CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Conference Paper

Methodologies for the WCET Analysis of Parallel Applications on Many-core Architectures

Vincent Nélis
Patrick Meumeu Yomsi
Luis Miguel Pinho

CISTER-TR-150607

2015/08/26

Methodologies for the WCET Analysis of Parallel Applications on Many-core Architectures

Vincent Nélis, Patrick Meumeu Yomsi, Luis Miguel Pinho

CISTER Research Center

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: nelis@isep.ipp.pt, pamyo@isep.ipp.pt, lmp@isep.ipp.pt

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

Abstract

There is an increasing eagerness to deploy and execute parallel applications on many-core infrastructures, preserving the time-predictability of the execution as required by real-time practices to upper-bound the response time of the embedded application. In this context, the paper discusses the application of the currently-available WCET analysis techniques and tools on such platforms and with highly parallel activities. After discussing the pros and cons of all different methodologies for WCET analysis, we introduce a new approach that is developed within the P-SOCRATES project.

Methodologies for the WCET Analysis of Parallel Applications on Many-core Architectures

Vincent Nélis, Patrick Meumeu Yomsi, Luís Miguel Pinho
CISTER/INESC-TEC, Polytechnic Institute of Porto, Portugal
Email: {nelis, pamyo, lmp}@isep.ipp.pt

***Abstract*—There is an increasing eagerness to deploy and execute parallel applications on many-core infrastructures, preserving the time-predictability of the execution as required by real-time practices to upper-bound the response time of the embedded application. In this context, the paper discusses the application of the currently-available WCET analysis techniques and tools on such platforms and with highly parallel activities. After discussing the pros and cons of all different methodologies for WCET analysis, we introduce a new approach that is developed within the P-SOCRATES project.**

I. Introduction

Traditionally, High Performance Computing (HPC) has been the realm and primary focus of specialized industries and specific groups within academia as it demands analytics and simulation applications that require large amounts of data to be processed. Similarly, researchers and industry in the embedded computing (EC) domain have focused mainly on specific systems with specialized and fixed set of functionalities for which timing requirements prevailed over performance requirements. Today, both the HPC and EC domains are broadening their initial focus to other application areas due to the ever-increasing availability of more powerful processing platforms, but therefore they need affordable and scalable software solutions [1], [2].

The need for energy-efficiency (in the HPC domain) and flexibility (in the embedded computing domain), that come 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.

Today, 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. 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) [3], [4]. Examples of many-core architectures in the HPC domain include the Intel MIC [5] and Intel Xeon Phi [6] (features 60 cores).

Similarly, the real-time embedded domain has seen the emergence of the 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) [8], which includes four quad-core CPUs coupled with a many-core processor. One can also cite the Parallela from Epiphany and the Keystone II from Texas Instrument. In most cases, the many-core fabric acts as a processing accelerator [9].

The introduction of such platforms has set up the basic environment that allowed for the deployment of new types of applications sharing objectives and requirements from both the EC and HPC domains. For such applications, the correctness of the result depends on both performance and real-time requirements, and the failure to meet those is critical to the functioning of the system. Real-time Complex Event Processing (CEP) systems[1] [10] are an example of such applications; they challenge the performance capabilities by crossing the boundaries between the two domains.

It is in that context that the project P-SOCRATES started in October 2013. P-SOCRATES stands for "Parallel Software Framework for Time-Critical Many-core Systems". It is an European project which intends to allow current and future applications with high-performance and real-time requirements 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 purpose of P-SOCRATES is to develop an entirely new design framework, from the conceptual design of the system functionality to its physical implementation, to facilitate the deployment of standardized parallel architectures in all kinds of systems. We refer the reader to [11] for the details about the software stack and computation model proposed by the project consortium to parallelize applications on a many-core architecture while providing guarantees on their response times.

The application use-cases investigated in P-SOCRATES share both high performance and real-time requirements. That is, they demand for massive parallel execution and are subject to strict timing requirements according to which they must be guaranteed to "react" within pre-defined time bounds. The said "reaction" may be understood as simply outputting the results of a basic computation, but may also mean engaging in complex interactions with the surrounding environment. Here we can see that these two requirements are somewhat orthogonal. While strict timing requirements advocate the use of simple and predictable hardware architectures that allow for the computation of tight upper-bounds on the software response time, the high performance requirements demand for more computational performance, which weighs in favor of specialized, complex, and optimized multi-core and many-core processors on which the execution of the application can be massively parallelized. However, it is not straightforward how event-based embedded applications can be structured in order to take advantage and fully exploit the parallelization opportunities and achieve higher performance and energy-

[1] A real-time CEP system is a system in which the data coming from multiple event streams are correlated in order to extract and provide meaningful information.

efficient computing.

The P-SOCRATES project envisions this necessity to bring together next-generation many-core accelerators from the embedded computing domain with the programming models and techniques from the high-performance computing domain, supporting this with real-time methodologies to provide timing predictability. Among the many technical challenges set up by the goals and objectives of the projects, there is the need of designing new techniques to provide guarantees on the worst-case execution time (WCET) of the application. There are different approaches to WCET analysis, commonly referred to as static, measurement-based, hybrid, and probabilistic. The objective of this paper is twofold.

1) We want to summarize these methodologies and discusses their up and downsides.

