Proceedings of the 4th International Real-Time Scheduling Open Problems Seminar

Edited by Liliana Cucu-Grosjean and Sathish Gopalakrishnan

Foreword

We would like to extend a warm welcome to you all to the 4th Real-Time Scheduling Open Problems Seminar (RTSOPS 2013). The goal of this meeting is to continue the tradition of providing a venue for the exchange of ideas and discussion of interesting unsolved problems in real-time scheduling. This seminar is intended as an enabler for collaborative and cooperative work within the real-time systems research community.

The day-long seminar is structured around a set of presentation sessions, each followed by sufficient time for collaborative work. During the collaborative sessions, we invite all participants to interact in groups to explore initial steps towards solving some of the presented problems and, sometimes, refining the problem definition and scope to make the problems more germane.

RTSOPS 2013 features 10 open problems and one invited talk. The invited talk is the novelty of this year edition. Alix Munier from the operations research community will present us techniques used to solve general cyclic scheduling problems. These problems are close to those of real-time scheduling. The abstract of this talk is also included the proceedings. The proceedings are available as INRIA research report.

Another novelty of this year is that we followed a new approach towards paper submissions. We did not enforce a deadline and reviewed submissions until we accepted 10 problems for discussion.

We would like to thank the program committee for their generous time contribution and efforts in providing feedback to the authors. The program committee for RTSOPS 2013 included:

- Marko Bertogna, University of Modena (Italy)
- Vincent Nelis, CISTER/IPP-Hurray (Portugal)
- Sebastian Altmeyer, University of Amsterdam (Nederland)

We are also grateful to the advice and support, and reviewing effort provided by the RTSOPS steering committee members:

- Rob Davis, University of York (UK)
- Nathan Fisher, Wayne University (USA)
- Gerhard Fohler TU Kaiserslautern (Germany)

A further "thank you" is extended to ECRTS '13 general chair Laurent George for his assistance in helping us organize this seminar. We thank Dorin Maxim with helping having in time the printed version of these proceedings.

We look forward to a productive RTSOPS 2013,
Liliana Cucu-Grosjean and Sathish Gopalakrishnan

Table of Contents

Invited talk – Alix Munier ……. *i*

Accepted papers

Towards a shared-data-aware multicore real-time schedule
Giovani Gracioli and Antônio Augusto Frohlich ……. *1*

Static probabilistic timing analysis for multicore processors with shared cache
Rob Davis, Jack Whitham and Dorin Maxim ……. *3*

Energy-aware scehduling for tasks with mixed energy requirements
M. Babagini, G. Butazzo and M. Bertogna ……. 6

Is Audsley's scheme the most expressive optimal priority assignment algorithm ?
A. Burns ……. *8*

Global laxity-based scheduling on multiprocessor resource reservations
J. Craveiro and J. Rufino ……. *12*

Uniform multiprocessor periodic resource model
J. Craveiro and J. Rufino ……. *14*

Exploiting Period Compatibility for Partitioned Rate-monotonic Scheduling using Circular Statistics
D. Müller and M. Werner ……. *16*

Reality Check: the Need for Benchmarking in RTS and CPS
M. Di Natale, C. Dong, H. Zeng ……. *18*

Schedulability Analysis with an Extended Timing Definition Language
T. Kloda, B. d'Ausbourg and L. Santinelli ……. *20*

Improvements to Static Probabilistic Timing Analysis for Systems with Random Cache Replacement Policies
R. Davis ……. *22*

INVITED TALK

Alix Munier (Professor, University of Marie et Pierre Currie, Paris, France)

- Liveness Evaluation for Synchronous and Cyclo-Static DataFlow Graphs

(Join work with Mohamed Benazouz, Bruno Bodin and Thomas Hujsa)

Synchronous DataFlow Graphs and Cyclo-Static DataFlow Graphs (respectiveley SDF and CSDFG in short) are formalisms commonly used to model parallel applications composed by actors communicating through buffers. The liveness of a SDF/CSDFG ensures that all actors can be executed infinitely often. This property is clearly fundamental for the design of embedded applications.

This talks aims to present an original method to compute sufficient conditions for the liveness of a SCD/CSDF. It is based on two simple polynomial transformations of the initial problem (namely normalization and useful tokens). Two algorithms of polynomial-time for checking the liveness are then derived for CSDF and compared to a symbolic execution of the graph. The performance of our methods are comparable to those existing in the literature for industrial applications. However, they are far more effective on randomly generated instances, ensuring their scalability for future more complex applications and their possible implementation in a compiler.

Towards a shared-data-aware multicore real-time scheduler

Giovani Gracioli and Antônio Augusto Fröhlich
Federal University of Santa Catarina, Florianópolis, Brazil
{giovani,guto}@lisha.ufsc.br

I. The Problem

Multicore processors have been established in the multicore embedded real-time system domain. Several applications that used to be implemented in dedicated hardware, such as digital signal processing algorithms and baseband processing in wireless communication, can now be ported to software with similar performance and more support flexibility for developers (e.g., bug fixes, online updating, maintainability) [1]. Such applications process a considerable amount of data under real-time conditions and are composed of several real-time cooperating threads (threads that share data and run concurrently in the available cores). In this scenario, due to multicore processor organization, we must consider some important characteristics, specifically, the memory hierarchy [2]. The memory hierarchy holds an important role, because it affects the estimation of the Worst-Case Execution Time (WCET), which is extremely important in the sense of guaranteeing that all threads will meet their deadlines through the design phase evaluation (schedulability analysis) [3], [4].

Several works have been proposed to deal with memory organization in multicore architectures and provide real-time guarantees [1], [5]. However, they only consider scenarios in which threads are independent (without data sharing). In this case, the influence among threads (contention for cache space and interference in the cache lines) can be solved by memory partitioning/locking mechanisms provided by a special hardware [4] or implemented into the real-time scheduler [2], [6].

The problem we are addressing in this paper rises from the memory hierarchy present in today's SMP architectures and their memory coherence protocols (e.g., MESI, MOESI, or MESIF). Each core has its own data and uses its private data cache for speeding up processing. However, when cores share data, each copy of the data is placed in the core's private cache and the cache-coherence protocol is responsible for keeping the consistency among copies (through bus snooping). When a core writes into a data that other cores have cached, the cache-coherence protocol invalidates all copies, causing an implicit delay in the application's execution time. In the same way, when a core reads a shared data that was just written by another core, the cache-coherence protocol does not return the data until it finds the cache that has the data, annotates that cache line to indicate that it is shared, and recovers the data to the reading core. These operations are performed automatically by the memory controller hardware and take hundreds of cycles (about the same time as accessing the off-chip RAM), increasing the application's execution time which may lead to deadline losses [7]. Two kinds of scaling problems occur due to shared memory contention [7]: access serialization to the same cache line done by the cache coherence protocol, which prevents parallel speedup, and saturation into the inter-core interconnection, also preventing parallel processing gains.

