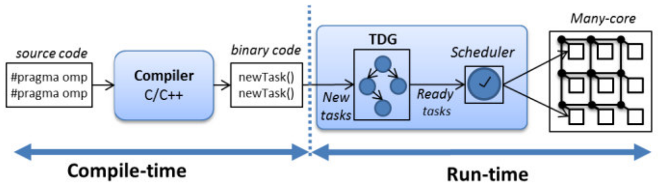

Fig. 2. High-performance computing approach

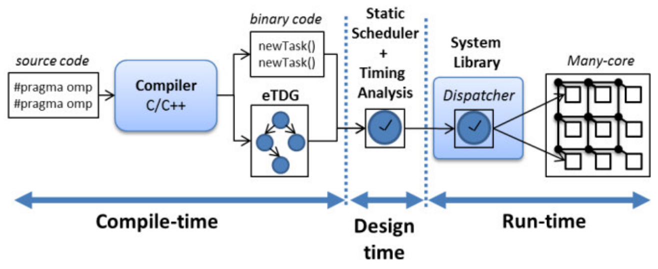

Fig. 3. P-SOCRATES apporach to extract information at design-time

In order to provide real-time guarantees without 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 (Figure 3). 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. Figure 4 shows these two extremes on a spectrum of potential solutions – the more we fix the usage of the system resources, tighter analysis can be provided, thus increasing guaranteed performance. However, caused by preventing dynamicity, average performance is decreased.

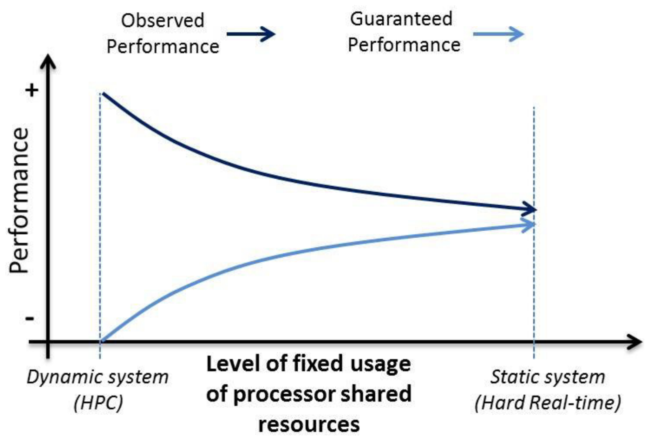

Fig. 4. Performance vs. predictabiltiy spectrum

The approach of the project is thus to, at compile time, extract a Direct Acyclic Graph of the data-flow and control-flow dependencies between the OpenMP tasks. This graph is extended to consider OpenMP task parts, which is task code executed between two task scheduling points. 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 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. Thus, if there is an unknown data-dependency, the construction of the graph 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 [4].

Afterwards, a new timing analysis approach is used to annotate these task parts with worst-case execution time estimates. The WCET estimates are based on a measurement-based approach, considering that the WCET of an OpenMP task is based on the sum of two terms: the first one is its intrinsic execution time, i.e. the time it spent executing the instructions of the code, whereas the second part is due to the stalling time, i.e. the time spent by the analyzed code waiting for a shared software or hardware resource to become available. This information is then used by the mapping and scheduling algorithms [5] to properly select the most suitable resource allocation strategies.

Two resource allocation approaches are considered. In the dynamic approach, OpenMP task-to-thread mapping is based on a global queue, and thread scheduling is also global within one parallel cluster, allowing migrations. This allows to achieve higher performance (with load balancing) but with more complex and pessimistic response time analysis. In the second configuration, a mapping algorithm statically builds the required run-time configuration, efficiently assigning tasks-to-threads in order to guarantee timing requirements. Thread scheduling uses a partitioned per-core scheduler. Note that in both cases the scheduling approach is based on a limited preemptive model, which only allows preemption at OpenMP task scheduling points, reducing cache invalidations issues.