2) We investigate if and how these techniques could be applied when considering an execution environment as complex as that defined within the P-SOCRATES project.

Organization—The rest of the paper is organized as follows. Section II presents an overview of the state-of-the-art WCET techniques. Specifically, this section discusses the advantages and drawbacks of *static*, *measurement-based*, *hybrid* and *probabilistic* WCET analysis approaches. Then, Section III outlines the application model and software stack defined in P-SOCRATES. Section IV discusses which of the summarized methodologies could be suitable and applicable in the context of P-SOCRATES. Section V presents the P-SOCRATES envisioned methodology and its innovative aspect and finally, Section VI concludes the paper.

II. Overview of the state-of-the-art WCET techniques

Most of timing analysis tools focus only on determining an upper-bound on the (WCET) of a program or a code fragment that runs without interruption and in isolation, i.e. these tools do not consider all the interferences that the analyzed program may suffer when run concurrently with other tasks or programs on the same hardware platform. This means that those tools typically ignore all execution interferences due to the contention for shared software resources (e.g., data shared between several tasks) and shared hardware resources (e.g., shared interconnection network). Interferences from the operating system (OS) which frequently re-schedules and interrupts the programs are also ignored by WCET analyzers. All these interactions between the analyzed task and the OS, and between the analyzed task and all the other tasks running in the system, are usually handled and analyzed separately and factored in the analysis at a later stage, e.g. in the schedulability analysis. In any case, for the timing requirements to be fulfilled, by no means these interferences can be ignored at the schedulability analysis level.

WCET analysis can be performed in a number of ways using different tools, but the main methodologies employed can be broadly classified in four categories: (1) Static analysis techniques, (2) Measurement-based analysis techniques, (3) Hybrid analysis techniques, and (4) Probabilistic analysis techniques. Note that the first three methodologies are usually recognized as equally important and efficient whereas the fourth one is more recent and thus less results are available as of now. Broadly speaking, measurement-based techniques are suitable for software that is less time-critical and for which

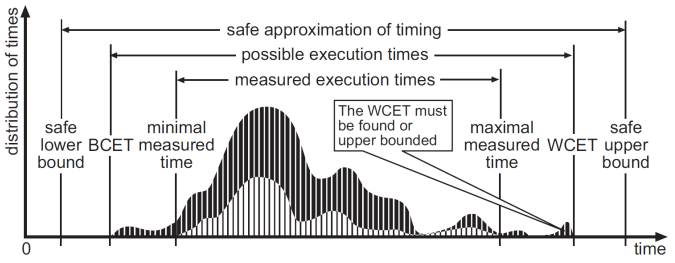

Fig. 1. Example distribution of execution time (picture taken from [12])

the average-case behavior (or a rough WCET estimate) is more meaningful or relevant than an accurate estimate. For example, systems where the worst-case scenario is extremely unlikely to occur and/or the system can afford to ignore it if it does occur. For highly time-critical software, where every possible execution scenario must be covered and handled, the WCET estimate must be as reliable as possible and static or hybrid methods are therefore more appropriate. Probabilistic analysis techniques are also designed for safety-critical systems to derive safe estimated execution time bounds but in our opinion they are not yet sufficiently mature to report on their efficiency and applicability.

Figure 1 illustrates how different timing estimates relate to the WCET and best-case execution time (BCET) for a single sequential program run in isolation. This contrived example illustrates the dependence of the execution time of a program on (1) the value of its input parameters and (2) its interactions with the system resources. The dark bars show the actual probability distribution of its execution time, delimited by its minimum (the BCET) and maximum (the WCET). The lighter bars show the set of execution times that have been measured during simulations, which is a subset of all executions. Even in "simple" single-core systems, for both static analysis tools and measurements-based tools, in most cases the program state space and the hardware complexity are too large to exhaustively explore all possible execution scenarios of the program. This means that the measured times are likely to be optimistic and the estimated times are likely to be pessimistic, i.e., the measured times will in many cases overestimate the actual BCET and underestimate the actual WCET, while the approximated estimated times will in many cases underestimate the actual BCET and overestimate the actual WCET. The next four sections introduce each methodology and discuss their respective benefits and drawbacks.

A. Static WCET analysis techniques

Phase 1: Flow analysis. During this phase all the information about the potential program execution paths is derived. Specifically this phase builds a control flow graph that capture all these execution paths with the aim of identifying the worst path w.r.t the execution time. The flow analysis mostly focuses on computing upper bounds on the number of iterations in each loop and on the depth of every recursive call. Although automatic methods to compute these bounds have been proposed by the research community, most of the tools that are available today still require the code to be annotated by the application developer. Another main objective of the flow analysis is to identify infeasible execution paths, i.e. paths that are executable in theory according to the control flow analysis, but not in practice considering the semantics of the

program and the set of possible values of the input arguments. Discarding infeasible paths at an early stage of the analysis considerably reduces the search space when trying to identify the longest path.

Phase 2: Low-level analysis. During this phase the execution time of every atomic part of the code (e.g., instructions, basic blocks or larger code sections) is estimated from a model of the target architecture. Low-level analysis methods typically use models of all the hardware components and their arbitration policies, including CPU caches, cache replacement policies, write policies, instruction pipeline, memory bus and their arbitration policies, etc. These models are usually expressed in the form of complex mathematical abstraction for which a worst-case behavior can be inferred.