II. Open Questions

Based on the problem description, we have identified the following open questions:

- *Task model.* Traditional real-time task models do not take into account the parallelism and data dependencies presented in multicore processors. The parallel synchronous [8] and multi-phase multi-thread [9] task models are important steps toward formally defining a task model for parallel real-time tasks. However, the task model presents restrictions and supports only DM and EDF scheduling policies. Moreover, is the parallel synchronous task model adequate for the described problem? We believe that there is a need for better parallel task models.
- *Cache contention.* Are there techniques to reduce the contention for cache spaces and decrease the influence of the cache coherency protocols in applications that share data without relying on hardware redesign? Cache partitioning/locking techniques may be the answer for eliminating the contention for cache spaces. Partitioning/Locking isolates application workloads that interfere with each other, increasing the predictability. Nevertheless, when there is data sharing between parallel threads, cache partitioning does not solve the problem, because threads will access the same data location on the memory hierarchy. Moreover, what is the influence of partitioning on global, partitioned, and semi-partitioned real-time schedulers? There is still room for combining cache partitioning with other techniques to provide real-time guarantees for data sharing application.
- *Real-time scheduling.* To the best of our knowledge, there is no shared-cache-aware real-time scheduler proposed in the literature. We believe that a shared-cache-aware real-time scheduler may benefit from static information collected by offline analyzers and run-time information collected by the hardware support, such as hardware performance counters (HPCs). However, which static information must be collected and which hardware events are more suitable for the scheduler are

still open questions. Moreover, which techniques could the scheduler use to decrease the contention for shared data? For example, the scheduler can prevent a thread from being run to avoid the contention, but how about the real-time guarantees? How much time can a thread wait? Also, the schedulability tests must not be too pessimistic, such as the very pessimistic Global-EDF sufficient tests.

III. Envisioned Solution Statement

Towards a solution for the problem, we envision a set of OS techniques designed to provide predictability and real-time guarantees for embedded multicore real-time applications. The proposed architecture is depicted in Figure 1 and is a step forward to mitigate the effects of contention for shared cache memory. It is composed of a two-level OS cache partitioning to minimize the contention for cache space between different applications (inter-application partitioning) and between threads of the same application (intra-application partitioning). It also has a shared memory-aware scheduler responsible for detecting and minimizing the influence of memory coherence from threads that share data (e.g., access serialization to the same cache line and saturation in the inter-core interconnection), while still compromising with real-time guarantees.

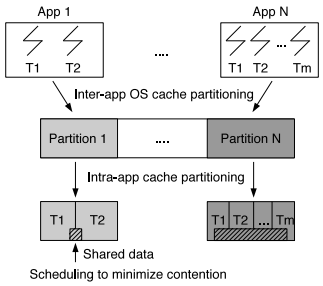

Figure 1. Proposed envisioned solution architecture.

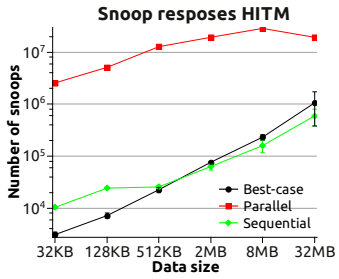

Figure 2. HPCs evaluation on Intel Q9550 processor: snoops HITM.

We are currently investigating which hardware events can be used together with the scheduler and which techniques the scheduler could use to mitigate the cache coherency effects. For example, in an experiment carried out using the Intel Q9550, we measured the number of snoop requests in a synthetic benchmark. Figure 2 shows the obtained results. We can note that the parallel application (in which there is data sharing) presents up to three orders of magnitude more snoop requests than the other two applications that do not share data. This experiment shows the feasibility of HPCs: they could correctly feed the scheduler with run-time information. The RTOS can use HPCs to detect sharing patterns among threads and take a scheduling decision, such as stopping a thread for a short period by sending an Inter-Processor Interrupt (IPI). We are also improving the multicore real-time support on the EPOS RTOS [10].

IV. Summary

In this position paper we presented the problem of scheduling real-time threads considering data sharing and the implicit delay caused by the cache coherency protocols implemented on today's SMP processors. Our proposal towards a solution is a combination of OS techniques to alleviate the effects of the shared cache coherency. As future work, we intend to investigate techniques to be incorporated into the OS real-time scheduling and evaluate our proposed solution. Moreover, there are still relevant open questions, as described in Section II. This work was partially supported by the Coordination for Improvement of Higher Level Personnel (CAPES) grant, projects RH-TVD 006/2008 and 240/2008, and CAPES-DFAIT 004/2011.

References

[1] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 101–110.

[2] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing shared resource contention in multicore processors via scheduling," in *Proceedings of the 15th edition of ASPLOS on Architectural support for programming languages and operating systems*, ser. ASPLOS '10, 2010, pp. 129–142.

[3] L. Wehmeyer and P. Marwedel, "Influence of memory hierarchies on predictability for time constrained embedded software," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 600–605.

[4] V. Suhendra and T. Mitra, "Exploring locking & partitioning for predictable shared caches on multi-cores," in *DAC '08: Proceedings of the 45th annual Design Automation Conference*. New York, NY, USA: ACM, 2008, pp. 300–303.

[5] J. M. Calandrino and J. H. Anderson, "Cache-aware real-time scheduling on multicore platforms: Heuristics and a case study," in *ECRTS '08: Proceedings of the 2008 Euromicro Conference on Real-Time Systems*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 299–308.

[6] N. Guan, M. Stigge, W. Yi, and G. Yu, "Cache-aware scheduling and analysis for multicores," in *EMSOFT '09: Proceedings of the seventh ACM international conference on Embedded software*. New York, NY, USA: ACM, 2009, pp. 245–254.

[7] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, and N. Zeldovich, "An Analysis of Linux Scalability to Many Cores," in *OSDI 2010: Proceedings of the 9th USENIX conference on Operating Systems Design and Implementation*.

[8] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *2011 IEEE RTSS*, 29 2011-dec. 2 2011, pp. 217 –226.

[9] B. Bado, L. George, P. Courbin, and J. Goossens, "A semi-partitioned approach for parallel real-time scheduling," in *Proc. of the RTNS '12*. New York, NY, USA: ACM, 2012, pp. 151–160.

[10] G. Gracioli, A. A. Fröhlich, R. Pellizzoni, and S. Fischmeister, "Implementation and evaluation of global and partitioned scheduling in a real-time os," *Real-Time Systems*, 2013.

Static Probabilistic Timing Analysis for Multicore Processors with Shared Cache

Robert I. Davis, Jack Whitham,
Computer Science Department, University of York, UK
rob.davis@york.ac.uk, jack.whitham@york.ac.uk

Dorin Maxim
INRIA Nancy-Grand Est, France
dorin.maxim@inria.fr

I. Introduction

As a result of stringent requirements on size, weight, and power consumption (SWaP), as well as the need to provide advanced new functionality through software, critical real-time embedded systems in the aerospace, space, and automotive markets are beginning to make use of multicore processors for hard real-time applications. The deployment of hard real-time applications on advanced multicore processors requires a number of challenges to be overcome. A wealth of research has been done on scheduling and schedulability analysis for multicore platforms (see [7] for a survey); however, this work needs to be underpinned by safe and tight estimates of the Worst-Case Execution Time (WCET) of component tasks. Considerable work has been done on WCET analysis assuming non-pre-emptive execution on single processors (see [13] for a survey), and the impact of Cache Related Pre-emption Delays (CRPD) has been taken into account for multitasking systems [1], [2], [11]. However, research into adapting WCET techniques to address the challenges of multicore processors, and specifically the use of caches that are shared between two or more processors, is in its infancy.