III. THE UPSCALE SDK

The UpScale Software Development Kit (SDK) [6] incorporates all software components required to execute, in an efficient and time predictable way, a system composed of real-time tasks, which internally use the OpenMP tasking model for acceleration.

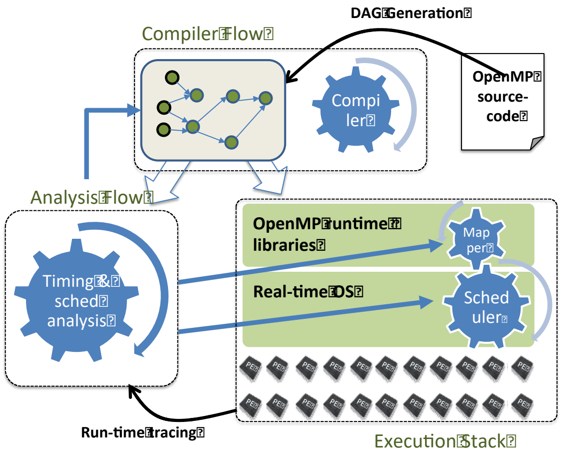

Fig. 5. UpScale SDK

Figure 5 shows the SDK components described below:

- Compiler flow. This component: (1) generates the binary application image that will execute on the many-core accelerator and (2) generates the application Direct Acyclic Graph (DAG) used for the timing analysis and run-time components.
- Analysis flow. This component 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 guarantees are derived by means of execution time bounds and static scheduler or dynamic scheduler supported with response time analysis.
- Execution stack. In charge of orchestrating the parallel execution of the application in a time predictable manner, based on the DAG.

The compiler flow, orchestrated by the Mercurium source-to-source compiler [7], is in charge of compiling each of the OpenMP applications that compose the system. Moreover, the compiler will generate a Direct Acyclic Graph (DAG) of the control-flow and data-flow dependencies of the OpenMP tasks of the application, referred to as OpenMP-DAG, including all the information required to perform the timing and schedulability analysis.

The Analysis flow, orchestrated by the Analyzer, based in the QuickTrace tool [8], is in charge of annotating each DAG node with estimates of worst-case execution time (WCET), with the methodology referred in Section I, feeding a schedulability analysis tool, which is able to provide the timing guarantees to the applications.

The execution stack is composed by a Real-Time Operating System (RTOS) and runtime library that synergistically orchestrate the parallel execution in a time predictable manner as estimated by the analysis flow. It consists of a lightweight OpenMP runtime implementing the OpenMP tasking model [9], on top of the ERIKA real-time operating system [10]. The execution stack can be configured in one of the two approaches (dynamic and static) presented in Section I.

This SDK is released open source, under commercial-friendly open source licenses.

IV. EVALUATION

The methodology and SDK has been evaluated over three use case applications defined in the project, two in the embedded domain, which are presented in this paper. The SDK has been used in order to enhance the parallelization strategies applied to the applications, and a round of evaluations has been done to compare the performance against other software setups, in particular when a non-sequential strategy is followed or using the standard SDK provided by the manufacturer.

A. Setup

The hardware board used during the experiments is the Kalray Bostan MPPA [11], which is the second generation of Kalray's manycore processor family. Bostan brings an ASIC-level of performance (high computing, low power consumption and real-time processing) with full programmability. Bostan runs 288 C/C++ programmable cores, optimized for networking and storage applications, and includes high speed interfaces like 80GE and PCIe x16 lane Gen3 directly connected to the large matrix of 288 cores and 128 crypto co-processors, whose core is a 64-bit Very Long Instruction Word (VLIW).

The processor is divided into clusters: 4 clusters which have more powerful cores, intended to perform the input-output with the outside (IO cores) and 16 computing clusters, each with 16 "working" cores. In the model considered in the project, applications start executing in the IO cores, and offload parallel computation into the computing clusters. A major restrictive factor of this model is that each cluster only provides 2Mb of local memory. This impacts in the parallelization strategies used.