Phase 3: Final WCET estimation. During this phase, the derived flow (Phase 1) and timing information (Phase 2) are combined into a WCET estimate.

Pros & Cons There are few advantages and drawbacks of using static analysis techniques that rely on mathematical models.

⊕ It eliminates the need of having the actual hardware available, which removes the non-negligible effort and cost of acquiring and setting up the target platform.

⊕ It enables safe WCET upper-bounds to be derived without actually running the program on the target platform while still considering the influence of the state changes in the underlying hardware [13]. State changes include, e.g. a cache line being evicted, a pipeline being totally flushed out, etc.

⊖ These approaches rely heavily on having an accurate model of the timing behavior of all the components of the target hardware architecture as well as their management policies, including modeling features like pipelines and caches that substantially affect the execution time of the code being executed. Although the embedded market used to be traditionally dominated by simple and predictable processors (which used to be moderately "easy" to model and allowed for deriving safe and tight bounds), with the increased computational needs of modern embedded systems, designers have moved to more complex processors which are now mainly designed for performance and not for predictability. For this new generation of processors, designing an accurate hardware model is very challenging, as all the intricacies contributing to the variation in the tasks execution time (e.g., caches, pipelines, out-of-order execution, branch prediction, automatic hardware prefetching, etc.) should be captured by the model to provide safe and sufficiently tight bounds. Because it is hardly feasible to accurately model all these acceleration mechanisms and their operation, static methods typically forbid their use and are struggling to adapt to modern hardware architectures.

⊖ Besides the difficulty of modelling all these performance-enhancement hardware features, it must also be noted that generally, chip manufacturers do not publish the details of their internal workings, which further complicates/makes impossible the design of an accurate model.

⊖ Although static approaches have the advantage of providing safe WCET bounds, they can be very pessimistic at times. This is because generally, each hardware resource is modelled separately and all the local worst-case estimates are then composed together to form the final WCET bound. However at runtime, it is often impossible for all these individual worst-case execution scenarios to occur at the same time.

⊖ The hardware model must be thoroughly verified to ensure that it indeed reflects the target hardware; failing to capture inherent performance enhancing features may result in overestimations of the execution times, whereas capturing all system states in a complex machine may lead to unacceptably long analysis times. Besides, building and verifying the timing model for each processor variant is expensive, time consuming, and error prone. Custom variants and different versions of processors often have subtle different timing behaviors rendering timing models either incorrect, or unavailable.

It is very important to stress at this point that static analysis techniques have been designed primarily to analyse simple software codes meant to run on simple and predictable hardware architectures. These targeted codes are typically implemented by using high-level programming languages and by obeying strict and specific coding rules to reduce the likelihood of programmer error. This is one of the reasons why, although those methods are very efficient for well-structured safety-critical systems built on predictable and highly controlled architectures, we came to the conclusion that they are not appropriate for the type of applications and platforms considered in P-SOCRATES. We will further detail this point in Section V.

The modelling framework adopted by static analysis lends itself to formal proofs which help in establishing whether the obtained results are safe. Today, there are several static WCET tools that are commercially available, including aiT [14] and Bound-T [15]. There also exist several research prototypes, including Chronos [16], developed at National University of Singapore, Heptane [17], developed at the French National Institute for Research in Computer Science and Control (INRIA) IRISA in France, SWEET [18], developed at Malardalen Real-Time Research Center (MRTC) in Sweden, and OTAWA [19] from IRIT in France.

B. Measurement-based WCET analysis techniques

The traditional and most common method in industry to determine program timing is by measurements. The basic principle of this method follows the mantra that "the processor is the best hardware model". The program is executed many times on the actual hardware, with different inputs and in isolation, and the execution time is measured for each run by instrumenting the source code at different points [20]. Each measurement run exercises one execution path throughout the program, and thus for a same set of input values several thousands of program runs must be carried out to capture variations in execution time due to the fluctuation in system states. For those measurement-based approaches, the main challenge is essentially to identify the set of input arguments of the application that leads to its WCET.

Pros & Cons

⊕ Measurements are often immediately at the disposal of the programmer, and are useful mainly when the average case timing behavior or an approximate WCET value is of interest.

⊕ Most types of measurements have the advantage of being performed on the actual hardware, which avoids the need to construct a hardware model and hence reduce the tremendous effort and cost of deriving the estimates.

⛔ Measurements require that the hardware is available, which might not be the case for systems for which the hardware is developed in parallel with the software.

⛔ It may be problematic to set up an environment which acts like the final system.

⛔ The integrity of the actual code to be deployed in the target hardware is somehow depleted by the addition of the intrusive instrumentation code to measure the time, i.e., the measurements themselves add to the execution time of the analyzed program. This problem can be reduced, e.g., by using hardware measurement tools with no or very small intrusiveness, or by simply letting the added measurement code (and thus the extra execution time) remain in the final program. When doing measurements, possible disturbances, e.g., interrupts, also have to be identified and compensated for.