Many multicore processors make use of a cache hierarchy with private L1 caches per core and an L2 cache shared between cores. Sharing the L2 cache provides a significant performance boost over private L2 caches of the same total size, improving average-case performance and energy efficiency; however, the inter-core cache interference makes the WCET analysis problem extremely challenging. The problem is exacerbated by timing anomalies: The worst-case path for code executed in isolation may no longer be the worst-case path when inter-core cache contention is accounted for – see Figure 2 in [14].

The simplest and, as far as we are aware, the only compositional solution available so far is to assume that all L2 accesses are cache misses; however, as L2 latencies are typically high this is extremely pessimistic. Initial papers in this area [10], [14], [15], and [16] provide analysis which has complex dependencies on the detailed execution behaviour of contending tasks on other cores. However, details of the contending tasks may not necessarily be available for analysis, and even if they are, then such cross dependencies go against requirements for composability, which are necessary for the efficient development and integration of complex systems.

II. Probabilistic Approach

One possible solution is to make use of randomisation techniques, in particular random cache replacement policies, which make the probability of pathological cases vanishingly small. Static Probabilistic Timing Analysis (SPTA) has been developed for single processor systems assuming both evict-on-access [4] and evict-on-miss policies [8], with analysis of probabilistic CRPDs also given in [8]. Such approaches have the potential to provide an increase in the level of performance of hard real-time systems that can be guaranteed with respect to an acceptable threshold for the timing failure rate [3], particularly as the bounds provided degrade far less rapidly, with respect to incomplete information about the execution history, than the WCET computed assuming deterministic cache replacement policies [12].

It is our conjecture that in the multicore case, with shared caches, random cache replacement policies have the potential to break dependencies on execution history and specific inter-core cache contention, and so permit effective probabilistic WCET analysis that supports timing composition. This is particularly important in the case of mixed-criticality systems where the tasks running on other processors may be of lower criticality. Separation via a simple interface, independent of task behaviour or misbehaviour, ensures that lower criticality tasks have a strictly bounded effect on higher criticality tasks of interest through interactions via the shared cache. A similar argument applies to open systems, where non-real-time tasks may be downloaded and run on a specific core. Here the complete set of tasks is not available at design time for analysis.

A. Single processor SPTA

We now recap on SPTA for single processor systems with an evict-on-miss random cache replacement policy for the instruction cache and no data cache. With the evict-on-miss policy, whenever an instruction is requested and is not found in the cache, then a randomly chosen cache line is evicted and the memory block containing the instruction is loaded into the evicted location. We assume an N-way associative cache, and hence the probability of any cache line being evicted on a miss is $1/N$.

For simplicity, we assume a single path program (extensions to multipath programs are given in [8]) comprising a fixed sequence of instructions. We represent these instructions via the memory blocks they access, with a superscript indicating the *re-use distance k.* (The re-use distance is the maximum number of evictions since the last access to the memory block containing that instruction). For example, $a, b, a^1, c, d, b^3, c^2, d^2, a^5, e, b^4, f, e^2, g, a^5, b^4, h$. For each instruction, the probability of a cache hit is lower bounded by:

$$P_{hit}(k) = \left(\frac{N-1}{N}\right)^k \quad (1)$$

Provided that $k < N$, otherwise $P^{hit}(k) = 0$, (details of this latter restriction are given in [8]). Given fixed costs for the cache-hit latency (e.g. $H = 1$) and the cache miss latency (e.g. $M = 10$), then an upper bound pWCET distribution of a program can be computed as the convolution ($\otimes$) of the probability mass functions (PMFs) of each instruction. For example, given two instructions with PMFs with cache hit probabilities of 0.8 and 0.7 respectively, we get a pWCET distribution for the 'program' that has a probability of the execution time being 2 on any given run of 0.56, a probability that it will be 11 of 0.38, and a probability that there will be two cache misses and hence an execution time of 20 of 0.06.

$$\begin{pmatrix} 1 & 10 \\ 0.8 & 0.2 \end{pmatrix} \otimes \begin{pmatrix} 1 & 10 \\ 0.7 & 0.3 \end{pmatrix} = \begin{pmatrix} 2 & 11 & 20 \\ 0.56 & 0.38 & 0.06 \end{pmatrix} \quad (2)$$

We note that for larger numbers of instructions, the probability of a large number of cache misses quickly becomes vanishingly small, as illustrated by the graphs of 1-CDFs or exceedance functions shown in Figure 1 and Figure 2 below.

B. Open problem: Multicore SPTA with shared cache

In the multiprocessor case, we would like to develop SPTA that derives the probabilities of cache hits / misses and hence the pWCET distribution for a sequence of instructions (i.e. a task) running on one core, given an instruction cache with an evict-on-miss random replacement policy that is shared with one or more other cores which also execute tasks that can cause evictions. (For clarity and simplicity in our initial investigation we consider only instruction cache and do not consider a cache hierarchy). Clearly there is a mutual dependency between tasks; however, we aim to use simple interface models to restrict the information required to analyse each task and hence support a compositional approach to system development.

For example:

(i) A simple interface might state that the tasks running on each contending core can cause at most $\lceil t / M \rceil$ evictions in a time interval of length t, where M is the cache miss latency. This interface description holds irrespective of the details of the tasks and hence supports composition.

(ii) An upper bound may also be derived on the number of evictions caused in a time interval by all cores, reflecting the limited bandwidth of the memory bus. Due to limited bandwidth, there can be at most $\lceil t / b \rceil$ evictions in a time interval of length t, where b is the time required to transfer one cache block.

(iii) More complex analysis of the number of bus requests may be possible. Thus more complex interfaces might provide sub-additive functions giving the maximum number of evictions that can possibly occur due to all cores as a function of the length of the time interval. We note; however, that modelling the maximum number of bus requests from tasks on other cores as done in [5], [6], while providing more precise analysis, goes against the requirements for composition. Related work could look at how hardware might potentially police a maximum rate of evictions due to tasks on a particular core.

To fully develop the theory in this area, a detailed model of the behaviour of the hardware is needed to correctly and accurately model the worst-case number of evictions that can occur due to contention from other cores in a time interval of length t, as well as to determine the maximum amount of time required for a sequence of instructions to execute on one core given some number of cache misses.

A first attempt at solving our open problem integrates simple interface models of additional evictions into existing SPTA.