The hardware manufacturer provides its own development environment for this board, the Kalray AccessCore™ SDK, providing a standard C/C++/Fortran based environment including all the tools to quickly develop, debug, and optimize high-performance applications for the MPPA processor.

Three different SW setups were used to evaluate the use case applications:

- Sequential – In sequential mode the application does not take advantage of the many-core architecture embedded in the Kalray board. On the contrary, the application does not spawn any cluster cores, leveraging the execution just to one core (IO core in the Kalray nomenclature).
- Parallel Kalray baseline – In baseline mode the application has been parallelized using the annotations and instructions provided by the HW manufacturer, using the Kalray AccessCore SDK 2.2.2. In one of the use cases it was also possible to compare with a low-level thread-based parallel baseline.
- Parallel P-SOCRATES – The consortium ported the SDK to the Kalray Bostan MPPA board. The evaluation focus is on the use of the SDK with a dynamic scheduling approach, where better average performance is expected, due to the higher adaptability.

B. *The Infra-red application use case*

The first use case is a computational intensive part of an application for pre-processing sampling of infra-red H2RG detectors in satellites [12]. The infra-red application, developed by the ESA under the license GPL, is for the Euclid spacecraft, whose objective is to better understand the geometry of dark energy and dark matter by measuring the red-shift of galaxies at varying distances from Earth. The computational intensive part is composed of seven stages (Figure 6) executed sequentially, i.e. one after the other, within a loop iterating given number of times.

These processing steps require different memory requirements, ranging from 8 to 48.5 Mb. The memory consumed by the application makes it impossible to be processed in a single MPPA cluster. The stages must be executed in order because the same frame is processed among the different stages. Hence, we use a wave-front parallelization strategy, in which the frame is divided in N × N blocks, assigning each of them to a different task.

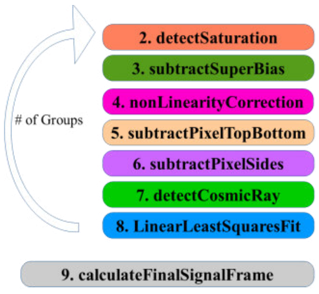

Fig. 6. Structure of infra-red application use case

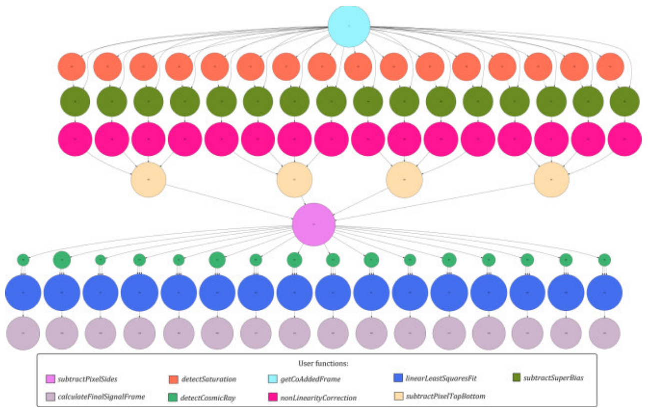

Fig. 7. Parallel decomposition of the infra-red application use case

Ideally, for each stage, the computation of each block would be assigned to a different task. Unfortunately, the data-flow defined by the infra-red application, as well as the requirements of some of the stages (which need to process all of a column or the full frame) makes it not possible. Overall, assuming that the sensor frame is divided in 64 blocks, we can distinguish between four different execution phases executed one after the other and summarized as follows (Figure 7):

- During phase 1, 16 parallel executions on clusters are offloaded, each processing 4 blocks in parallel with the three first processing stages.
- During the cluster phase 2, 8 parallel execution on clusters are offloaded, each processing 8 blocks (corresponding to 1 frame column) sequentially by the fourth stage.
- During the IO phase, the complete frame is processed in a single IO core in the fifth stage (requires all frame).
- During the cluster phase 3, 64 executions on clusters are offloaded (being able to execute only 16 in parallel), each processing 1 block with the two last stages in the loop.
- Finally, once the stages within the loop finish, the cluster phase 4 offloads 64 executions on clusters (being able to execute only 16 in parallel), each processing 1 block by the final function.