⛔ For most programs, the number of possible execution paths is too large to do exhaustive testing and therefore, measurements are carried out only for a subset of the possible input values, e.g., by giving potential "nasty" inputs which are likely to provoke the WCET, based on some manual inspection of the code. Unfortunately, the measured times will in many cases underestimate the WCET, especially when complex software and/or hardware are being analyzed. To compensate for this, it is common to add a safety margin to the worst-case measured timing, in the hope that the actual WCET lies below the resulting augmented WCET estimate. The main issue is whether the extra safety margin provably provides a safe bound, since it is based on some informed estimates. A very high margin will result in resource over-dimensioning, leading to very low utilization whereas a small margin could lead to unsafe/under-estimated WCET bounds.

C. Hybrid WCET techniques

Hybrid approaches, as the name implies, combines the merits of static and measurement-based analysis techniques. First, they borrow the flow analysis phase from static methods to construct a control flow-graph of the given program and identify a set of feasible and potentially worst execution paths (w.r.t. the execution time). Next, unlike static methods that use mathematical models of the hardware components, hybrid tools borrow their second phase from measurement-based techniques and determine the execution time of the basic blocks of those paths by executing them on the target platform or in cycle-accurate simulators. To do so, the source code of the application is instrumented with expressions (instrumentation points) that indicate that a specific section of code has been executed. These instrumentation points are typically placed along the paths identified in the first phase as potentially leading to a WCET. The application is then executed on the target hardware platform or on the simulator to collect execution traces. These traces are a sequence of time-stamped values that show which parts of the application has been executed and for how much time. Hybrid tools then produce performance metrics for each part of the executed code and, by using the performance data and knowledge of the code structure, they estimate the worst-case execution time of the program.

Pros & Cons

➕ Hybrid approaches do not rely on complex abstract models of the hardware architecture.

➕ They generally provide safe WCET estimates (i.e. higher than the actual WCET) and those are tighter than the estimates returned by static approaches (i.e. closer to the actual WCET).

⛔ The uncertainty of covering the worst-case behavior by the measurement remains since it cannot be guaranteed that the maximum interference and the worst-case execution scenario has been experienced when collecting the traces during the second phase.

⛔ It is required to instrument the application source code, which may potentially pose the same issue of intrusiveness as in measurement-based approaches.

Examples of hybrid WCET analysis tools include RapiTime [21] and MTime, a research prototype from the Real-Time Systems Group at Vienna University of Technology.

D. Probabilistic techniques

With the current hardware designs, the execution time of a given application depends on the states of the hardware components, and those states depend in turn on what has been executed previously. A classic example of such a tight relationship between the application and the underlying hardware architecture is the execution time discrepancy that can be observed when a program executes on a processor equipped with a cache subsystem. During the first execution of the program, every request to fetch instructions and data results in a cache miss and must be loaded from the main memory. At the second execution, most of these information are already in the cache and need not to be reloaded from the memory, which results in an execution time considerably shorter than during the first run.

Because of this dependence to past events, the set of measured execution times of a same program cannot be seen as an i.i.d. (independent and identically distributed) random variable and most statistical tools cannot be applied to analyse the collected execution traces. The objective of probabilistic techniques is to break this dependence to past events, so that one can sample the execution behavior of an application and then derive probabilistic estimates from that sample that are not biased (or with an upper-bounded maximum bias) when applied to the overall population, i.e. to the execution behavior of the application *under all circumstances and in all situations*.

To achieve this ambitious goal, researchers are nowadays working on modifying the hardware components and their arbitration policies to make them behave in a stochastic manner, without losing too much of their performance. For example, by replacing the traditional LRU or pseudo-LRU cache replacement policy for a policy that randomly chooses the cache line to be evicted (and assume that every cache line has the same probability to get evicted), the time overhead due to cache penalties and cache line evictions can be analysed as an i.i.d. random variable with a known distribution (see [22], [23], [24] for examples of such techniques). If every source of interference exhibits a randomized behavior with a known distribution then the execution time itself can be analysed statically.

Probabilistic approaches come in two flavors: static and measurement-based. Both of them are deeply rooted in the concept of ETP (Execution Time Profile) that represent the

probabilistic timing behavior of an execution component[2]. An ETP is a pair of vector: a timing vector that enumerates all possible times that the execution of that component may take, and a probability vector that lists the probability of occurrence for each execution time. Static probabilistic approaches derive the ETPs of the individual execution components analytically, from a mathematical model of the system. Individual ETPs are then combined by convolutions to derive the timing behavior of higher-level components, and this is done all the way up till the ETP of the overall application is computed. In contrast, measurement-based probabilistic techniques define the timing and probability vectors of an ETP (typically the ETPs are defined at the instruction level) by collecting runtime measurements. Then, the current trend is to apply results from the extreme value theory (EVT) framework to these ETPs [26], [27]. In a nutshell these EVT-based solutions organize the sample into multiple groups/intervals, analyse the distribution of the local maxima within these intervals and then estimate how far the execution time may deviate from the average of that "distribution of the extremes". Although recent, probabilistic techniques have been the object of tremendous research efforts in the last few years. Most of the breakthroughs in that discipline have been made in the scope of the European projects PROARTIS [28] and PROXIMA [29].

Pros & Cons

⊕ Probabilistic techniques provide safe and potentially tighter WCET estimates than static and hybrid techniques.

⊕ They provide information not only on the WCET of a program but on the complete spectrum of the distribution of its execution time.