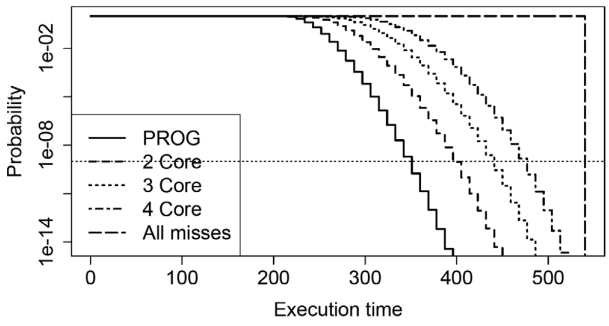

Figure 1: pWCET distributions (1-CDF) for FAC

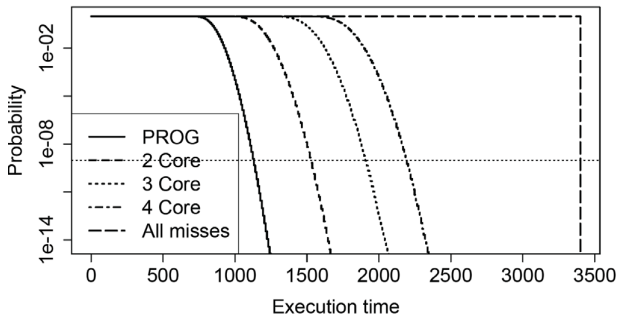

Figure 2: pWCET distributions (1-CDF) for FIBCALL

As an example, we model the contention from other cores as increasing the maximum number of evictions between instructions and hence their re-use distances. With m cores, the re-use distance k of each instruction (assuming execution in isolation) could potentially be increased by contention to $(m-1)+mk$. Figure 1 and Figure 2 show the effect that this increase in re-use distances has on the pWCET distributions for the FAC and FIBCALL programs from the Malardalen

benchmark suite [9]. PROG represents the program run in isolation, while the lines on the graphs for 2 Core, 3 Core, and 4 Core show the effect on the pWCET distribution due to contention from other cores. We observe that the increase in execution time at a given probability (shown for 10^{-9}) though significant, is much less than would have to be assumed for a deterministic system. In the deterministic case, with no knowledge of which cache blocks programs on other cores are using, the only safe assumption is the 'all misses' scenario shown as the final line in the figures. (Note, the results are for an evict-on-miss random cache replacement policy, a cache block size of 1 instruction, and a cache size of N=128 blocks).

The provision of WCETs and pWCETs form the basis for schedulability analysis. The context for the problem presented is one of higher level task allocation and scheduling. Assuming, for example, partitioned scheduling, pWCETs are needed as part of the schedulability analysis for each processor, and hence as part of the task allocation algorithm. These pWCETs could potentially vary depending on the detailed task allocation, indicating the need for an integrated approach. However, with simpler models fully supporting composition, then it would be sufficient to obtain pWCETs that account for the maximum impact of contention. Such pWCETs distributions would be independent of the task allocation and could be used directly in a separate task allocation and schedulability analysis algorithm, i.e. without integration. Further, with the development of a SPTA for shared caches, a comparison can be made to see if better pWCET estimates are obtained with a shared cache or with partitioning of the cache to individual cores.

ACKNOWLEDGEMENTS

The authors would like to thank Luca Santinelli from ONERA for his help with the example figures.

REFERENCES

[1] S. Altmeyer, R.I. Davis, C. Maiza "Cache related Pre-emption Delay aware response time analysis for fixed priority pre-emptive systems". In proceedings Real-Time Systems Symposium, pp. 261-271, 2011.

[2] S. Altmeyer, R.I. Davis, C. Maiza "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems" . Real-Time Systems, Volume 48, Issue 5, Pages 499-526, Sept 2012.

[3] F. Cazorla, E. Quinones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically analysable real time systems. *ACM Transactions on Embedded Computing Systems (to appear)*, 2013.

[4] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), 2012.

[5] D. Dasari, B. Andersson, V. Nelis, S.M. Petters, A. Easwaran, L. Jinkyu, "Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus," In Proceedings 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp.1068,1075, 16-18 Nov. 2011.

[6] D. Dasari, V. Nelis, "An Analysis of the Impact of Bus Contention on the WCET in Multicores," In Proceedings High Performance Computing and Communication & International Conference on Embedded Software and Systems (HPCC-ICESS), pp.1450,1457, 25-27 June 2012.

[7] R.I. Davis, A. Burns, "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems", ACM Computing Surveys, 43, 4, Article 35 (October 2011), 44 pages. DOI 10.1145/1978802.1978814.

[8] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. "Analysis of probabilistic cache related pre-emption delays". In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS) 2013. (Techical report available from http://www-users.cs.york.ac.uk/ robdavis/).

[9] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Malardalen WCET benchmarks – past, present and future. In B. Lisper, editor, *WCET 2010*, pages 137–147, Brussels, Belgium, July 2010. OCG.

[10] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury,"Timing analysis of concurrent programs running on shared cache multi-cores," Proceedings of the Real-Time Systems Symposium (RTSS), 2009.

[11] W. Lunniss, S. Altmeyer, C. Maiza and R.I. Davis, "Integrating Cache Related Pre-emption Delay Analysis into EDF Scheduling" In proceedings Real-Time and Embedded Technology and Applications Symposium 2013.

[12] E. Quinones, E. Berger, G. Bernat, and F. Cazorla. "Using Randomized Caches in Probabilistic Real-Time Systems". In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), pages 129–138, 2009.

[13] R.Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckman, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenstrom. TheWorst case execution time problem - overview of methods and survey of tools. In ACM Transactions on Embedded Computing Systems, January 2007.

[14] J. Yan and W. Zhang, "WCET analysis for multi-core processors with shared L2 instruction caches," Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS), 2008.

[15] W. Zhang, J. Yan "Static Timing Analysis of Shared Caches for Multicore Processors", Journal of Computing Science and Engineering, Vol. 6, No. 4, December 2012, pp. 267-278

[16] W. Zhang and J. Yan, "Accurately estimating worst-case execution time for multi-core processors with shared directmapped instruction caches," Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, (RTCSA) 2009.

Energy-Aware Scheduling for Tasks with Mixed Energy Requirements

Mario Bambagini, Giorgio Buttazzo
{mario.bambagini, giorgio.buttazzo}@sssup.it
Scuola Superiore Sant'Anna, Pisa, Italy

Marko Bertogna
marko.bertogna@unimore.it
University of Modena and Reggio Emilia, Italy

Abstract

Two widespread techniques, Dynamic Voltage/Frequency Scaling (DVFS) and Dynamic Power Management (DPM), rely on speed reduction and low-power state exploitation, respectively, to reduce energy consumption in modern computing platforms. Traditionally, dynamic power was the main source of processor power consumption, therefore DVFS was more effective than DPM to reduce energy consumption in processing elements, whereas DPM was mainly used to shut down system components not currently in use. As the technology continues to miniaturize processors, leakage power has become dominant with respect to dynamic power consumption, hence today most people believe that DPM is the most effective technique also in processing units.