Table I provides the performance evaluation of the use case. It compares the average execution time, the maximum observed execution time, (also known as high *water-mark* in the critical real-time embedded domain) and the standard deviation of the infra-red application in milliseconds (ms), when parallelizing the application using both the native Kalray SDK, and the SDK developed within the project.

The execution time measurements presented have been obtained by executing the application 100 times. We consider that the loop iterates 5 times (#Groups = 5). We consider the observed execution time of the end-to-end execution of the application (labelled as TOTAL), and each of the execution phases within one loop iteration as presented (labelled as Cluster Phase 1, Cluster Phase 2, IO Phase, Cluster Phase 3 and Cluster Phase 4). The initialization and end phase of the application are not accounted for.

The table compares the average performance and the performance speed-up, considering as a baseline the execution time of the sequential version of the application executed in one IO core. The table also presents the maximum theoretical speed-up that the MPPA can exhibit. This maximum theoretical speed-up corresponds to the number of cores that can potentially execute in parallel.

TABLE I. PERFORMANCE EVALUAITON OF INFRA-RED USE CASE

	Execution time (ms)			Speed-up		
	Sequential	MPPA Native SDK	P-SOCRATES SDK	MPPA Native SDK	P-SOCRATES SDK	Maximum theorical
TOTAL	**63140**	**8872,4**	**9093,2**	**7,1**	**6,9**	**256**
Cluster Phase 1	2710	162,64	150,34	16,7	18,0	48
Cluster Phase 2	562	147,56	120,08	3,8	4,7	8
IO Phase	612	612	612	1,0	1,0	1
Cluster Phase 3	8554	804,9	879,1	10,6	9,7	16
Cluster Phase 4	950	236,9	285,6	4,0	3,3	16

The infra-red application achieves similar performance speed-ups when parallelizing and executing the application by using the Kalray SDK and the P-SOCRATES SDK, with 7.1x and 6.9x respectively. This speed-up however, is very far from the theoretical one (256x) due to the very limited parallelism exposed by the application in each of the execution phases.

When analyzing each of the execution phases, speed-ups are similar for the Kalray and P-SOCRATES SDKs, with speed-ups of 16.7x and 18x during the first cluster phase, 3.8x and 4.7x during the second phase, 10.6x and 9.7x during the third phase and 4.0x and 3.3x during the fourth phase. The theoretical speed-up accounts for the maximum number of cores that can be used in each phase. Thus, in the first phase executions are spawned across the 16 clusters, each using up to 4 cores, and so 16*4 = 48. It is important to note that the speed-up also accounts for the impact of transferring code and data at every offloading and the overhead of the OpenMP run-time, very similar in both the Kalray and P-SOCRATES SDKs.

Overall, the P-SOCRATES SDK does not degrade the average performance speed-up, being comparable to the MPPA native SDK.

Table II shows the average power (in W) and the energy consumption (in J) of the infra-red application when executing it sequentially (in one IO core) and in parallel, using the Kalray native SDK and the project SDK. The average power measured in both MPPA and P-SOCRATES SDK are very similar, resulting in similar energy consumptions. It is important to note that numbers presented in the table have been obtained with the k1-power tool provided by Kalray, and so the execution time values differ from the ones obtained in the previous table which are measured with instrumented code.

TABLE II. ENERGY EVALUAITON OF INFRA-RED USE CASE

	Sequential	MPPA Native SDK	P-SOCRATES SDK
Acquisition time (s)	75,84	30,38	24,81
Average power (W)	7,27	7,77	8,02
Energy (J)	551,14	235,93	199,12

C. *The Complex Event Processing Engine use case*

A Complex Event Processing (CEP) engine can be briefly described as a software solution to collect raw data streams, process them, and derive meaningful value-added information to a third party. The pCEP (parallel Complex Event Processing) is a parallel implementation of the lightweight µCEP of Atos [13], ported to the MPPA architecture.