⊖ Require to modify the hardware to ensure that the components exhibit a stochastic behavior.

⊖ As the i.i.d. requirement is hardly verified in currently available platforms (especially COTS platforms), the applicability of probabilistic techniques is limited as briefly discussed in [30].

III. Overview of the P-SOCRATES application model

In the P-SOCRATES project, the *application* comprises all the software parts of the systems that operate at the user-level and that have been explicitly defined by the user. The application is the software implementation (i.e., the code) of the functionality that the system must deliver to the end-user. It is organized as a collection of *real-time tasks*.

A *real-time (RT) task* is a recurrent activity that is a part of the overall system functionality to be delivered to the end-user. Every RT task is implemented and rendered parallelizable using OpenMP 4.0, the de facto standard parallel programming model used in shared memory-based architectures such as the Kalray MPPA. OpenMP version 4.0 has evolved from previous versions to consider very sophisticated types of dynamic, fine-grained and irregular parallelism.

A RT task is characterized by a software procedure that must carry out a specific operation such as processing data, computing a specific value, sampling a sensor, etc. It is also characterized by a few (user-defined or computed) parameters related to its timing behavior such as its worst-case execution time, the frequency of its activation (aka period), the time frame in which it must complete (aka its deadline), etc. Every RT task comprises a collection of *task regions* whose inter-dependencies are captured and modeled by a graph called the extended task dependency graph (eTDG). We refer the reader to [31] for a description of how to extract the TDG from an OpenMP program.

A *task region* is defined at run-time by the syntactic boundaries of an openMP task construct. For example:

```
#pragma omp task
{
  // The brackets identify the boundaries
  // of the task region
}
```

Since the task regions are defined in the code through the openMP task constructs, we will henceforth refer to them as *openMP tasks*.

An *openMP task part* (or simply, a *task part*) is a non-preemptible (at least from the OpenMP view of the world) portion of an openMP task. Specifically, consecutive task scheduling points (TSP) such as the beginning/end of a task construct, the synchronization directives, etc., identify the boundaries of an openMP task part. In the plain OpenMP task scheduler a running openMP task can be suspended at each TSP (not between any two TSPs), and the thread previously running that openMP task can be re-scheduled to a different openMP task (subject to the task scheduling constraints).

In practice, very small programs (RT tasks) can potentially create at runtime a very high number of tasks and thus task parts that will all compete for the cores and the other shared resources of the processor. The mapping of the task parts to the OS threads and the scheduling of those threads on the cores are typically very dynamic as they are carried out in the way that balances the workload and maximizes the utilization of the resources. As a consequence, it is practically infeasible to know, at design time, which task parts will execute concurrently, which further complicates the interference analysis.

IV. Which timing analysis method could be used in P-SOCRATES?

To re-iterate, the ultimate goal of P-SOCRATES is to enable the use of practices coming from the high-performance computing domain and apply them in a time predictable way considering COTS many-core architectures. It is important to note at this point something very important that is never denied but often overlooked. WCET and interference analysis in multi-cores is still a very active topic today. Despite all the work and results in multi-core scheduling theory that have been obtained in the last decade, there is currently no industrially-accepted, sound and verified results whatsoever (academic works aside) that is able to compute tight and safe estimates of the WCET of program executed in a multi-core environment. This is why our first step in P-SOCRATES was an exploratory phase in which we endeavoured to figure out which methodology could be used in the context of the project. We investigated the current practices in WCET analysis and came to the following conclusions.

[2]"An execution component represents at the finest granularity, an access to a resource and at the highest, the entire program. In between, we can find instructions (which may access several resources), basic blocks, functions, etc." (definition taken from [25])

For highly complex platforms and execution model as that of P-SOCRATES (see [11] for the details on the P-SOCRATES software stack), it is practically infeasible to derive tight WCET estimates by using *static timing analysis* techniques. This is because the complexity of the P-SOCRATES execution environment combined to the intrinsic complexity of typical COTS hardware components, and also sometimes the non-availability of their specification details, make it difficult, if not impossible, to derive accurate models of the execution environment. Static timing analysis tools are designed primarily for applications safety-critical embedded executed sequentially. Those systems generally provide a very time-predictable and "inflexible" environment in which every mapping and scheduling decision is statically taken at design-time and is then final. Unlike those systems, the P-SOCRATES software stack offers a much more complex and dynamic runtime environment composed of multiple conceptual layers: the code of the real-time tasks is executed in parallel by being fractioned into OpenMP tasks, those tasks are mapped to clusters, then to threads inside the clusters, and then these threads are scheduled dynamically on the cores. The dynamicity of the processor resource usage ensures a decent application throughput (by maximizing the utilization of the available computing resources) but it naturally impacts adversely on its time-predictability. Besides, using static timing analysis techniques in P-SOCRATES would also defeat one of the main goals of the project: develop a flexible and generic framework which can be "easily" ported to other platforms.

Traditional hybrid approaches are also not applicable as the complexity of the software stack makes the static control flow analysis step impossible. Since the workload execute in parallel, even using static mapping approaches the total order of execution of task parts is only determined at run-time, thus it is infeasible to investigate all possible scenarios at design-time to identify the worst-case execution flow/path. It is important to re-iterate that traditional timing analysis techniques have been designed primarily to analyse simple software codes executed on simple and predictable hardware architectures, typically implemented by using high-level programming language and by obeying strict and specific coding rules to reduce the likelihood of programmer error. The P-SOCRATES framework clearly targets much more complex software applications that exhibit a high degree of flexibility and dynamicity in their execution.