This paper shows that, considering a more realistic task model derived from the analysis of actual executed code, tasks may exhibit different energy requirements, so requiring an integrated DVFS-DPM approach.

I. Introduction

Two widely used techniques to save energy are *Dynamic Voltage/Frequency Scaling* (DVFS) and *Dynamic Power Management* (DPM). DVFS approaches [1] decrease the voltage and/or frequency of the processor to reduce energy consumption, while DPM techniques [2] aim at putting the processor in a low-power inactive state as long as possible, but still guaranteeing tasks timing constraints. Nowadays, DPM techniques are more effective for reducing the overall energy consumption as they significantly affect the leakage dissipation whose impact has become dominant in actual hardware due to shrunk transistor size and low supply voltage. However, such energy reduction is obtained by taking into account only hardware features, without considering any characteristic related to the application. For instance, for intensive I/O-bound applications, task execution times are less dependent on the processor speed, hence the energy consumption ascribable to the leakage dissipation is barely affected by speed changes, offering the possibility of achieving higher reductions due to dynamic consumption by scaling the speed.

Experimental measurements show that DPM techniques work better for CPU bound tasks, while DVFS techniques are more appropriate for I/O bound tasks. Hence, this paper suggests to enrich the task model to consider the types of operations carried out by tasks, to apply the most appropriate technique or integrate them to achieve a better performance.

This paper also introduces several ideas for dealing with applications with mixed energy requirements. To the best of our knowledge, the state of art algorithms that cope with tasks and devices together consider only DVFS features for the task set and low-power states for the devices [3]. Moreover, tasks with different energy requirements was analyzed by Aydin et al. [4], but the solution considered only a DVFS approach for CPU bound tasks with different critical speeds.

II. System model

We consider a set Γ of n sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$ executing upon a single processor. The processor can vary the running speed s, defined as the normalized frequency with respect to the maximum frequency, $s = \frac{f}{f_{max}}$. The speed set is assumed to be finite and composed of m different speeds $s_1, s_2, \ldots, s_m$ sorted in ascending order, thus $s_{min} = s_1$ and $s_{max} = s_m = 1.0$.

Each sporadic task τ_i ($1 \leq i \leq n$) is characterized by a worst-case execution time (WCET) $C_i(s)$, which is a monotonic not decreasing function of the speed, a relative deadline D_i and a minimum inter-arrival time T_i, also referred to as the period. The WCET value of τ_i depends on the actual speed of the processor and is computed as $C_i(s) = \alpha_i C_i^{max} + (1 - \alpha_i)\frac{C_i^{max}}{s}$, where C_i^{max} denotes the amount of time required to execute τ_i at the maximum speed and $\alpha_i \in [0, 1]$ represents the fraction of execution time that does not scale with the speed (e.g., due to I/O operations). The larger α_i, the bigger the portion of τ_i that does not scale with the speed.

The power consumption of each gate depends on the supply voltage V and clock frequency f as reported in Equation 1:

$$P_{gate} = C_L V^2 p_s f + p_s V I_{short} + V I_{leak}. \quad (1)$$

A generic formulation to model the power consumption of the entire processor in the active state has been proposed by Martin et al. [5] as a function of the running speed, $P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0$.

Moreover, the processor provides a set of low-power states during which the task execution is suspended while the power consumption is low. Typically, sleep states characterized by a lower power consumption have longer wake-up times to restore the fully operating state. This feature leads to discarding several low-power states when the available idle time is shorter than such a wake-up time (also referred to as break-even time).

III. Motivational Example

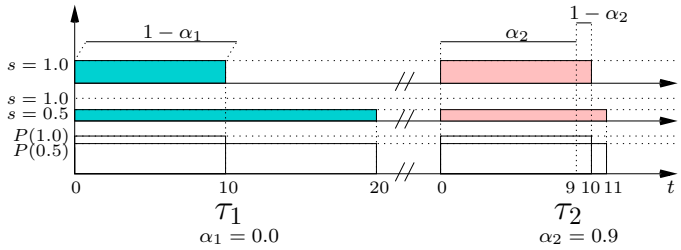

Fig. 1. Execution and consumption of τ_1 and τ_2 at speed 1.0 and 0.5

This section provides an example that shows how the type of code executed by a task (either CPU or I/O bound) can affect the strategy to minimize energy consumption, between pure DPM, according to which the task is executed at the maximum speed to exploit low-power states, and pure DVFS, where the processor is set at a slower speed reducing idle intervals.

Let us consider two tasks τ_1 and τ_2 with computation times $C_1(1.0) = C_2(1.0) = 10$, $\alpha_1 = 0$ and $\alpha_2 = 0.9$.

Concerning the hardware, let us consider a NXP LPC1768 equipped with an ARM Cortex M3. Such a processor supports frequencies within $[36, 96]$ MHz with steps of 4 MHz and the normalized power function is $P(s) = 0.4s + 0.6$ ($s_m = 1.0$ corresponds to 96 MHz). Note that, since this processor does not support voltage scaling, according to Equation 1 the power consumption can be considered a linear function of the speed. However, a quadratic or cubic power function would only decrease the impact of the dynamic power component. Considering the normalized power function reported above, 60% of the overall power consumption is due to static dissipation, while the remaining 40% is affected by the speed dependent part (which may be linear, quadratic or cubic).

For this architecture, the energy required for the execution of CPU bound code is minimized when the system runs at the maximum speed. For the sake of simplicity, let us exploit only the frequencies 48 and 96 MHz ($s_1 = 0.5$ and $s_2 = 1.0$).

The two values for α are obtained by executing, on the platform in use, the Coremark benchmark ($\alpha = 0.0$) and a test program that heavily exchanges data within an external storage unit ($\alpha = 0.9$).

Figure 1 shows the execution of the two tasks for different processor speeds with the corresponding power consumption. Note that when τ_1 (characterized by $\alpha_1 = 0.0$) executes at speed $s = 1.0$, its computation time is $C_1(1.0) = 10$ and the corresponding energy consumed by the processor is $10 \times P(1.0) = 10$. When the same task executes at speed $s = 0.5$, its computation time is $C_1(0.5) = 20$ and the consumed energy is $20 \times P(0.5) = 16$. For task τ_2 (with $\alpha_2 = 0.9$) the situation is quite different. Indeed, at speed $s = 1.0$, we have $C_2(1.0) = 10$, consuming an energy equal to $10 \times P(1.0) = 10$, whereas at speed $s = 0.5$, we have $C_2(0.5) = 11$, consuming an energy equal to $11 \times P(0.5) = 8.8$.

Hence, for CPU bound tasks, like τ_1, the energy is minimized at the highest speed $s = 1.0$, whereas for I/O bounded tasks, like τ_2, energy is minimized at lower speed ($s = 0.5$ in the example), leaving space for DVFS techniques.

IV. Open Questions

Given the different energy characteristics of CPU and I/O bound tasks, the open problem to be investigated is then to find energy-efficient scheduling strategies that mix DPM and DVFS approaches for exploiting the characteristics, in terms of computational time and energy-efficiency, of the software that composes the task set.