The architecture of the pCEP is composed by three main elements (Figure 8). The Event Collector module responsible for acquiring data from external sources and converting them into Events; the output is the responsibility of the Complex Event Publisher module. The core of the solution is the Complex Event Processor module, which entails the case study of the work. In brief, as depicted in Figure 8, this module receives Events that trigger rules (conditions) and output Complex Events, both being previously specified. Those rules invoking complex functions make use of the Instruction Evaluator sub-module, which is the part of the pCEP with the most demanding and time consuming features. Particularly, the Complex Event Processor module has been rewritten to offload to the accelerator devices the catalogue of complex functions that can be used.

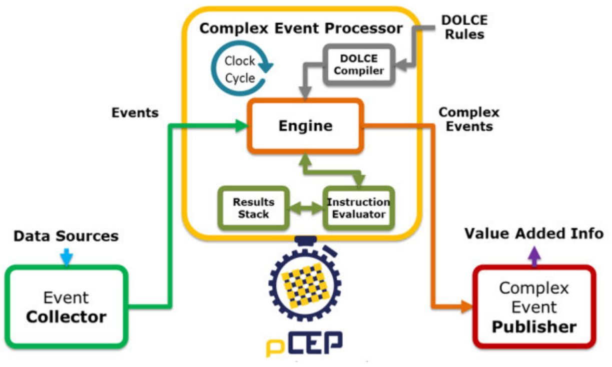

Fig. 8. Structure of the pCEP

Initial experiments have shown that, as expected, parallelizing the functions was only relevant if computational intensive. Therefore, the evaluation was performed with a complex function which implements a Fast Fourier Transform (FFT) algorithm on a buffer of events.

Following the same procedure as for the previous use cases, Table III compares the average execution time and the maximum observed execution time of the pCEP engine when it is executed with a list of events that trigger rules invoking the execution of this complex function. This implies spawning to compute clusters and offloading parallel executions of the complex functions on the accelerator devices. Sequential vs. native Kalray SDK vs. project SDK versions are compared.

TABLE III. PERFORMANCE EVALUAITON OF PCEP USE CASE

	Execution time (secs.)			Speed-Up	
Tasks	Sequential	Kalray SDK	P-Socrates SDK	Kalray SDK	P-Socrates SDK
	135,65				
1		152,77	131,45	0,89	1,03
2		76.28	65,73	1,78	2,06
4		38,01	32,82	3,57	4,13
8		18.99	16,41	7,14	8,27
16		9,53	8,29	14,24	16,37
32		9,53	8,29	14,24	16,36
64		9,53	8,29	14,24	16,37
128		9,53	8,28	14,24	16,37
256		9,53	8,29	14,24	16,37

For each input event, we launched the execution of a set of 256 FFTs of 512 elements. Since the usage of different number of threads changes the performance, also the experiment cases using from 1 up to 256 parallel tasks in the same cluster have been evaluated. Results show that performance improvement (speed-up against sequential) is achieved (stabilizing when the cluster is saturated). The speed-up increases when the number of tasks grows, achieving the best performance (speed-up x16) when using 16 tasks or more, since all the threads available in the clusters are used.