Probabilistic approaches are also not applicable since typical COTS components and architectures (like the ones studied in P-SOCRATES) do not provide the required features that confer a stochastic runtime behavior on the application.

By proceeding by elimination, the only remaining candidates are the "pure" measurement-based approaches. Our proposed methodology relies solely on timing-related data collected at runtime, by running the application on the target hardware. This way, we avoid both the burden of modelling the various hardware components (which takes considerable effort and time) as in static timing analysis tools and the pitfalls and pessimism associated to the over-approximations resulting from the confidentiality, and thus the non-availability, of specific information related to the internal configuration of the components. In addition, the fact that our approach is not tied to specific hardware infrastructures and application designs makes it benefit from a higher flexibility and portability than static timing analysis methods and it considerably reduces the time-to-model and time-to-result. Before we discuss the specifics of our method, we shall briefly discuss how we propose to overcome or at least mitigate the negative aspects inherent to measurement-based techniques. Those downsides listed in the previous section are repeated below.

⛔ Measurements require that hardware is available, which might not be the case for systems for which the hardware is developed in parallel with the software.

✔ Several research partners of P-SOCRATES have acquired an MPPA-256 board. Besides, a representative of Kalray is member of the IAB (International Advisory Board), which facilitates the communication with the company when assistance is needed (without mentioning the responsive and high-quality hotline service of the company).

⛔ It may be problematic to set up an environment which acts like the final system. It is common in practice that the whole application software is not available during the validation and testing of each software component.

✔ To solve this severe issue, recent works such as [32] propose to capture the resource[s] usage of each component (called the "signature" of the component) and implement specific programs that reproduce that signature. These specialized programs are then used during the testing phase of the program under analysis, to simulate the execution environment of the final system by reproducing a similar interference/resource usage pattern from the other applications that will eventually run concurrently. We envisage to use such or similar techniques during the validation phase of the project.

⛔ The integrity of the actual code to be deployed in the target hardware is somehow depleted by the addition of the intrusive instrumentation code to measure the time, i.e., the measurements themselves add to the execution time of the analyzed program. This problem can be reduced, e.g., by using hardware measurement tools with no or very small intrusiveness, or by simply letting the added measurement code (and thus the extra execution time) remain in the final program. When doing measurements, possible disturbances, e.g., interrupts, also have to be identified and compensated for.

✔ The MPPA 256 platform considered in our testing phasing provides a lightweight and non-intrusive trace system that enables to collect execution traces in predefined time bounds. By using this system we are able to collect meaningful traces of execution without generating too much disturbances in the regular timing behavior of the analyzed application.

⛔ For most programs, the number of possible execution paths is too large to do exhaustive testing and therefore, measurements are carried out only for a subset of the possible input values, e.g., by giving potential "nasty" inputs which are likely to provoke the WCET, based on some manual inspection of the code. Unfortunately, the measured times will in many cases underestimate the WCET, especially when complex software and/or hardware are being analyzed. To compensate for this, it is common to add a safety margin to the worst-case measured timing, in the hope that the actual WCET lies below the resulting WCET estimate. The main issue is whether the extra safety margin provably provides a safe bound, since it is based on some informed estimates. A very high margin will result in resource over-dimensioning, leading to very low utilization and while a small margin could lead to an unsafe system.

✔ Traditionally, the safety margin applied to the WCET bounds is an educated estimation of the interference (from the system or from other applications) that has not been observed during the testing phase but that the analyzed application could potentially incur at runtime. For single-core systems, this estimation is usually built on past experience. For example, in the IEC 61508 standard related to functional safety of electrical/electronic/programmable electronic safety-related systems, in order to ensure that the working capacity of the system is sufficient to meet the specified requirements it is said that:

"For simple systems an analytic solution may be sufficient, while for more complex systems some form of simulation may be more appropriate to obtain accurate results. Before detailed modelling, a simpler 'resource budget' check can be used which sums the resources requirements of all the processes. If the requirements exceed designed system capacity, the design is infeasible. Even if the design passes this check, performance modelling may show that excessive delays and response times occur due to resource starvation. To avoid this situation, engineers often design systems to use some fraction (for example 50%) of the total resources so that the probability of resource starvation is reduced."

IEC 61508 [33], 2nd Edition, Part 7. Requirement C.5.20 (Performance modelling), page 99.

Unfortunately, on multicore and manycore architectures there is currently no past experience to rely upon for experts to estimate safe margins. Hence we have to build a new body of knowledge and investigate novel approaches to produce trustworthy timing estimates, and we must motivate these approaches and justify why we believe in their reliability.

Our first move towards this ambitious goal is to reconsider the very concept of a safety margin. As explained above, it is a common practice in single-core systems to simply add a margin of 50% (or any other percentage depending on the design preferences/constraints and his confidence in those margins) to the maximum execution time observed. With this approach only two undesired scenarios may happen:

1) The simulation process failed to identify the set of input arguments that takes the longest execution path throughout the program and leads to its WCET.