Two possible approaches have been identified, which may lead to a further energy saving.

The first approach consists of extrapolating from the task set and platform a generalized parameter representing the dominant behavior to find out which kind of technique is more appropriate (either DPM or DVFS). More precisely, such a parameter would be useful to select at design-time the most appropriate technique (DPM or DVFS).

A second and more sophisticated approach may exploit speed scaling according to the task in execution [4] (for instance, the speed would increase at the maximum value when running a CPU bound task). In practice, scaling speed for every task introduces a significant overhead [6] and affects the system reliability [7]. For overcoming these drawbacks the algorithm could try to compact the execution of tasks which belong to the same category, so reducing the number of speed scaling events.

References

[1] H. Aydin, P. Mejía-Alvarez, D. Mossé, and R. Melhem, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, 2001.

[2] M. A. Awan and S. M. Petters, "Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems," in *Proceedings of the 2011 23rd ECRTS*, 2011.

[3] V. Devadas and H. Aydin, "On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications," in *Proceedings of the 8th ACM international conference EMSOFT*, 2008.

[4] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," in *RTSS*, 2006.

[5] T. Martin and D. Siewiorek, "Non-ideal battery and main memory effects on cpu speed-setting for low power," *IEEE Transactions on VLSI Systems*.

[6] M. Bambagini, F. Prosperi, M. Marinoni, and G. Buttazzo, "Energy management for tiny real-time kernels," in *ICEAC*, 2011.

[7] B. Zhao, H. Aydin, and D. Zhu, "Energy management under general task-level reliability constraints," in *RTAS*, 2012.

Is Audsley's Scheme the Most Expressive Optimal Priority Assignment Algorithm?

Alan Burns
Department of Computer Science,
University of York,
York, YO10 5GH, UK.

1 Introduction

For fixed priority scheduling, the allocation of priorities to the application's tasks is a fundamental activity. If all possible allocations have to be checked (to find the optimal one) then a total $n!$ ordering must be examined (for n tasks). For any large n this is prohibitive. Fortunately for simple task models straightforward optimal priority schemes are available; for example rate monotonic for sporadic task sets with implicit deadlines [6], deadline monotonic for tasks with constrained deadlines [5] and deadline-jitter monotonic for tasks with constrained deadlines and release jitter [9].

Where these simple schemes are not applicable, for example with non-preemptive scheduling, or task sets with arbitrary deadlines or mixed criticality systems, there exists a scheme with n^2 complexity that is often applicable. This scheme was first proposed in 1991 and is generally known as Audsley's algorithm [1, 2]. Unfortunately not all task models have the prerequisites (see below) that allow this algorithm to be applied. Examples of such models are to be found in fixed priority multiprocessor scheduling. Often when Audsley's algorithm is found to be not applicable the only option remaining is to develop and use heuristics that aim for good, rather than optimal, priority orderings. This leads to the question being asked in this paper: *if Audsley's algorithm is not applicable can an optimal priority assignment scheme only be found by enumerating all n! possibilities?*

2 Audsley's Priority Assignment Scheme

An optimal priority assignment scheme is one that will find an ordering that will deliver a schedulable task set if a successful priority ordering exists for that task set. Obviously an enumeration of all possible orderings is optimal is that sense, but is impractical for large n.

To apply Audsley's algorithm requires the task model to satisfy a set of prerequisites [3, 4].

- The schedulability of a task may be a function of the set of higher priority tasks, but not their specific priorities.
- The schedulability of a task may depend on the set of lower priority tasks, but not on their specific priorities.
- A schedulable task that has its priority raised cannot become unschedulable, and conversely an unschedulable task that has its priority lowered cannot become schedulable.

Although these prerequisites are sufficient, it has not actually been proved that they are all necessary. *This is offered as a second open research question.*

The algorithm works by first finding any task that is schedulable at the lowest priority. If none can be found then the task set is unschedulable. If one is found then that task can be given this lowest priority, and the algorithm continues to try and allocate a task to the lowest+1 priority. This is repeated for all priority levels. If at any level no schedulable task can be found then the task set is unschedulable. Backtracking will not improve on this, so in the worst-case $n(n-1)/2$ single task schedulability tests need to be undertaken with this scheme.

A typical property that prevents the use of Audsley's algorithm is when the response-time of a task is not only dependent of the set of higher priority tasks but also on their relative priorities. For example, in some forms of multiprocessor analysis the amount of interference from a higher priority task is a function of the interval during which that task is executing. If the task's deadline is used to define this interval then this is a static property that does not depend on the task's priority; and hence Audsley's algorithm can be used. But more effective analysis is possible if calculated response time rather than deadline is used [3, 4]. Unfortunately the task's response-time will depend on its own priority and hence is not static. As a result Audsley's algorithm cannot be used. A trade-off must therefore be made, between an adequate form of analysis supported by optimal priority assignment, or a superior form of analysis only supported by sub-optimal priority assignment [3, 4].

Another example of non-compliance is with the Abort-Restart model [7]. Here tasks are aborted rather than preempted. When a task is released for execution its effective execution time is the sum of its own computation time and the maximum computation time of lower priority tasks. Hence computation time is a function of priority ordering with the result that the first prerequisite for Audsley's algorithm does not hold. Interestingly, an evaluation of a heuristic priority assignment scheme was only able to compare itself with an optimal ordering (derived via complete enumeration) for task sizes up to $n = 8$ [8]. After that $n!$ was too large for the required experiments (using thousands of task sets). Remember 10! is 3,628,800.

3 Open Research Question

The open research question is simple stated. *With uniform hardware and run-time support, if the prerequisites for Audsley's algorithm do not apply can an optimal priority ordering only be delivered by complete enumeration of all possible priority orderings?*

Here we are concerned only with optimal priority assignment. There are many effective heuristics, using for example backtracking and bounding. But in the worst-case (with, for example, a 'just' schedulable or 'just' unschedulable task set) an exponential number of checks need to be made or only a sufficient but not necessary pronouncement can result.

The Open Question requires the hardware to have uniform behaviour. By which is implied that tasks are treated fairly and equally. To illustrate this pre-condition consider a model that can be optimally assessed using an n^3 algorithm, but for which Audsley's algorithm does not formally apply. Assume a single processor with an accelerator that can execute the highest priority task at twice its usual speed. The single task must be statically mapped to the accelerator and will itself be schedulable if its computation time is less that its deadline (as its execution time is now $C/2$ rather than C). Now Audsley's algorithm does not apply as the schedulable at the lowest priority level will depend on which task will be given the highest priority (i.e. it will use the accelerator). However, it is easy to see that an optimal scheme exists. First, assign one (arbitrary) task to the top priority and then use Audsley's algorithm to see if the others can be scheduled. If they cannot, try a different task at the top level. In total n distinct tests need to be undertaken each requiring $(n-1)(n-2)/2$ steps; delivering n^3 complexity.