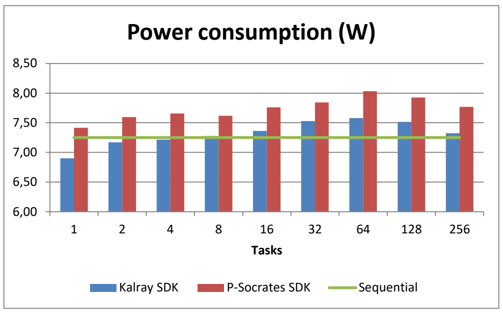

Fig. 9. Power consumption of the pCEP application

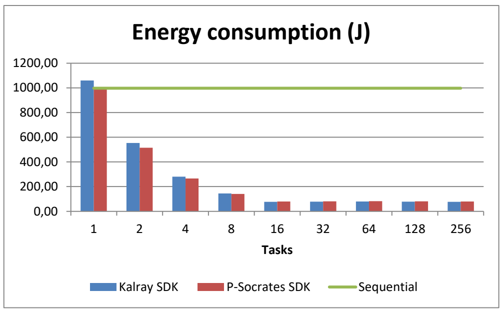

Fig. 10. Energy consumption of the pCEP application.

Following the same methodology as in the previous use case, we obtain the power and energy consumption of the accelerator using the 'k1-power' tool provided by Kalray. The power consumption remains quite constant for all the analyzed cases, with values around 7 to 8 Watts (Figure 9). Because the execution time decreases with the number of tasks used, especially with the P-SOCRATES implementation, it is in those cases that the energy consumption is significantly slower (see Figure 10).

V. CONCLUSIONS

Integrating time-predictability in high-performance embedded computing is of paramount importance to cope with the requirements of future real-time applications. However, this integration brings difficult challenges that need to be addressed since high-performance hardware and software stacks are not designed for predictability.

The P-SOCRATES project tackled this important challenge by devising a methodology and a Software Development Kit (the UpScale SDK) that allows to reason on the timing and schedulability analysis of real-time high-performance applications. Furthermore, the project evaluation shows that the dynamic configuration of the software stack, allows to achieve the same average performance and energy consumption than the default SDK of the considered platform.

ACKNOWLEDGMENT

The work in this paper has been financially supported by the European Commission through the PSOCRATES project (grant agreement 611016).

REFERENCES

[1] P-SOCRATES Project, http://www.p-socrates.eu, last accessed January 2017.

[2] Pinho, L, Nélis, V, Yomsi, P, Quiñones, E, Bertogna, M, Burgio, P, Marongiu, A, Scordino, C, Gai, P, Ramponi, M, Mardiak, M, "P-SOCRATES: A parallel software framework for time-critical many-core systems", Microprocessors and Microsystems, Elsevier. Nov 2015.

[3] OpenMP Consortium, OpenMP Specification. http://www.openmp.org/ wp-content/ uploads/openmp-4.5.pdf, last accessed January 2017.

[4] AbsInt Corp. aiT WCET analyser, http://www.absint.com/ait/, last accessed January 2017.

[5] M. A. Serrano, A. Melani, M. Bertogna and E. Quinones, "Response-time analysis of DAG tasks under fixed priority scheduling with limited preemptions," 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2016, pp. 1066-1071.

[6] UpScale SDK, htttp:www.upscale-sdk.com, last accessed January 2017.

[7] BSC, Mercurium source-to-source compiler, https://pm.bsc.es/mcxx, last accessed January 2017.

[8] Nélis, V, Pinho, L, "Using Quicktrace to collect runtime execution traces easily and automatically", RTSS@Word Demo Session, IEEE Real-Time Systems Symposium (RTSS 2015). 1 to 4, Dec, 2015. U.S.A..

[9] D. Cesarini, A. Marongiu and L. Benini, "An optimized task-based runtime system for resource-constrained parallel accelerators," 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2016, pp. 1261-1266.

[10] ERIKA Enterprise, http://erika.tuxfamily.org/, last accessed January 2017.

[11] Kalray, MPPA Processor, http://www.kalrayinc.com/kalray/products/, last accessed January 2017

[12] ESA, NIR HAWAII-2RG benchmark, http://www.esa.int/ Our_Activities/Space_Engineering_Technology/Onboard_Data_Process ing/General_Benchmarking_and_Specific_Algorithms, last accessed January 2017.

[13] A. Akbar, F. Carrez, K. Moessner, J. Sancho, J. Rico, "Context-aware stream processing for distributed IoT applications", 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), http://ieeexplore.ieee.org/document/7389133/