2) The simulation process found the path[s] leading to the WCET but it did not generate the maximal possible interference while exercising those paths.

In the former case the percentage applied as a safety margin is somewhat meaningless, because the longest execution path[s] that have been undetected may a priori take an arbitrarily long time to finish, whereas in the latter case the actual WCET has not been observed only because the interference patterns generated during the simulations did not put the application into the worst execution conditions. Very little can be done to handle the first case except from optimizing the worst-input-arguments-finder algorithm. On the other hand, to avoid the second case, it is relevant to design a method capable of extracting and isolating the part of the execution time that is due to the interference with the other applications or with the system itself, from the part of the execution time that is intrinsic to the execution path taken. Doing so would allow us to apply a safety margin not on the overall execution time as done in single-core systems, but only on the time-overhead caused by the interference endured by the analyzed application at runtime. This is the main objective of our envisioned approach.

V. The P-SOCRATES envisioned methodology and its innovative aspect

Our method executes many times the RT task to be analysed on the actual hardware, over multiple input sets and in different execution conditions for each input. The source code of the RT task is instrumented at different points. Specifically, a "trace-point" is added at the beginning and at the end of every OpenMP task part, which enables to record and compute the time taken to execute the body of every task part at runtime. Obtaining those traces was our first challenge for which we developed a software tool that (1) instruments the source code with appropriate trace-points (this part is still under development), (2) uploads the instrumented code on the MPPA-256 (the test-platform used in the project), (3) prepares and sets up the runtime environment to activate the trace system, (4) re-compiles the application source code, (5) runs the RT task, (6) collects, decodes, and processes the traces, and finally (7) returns the traces in an appropriate and readable format.

Typically, feeding the RT task to be analysed with different sets of input arguments modifies its runtime behavior: some task parts execute for longer, others execute for less time, some do or do not execute at all, etc. Even the order and the way the task parts are mapped to the OS threads is affected as distinct input sets are given to the analysed task because the overall execution time of the task parts ends up being very different from one input set to another. That is, the execution time of the whole RT task is extremely sensitive to the context in which it is executed.

Each measurement run exercises only one execution path throughout the RT task, and thus for a same set of input values several hundreds or even thousands of runs must be carried out to capture variations in the execution time due to the fluctuation in system states. Assuming that the collection of input values contains those that lead to the WCET, it is now a matter of separating from the total execution time of each measurement run the fraction of time due to the interference on the shared resources. After separating these two components, i.e. the intrinsic execution time of the task part and the overhead caused by the interference, we envisage to model this overhead through kernel smoothing techniques.

A kernel smoother is a statistical technique for estimating a real valued function by using its noisy observations, when no parametric model for this function is known. In short, for each set of input values we will run the application thousands of times before applying a kernel smoother on the thousands of measured execution times obtained for every task parts that have been exercised, and specifically on the time-overhead due to the interference. This way we will estimate the function that characterizes the time-penalty assigned to every task part for sharing the processor resources. The estimated function is smooth and its level of smoothness is set by a single parameter called the *bandwidth*. Traditional statistical techniques (based on the MSE for example) can then be used to evaluate the goodness of fit of the kernel. The higher the bandwidth, the more over-approximate the data fitting, and vice-versa. In the context of P-SOCRATES, for each set of input values we will use the bandwidth parameter as a mean to set up the desired safety margin within the model of the time-overhead

due to the interference. Then we will use some "goodness-of-fit" evaluation techniques to evaluate the degree of over-approximation resulting from the chosen bandwidths.

VI. Conclusion

In a nutshell the paper discussed the application of the currently-available WCET analysis techniques and tools within the context of the P-SOCRATES project. After discussing the pros and cons of each methodology, we introduced the approach that we intend to research upon for computing WCET estimates in the project, considering many-cores architectures and highly parallelized applications.

Acknowledgements

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and when applicable, co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); and also by the European Union under the Seventh Framework Programme (FP7/2007-2013), grant agreement n 611016 (P-SOCRATES).

References

[1] S. Girbal, M. Moretó, A. Grasset, J. Abella, E. Quiñones, F. Cazorla, and S. Yehia, "The next convergence: High-performance and mission-critical markets," in *1st Workshop on High-performance and Real-time Embedded Systems (HiRES)*, 2013.

[2] L. Pinho, E. Quiñones, M. Bertogna, A. Marongiu, J. Pereira-Carlos, C. Scordino, and M. Ramponi, "P-socrates: A parallel software framework for time-critical many-core systems," in *Proceedings of the 17th Euromicro Conference on Digital System Design (DSD)*, 2014.

[3] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: A many-core x86 architecture for visual computing," in *ACM SIGGRAPH 2008 papers*, vol. 27(3), 2008, pp. 1–15.

[4] H. Wong, A. Bracy, E. Schuchman, T. Aamodt, J. Collins, P. Wang, G. Chinya, A. Khandelwal-Groen, H. Jiang, and H. Wang, "Pangaea: A tightly-coupled IA32 heterogeneous chip multiprocessor," in *Proceedings of the 17th ACM International Conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 52–61.

[5] *Intel Many Integrated Core Architecture - Advanced*, Intel Corporation, last access 8 April 2015, **[online]** http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html.