Although this example seems to give a counter example to the Open Question; it is contrived and would be excluded on the basis that the hardware is not uniform. In reality it is really a task allocation problem (rather than a priority assignment problem). So the question remains: do more realistic counter examples exist?

4 Conclusion

As far as the author is aware there are no published polynomial time optimal priority ordering schemes that do not conform to the prerequisites of Audsley's algorithm. If it is indeed the case that no such schemes can exist then this is a very useful result. It implies that if in any new scheduling scheme there is not this conformity then effort need not be wasted trying to derive optimal priorities. A sub-optimal heuristic is the best that can be employed.

A further open question raised in this paper concerns the prerequisite for Audsley's algorithm. Are they all necessary?

References

[1] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS164, Department of Computer Science, University of York, 1990.

[2] N.C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[3] R.I. Davis and A. Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, pages 398–409, 2009.

[4] R.I. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems Journal*, 47:1–40, 2011.

[5] J.Y.T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation (Netherlands)*, 2(4):237–250, 1982.

[6] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20(1):46–61, 1973.

[7] J. Ras and A.M.K Cheng. Response Time Analysis for the Abort-and-Restart Task Handlers of the Priority-Based Functional Reactive Programming (P-FRP) Paradigm. In *Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 305–314, 2009.

[8] H.C. Wong and A. Burns. Improved priority assignment for the abort-and-restart (AR) model. techreport YCS-2013-481, Department of Computer Science, University of York, UK, 2013.

[9] A. Zuhily and A. Burns. Optimal (D-J)-monotonic priority assignment. *Information Processing Letters*, 103(6):247–250, 2007.

Global Laxity-Based Scheduling on Multiprocessor Resource Reservations

Open Problem

João Pedro Craveiro and José Rufino
Universidade de Lisboa, Faculdade de Ciências, LaSIGE — Lisbon, Portugal
jcraveiro@lasige.di.fc.ul.pt, ruf@di.fc.ul.pt

I. Introduction

In uniprocessor scheduling, both EDF and LLF are found to be *optimal*, in the sense that both are able to schedule any feasible task set. EDF has received more preference and attention because, both being optimal, it leads to less preemptions than LLF. In global multiprocessor, neither EDF and LLF are optimal [1]. EDF, for instance, is subject to the Dhall effect [2], which causes relatively light task sets to be unschedulable. The prototypical task set for this effect has a high utilization task which, due to its characteristics, is relegated to a lower priority. LLF does not suffer from the Dhall effect, but is still characterized by a higher number of preemptions. To capture the best of these algorithms, Lee [3] has proposed EDZL (Earliest Deadline until Zero Laxity), a variation of EDF that promotes tasks to highest priority when they reach zero laxity; this allows scheduling task sets which are unschedulable with EDF [4].

In this paper, we focus on the problem of scheduling a task set on a *multiprocessor resource reservation*, i.e., in a scenario in which some (or all) processors may be at some times unavailable. We focus on the approach of adding laxity-based priority assignment decisions to EDF, as in the case of EDZL.

II. System model and background

Constrained-deadline sporadic task model: Each task, $\tau_i \stackrel{\text{def}}{=} (T_i, C_i, D_i)$, releases an infinite sequence of jobs, whose release times are separated by at least T_i time units. The jth job of task τ_i, $J_{i,j} \stackrel{\text{def}}{=} (a_{i,j}, e_{i,j}, d_{i,j})$ is released at time instant $a_{i,j} \geq a_{i,j-1} + T_i$ and must receive $e_{i,j} \leq C_i$ units of execution capacity within the time interval $[a_{i,j}, d_{i,j}[$ (with $d_{i,j} = a_{i,j} + D_i$). A job which has arrived and has not yet executed $e_{i,j}$ is said to be *active*. At some instant t, the *laxity* of an active job $J_{i,j}$ is the difference between how much time there is until the deadline ($t - d_{i,j}$) and the remaining execution (i.e., the units of execution capacity that such job must receive within the time interval $[t, d_{i,j}[$). We assume the first jobs of all tasks arrive at $t = 0$.

Scheduling algorithms: We consider that jobs are scheduled over the multiprocessor resource reservation using a *work-conserving unrestricted-migration global* policy. This means that, at each instant, the highest priority active jobs will be selected for execution; the number of jobs that will execute is upper-bounded by the number of available processors. An active job may be preempted from one processor and, eventually, resume execution on another processor. We now describe the priority assignment policies used by the algorithms refered to in this paper:

- **EDF** Active jobs are prioritized according to their absolute deadlines. The higher a job's deadline, the higher its priority.
- **LLF** Active jobs are prioritized according to their laxities. The less laxity a job has, the higher its priority.
- **EDZL** Jobs with zero laxity are assigned highest priority. The remaining jobs are prioritized as EDF.

Resource reservation: To describe the open problem, we will consider a simple model for a resource reservation $R \stackrel{\text{def}}{=} (\Pi, a, m)$, which is defined as a reservation of an identical multiprocessor platform (with m unit-capacity processors) for a consecutive time units every Π time units; this means that, at any instant, there are either 0 or m processors available. When a job is scheduled on one of the processors for one time unit, its remaining execution decreases by one unit. We also assume that the first a-long interval of availability of the resource starts at $t = 0$.

III. Motivating example and problem

We describe the open problem resorting to a contrived task set $\mathcal{T}$ — with independent tasks $\tau_1 = (20, 6, 20)$, $\tau_2 = (20, 7, 20)$, $\tau_3 = (20, 8, 20)$, and $\tau_4 = (21, 12, 21)$ — and a resource reservation $R = (20, 12, 3)$. When simulating[1] scheduling this example using EDF, we have the first job of task τ_4 miss a deadline at $t = 21$ (Fig. 1a); grayed-out boxes represent platform unavailablity, and the bar traced in orange is the job's execution beyond the deadline (its response time is 5 time units beyond the deadline). This is because the jobs of the remaining three tasks, having earlier deadlines, occupied the three processors for enough time to push the execution of τ_4's job forward. The observed phenomenon is the same we see in classical examples of the Dhall effect [2], which EDZL aims to eliminate. However, when simulating the scheduling of the same example using EDZL we see no difference (Fig. 1b). When the laxity of τ_4's job reaches zero, the jobs of the remaining tasks have already finished, so the promotion to the highest priority has no practical effect.

This work was partially supported by FCT/Égide (PESSOA programme), through the transnational cooperation project SAPIENT; by the EC, through project IST-FP7-STREP-288195 (KARYON); and by FCT, through project PTDC/EEI-SCR/3200/2012 (READAPT), multiannual funding to LaSIGE (UI 408), and Doctoral Grant SFRH/BD/60193/2009.

[1] Source code available at http://lasige.di.fc.ul.pt/~jcraveiro/EDzetaL/.