[6] *Intel Xeon Phi Product Family*, Intel Corporation, last access 8 April 2015, **[online]** http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html.

[7] L. Benini, E. Flamand, D. Fuin, and D. 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*, 2012, pp. 983–987.

[8] *Kalray*, Kalray Corporation, last access 8 April 2015, **[online]** http://www.kalrayinc.com.

[9] L. Pinho, E. Quiñones, M. Bertogna, A. Marongiu, J. Pereira-Carlos, C. Scordino, and M. Ramponi, "Time criticality challenge in the presence of parallelised execution," in *2nd Workshop on High-performance and Real-time Embedded Systems (HiRES)*, 2014.

[10] D. Luckham, "The power of events: An introduction to complex event processing in distributed enterprise systems," in *Addison-Wesley Longman Publishing Co. Inc.*, 2001.

[11] V. Nélis, P. M. Yomsi, L. M. Pinho, E. Quinones, M. Bertogna, A. Marongiu, P. Gai, and C. Scordino, "A system model and stack for the parallelization of time-critical applications on many-core architectures," in *3rd Workshop on High-performance and Real-time Embedded Systems*, 2015.

[12] A. Ermedahl and J. Engblom, "Execution time analysis for embedded real-time systems," in *Ed. Insup Lee, Jospeh Y-T. Leung, Sang H. Son*, vol. Chapman and Hall-CRC - Taylor and Francis Group, 2007.

[13] P. Lokuciejewski and P. Marwedel, "Worst-case execution time aware compilation techniques for real-time systems - summary and future work," in *Springer Netherlands*, 2011, pp. 229–234.

[14] *How Does WCET Analysis with aiT Work?*, AbsInt GmbH, last access 8 April 2015, **[online]** http://www.absint.com/ait/analysis.htm.

[15] *Bound-T time and stack analyser*, Tidorum Ltd, last access 8 April 2015, **[online]** http://www.bound-t.com/.

[16] *Chronos*, NUS, last access 8 April 2015, **[online]** http://www.comp.nus.edu.sg/ rpembed/chronos/.

[17] *Heptane static WCET estimation tool*, IRISA, last access 8 April 2015, **[online]** https://team.inria.fr/alf/software/heptane/.

[18] *SWEET*, MRTC, last access 8 April 2015, **[online]** http://www.mrtc.mdh.se/projects/wcet/sweet/DocBook/out/webhelp/index_frames.html.

[19] *Computing a WCET - Otawa*, IRIT, last access 8 April 2015, **[online]** http://www.irit.fr/recherches/ARCHI/MARCH/OTAWA/doku.php?id=doc:computing_a_wcet.

[20] R. Kirner, P. Puschner, and I. Wenzel, "Measurement-based worst-case execution time analysis using automatic test-data generation," in *4th Euromicro International Workshop on WCET Analysis*, 2004, pp. 67–70.

[21] *How does RapiTime work?*, Rapita Systems LTD, last access 8 April 2015, **[online]** http://www.rapitasystems.com/products/rapitime/how-does-rapitime-work.

[22] D. Griffin, B. Lesage, A. Burns, and R. I. Davis, "Static probabilistic timing analysis of random replacement caches using lossy compression," in *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, 2014, p. 289:298.

[23] R. I. D. Sebastian Altmeyer, Liliana Cucu-Grosjean, "Static probabilistic timing analysis for real-time systems using random replacement caches," *Real-Time Systems*, vol. 51, no. 1, pp. 77–123, 2013.

[24] E. Mezzetti, M. Ziccardi, T. Vardanega, J. Abella, E. Quiones, and F. J. Cazorla, "Randomized caches can be pretty useful to hard real-time systems," *Leibniz Transactions on Embedded Systems (LITES)*, vol. 2, no. 1, pp. 77–123, 2015.

[25] J. Abella, D. Hardy, I. Puaut, E. Quinones, and F. Cazorla, "On the comparison of deterministic and probabilistic wcet estimation techniques," in *26th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2014, pp. 266–275.

[26] L. Carnevali, A. Melani, L. Santinelli, and G. Lipari, "Probabilistic deadline miss analysis of real-time systems using regenerative transient analysis," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, 2014, pp. 299–308.

[27] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart, "On the sustainability of the extreme value theory for WCET estimation," in *14th International Workshop on Worst-Case Execution Time Analysis*, 2014, pp. 21–30.

[28] *Proartis: Probabilistically Analysable Real-Time Systems*, Proartis, last access 8 April 2015, **[online]** http://www.proartis-project.eu/.

[29] *Probabilistic real-time control of mixed-criticality multicore and manycore systems*, PROXIMA, last access 8 April 2015, **[online]** http://www.proxima-project.eu/.

[30] D. Griffin and A. Burns, "Realism in Statistical Analysis of Worst Case Execution Times," in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, 2010, pp. 44–53.

[31] R. Vargas, E. Quinones, and A. Marongiu, "Openmp and timing predictability: A possible union?" in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, 2015, pp. 617–620.

[32] G. Fernandez, J. Jalle, J. Abella, E. Q. nones, T. Vardanega, and F. J. Cazorla, "Resource usage templates and signatures for COTS multicore processors," in *52nd Design Automation Conference (DAC)*, 2015.

[33] The International Electrotechnical Commission, *IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2010.