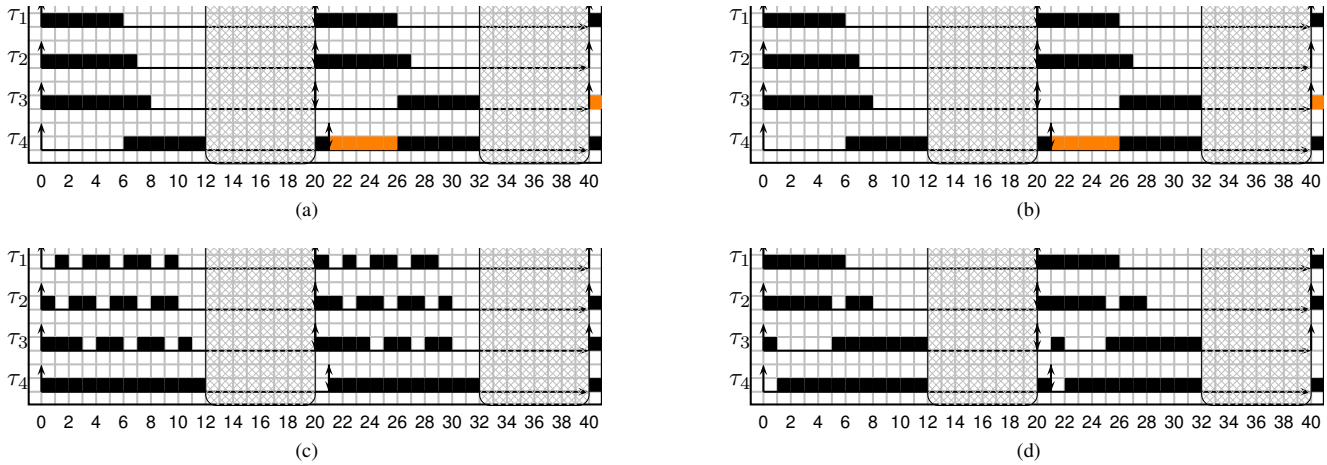

Fig. 1: Scheduling $\mathcal{T}$ on $R = (20, 12, 3)$ using (a) EDF; (b) EDZL; (c) LLF; (d) EDζL.

Problem: *The classical notion of laxity does not consider the unavailability of the platform. As such, comparing laxity to a constant value (e.g., EDZL) is inappropriate when dealing with resource reservations.*

We did not observe deadline misses when simulating with LLF either (Fig. 1c); the classical notion of laxity is not an issue for LLF, since it only compares the laxities of different jobs among them (not against a constant value, like EDZL). A higher number of preemptions is, however, clearly seen.

To the best of our knowledge, the problem of analysing laxity-based scheduling algorithms on multiprocessor resource reservations has not been approached in the literature.

IV. Proof of concept: EDζL

As a proof of concept, we have implemented an algorithm which we call Earliest Deadline until ζ Laxity (EDζL)[2]. The value of ζ may be greater than zero, and should be appropriate for the available resource reservation. For our simple example and model, we can simply assign $\zeta = \Pi - a$. We implement the priority assignment policy as follows: jobs with laxity $\leq \zeta$ are prioritized as LLF among them, all of them having higher priority than jobs with laxity $> \zeta$; the latter are prioritized among them as EDF. Any remaining ties are broken arbitrarily but consistently. This implementation of EDζL has the advantage of generalizing other algorithms: with appropriate values for ζ, EDζL behaves like EDF ($\zeta \ll 0$), LLF ($\zeta > \max_{\tau_i \in \tau}(D_i - C_i)$), or EDZL ($\zeta = 0$).

We have simulated scheduling our example using EDζL, with $\zeta = 20 - 12 = 8$, for 840 time units — $2 \times \mathrm{lcm}_{\tau_i \in \tau} T_i$. The trace for the initial 40 time units of the simulation can be seen in Fig. 1d. For the whole length of the simulation, no deadlines were missed, and there are less preemptions than with LLF (cf. Fig. 1c).

V. Conclusion and future work

We have presented the inappropriateness of EDZL for the case when scheduling is performed over a multiprocessor resource reservation — i.e., a multiprocessor platform which can be partially or totally unavailable at some times. We have illustrated the problem with a contrived example and (deliberately simple and optimistic) resource model, and shown EDζL as a proof of concept for the general idea of a possible solution. The main open questions are: **(i)** *What is the dynamics of ζ which achieves better performance?* **(ii)** *How can we determine schedulability with such a scheduling policy on various multiprocessor resource reservations* [6]–[9]*?* Our intuition is that the solution for this problem may be based on comparing an adapted notion of laxity to 0 (or, equivalenty, comparing classical laxity to a dynamic ζ); in turn, this adapted notion of laxity should consider the difference between how much execution capacity one single job can have available until its deadline (according to the resource model) and its remaining execution.

References

[1] M. L. Dertouzos and A. K. Mok, "Multiprocessor online scheduling of hard-real-time tasks," *IEEE Trans. Softw. Eng.*, vol. 15, no. 12, Dec. 1989.
[2] S. K. Dhall, "Scheduling periodic-time-critical jobs on single processor and multiprocessor computing systems," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1977.
[3] S. K. Lee, "On-line multiprocessor scheduling algorithms for real-time tasks," in *TENCON '94)*, Aug. 1994.
[4] T. P. Baker, M. Cirinei, and M. Bertogna, "EDZL scheduling analysis," *Real-Time Syst.*, vol. 40, no. 3, 2008.
[5] R. I. Davis and S. Kato, "FPSL, FPCL and FPZL schedulability analysis," *Real-Time Systems*, vol. 48, no. 6, 2012.
[6] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *ECRTS '08*, Jul. 2008.
[7] E. Bini, M. Bertogna, and S. Baruah, "Virtual multiprocessor platforms: Specification and use," in *RTSS '09*, Dec. 2009.
[8] G. Lipari and E. Bini, "A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation," in *RTSS '10*, Dec. 2010.
[9] A. Burmyakov, E. Bini, and E. Tovar, "The Generalized Multiprocessor Periodic Resource interface model for hierarchical multiprocessor scheduling," in *RTNS '12*, Nov. 2012.

[2] Akin to Fixed Priority until Static Laxity (FPSL) [5].

Uniform Multiprocessor Periodic Resource model

Progress Report and Ongoing Work

João Pedro Craveiro and José Rufino
Universidade de Lisboa, Faculdade de Ciências, LaSIGE — Lisbon, Portugal
jcraveiro@lasige.di.fc.ul.pt, ruf@di.fc.ul.pt

I. Introduction

Hierarchical scheduling is a current trend in embedded software, with applications ranging from multimedia applications [3] to safety-critical domains such as the civil aviation, aerospace and automotive industries [4]. The need for independent development and arbitrary number of levels are the main motivation and advantages of *compositional analysis*; compositionality is the property of a complex system that can be analysed by analysing some properties of its components (without knowing their internal structure or hierarchy) and the way they are composed. In this sense, a component comprises a workload, a scheduler, and a resource supply. Compositional analysis comprises three main points [5]. **(i) Local schedulability analysis** Analysing the schedulability of a component's workload upon its scheduler and resource supply. **(ii) Component abstraction** Obtaining the component's interface from its inner characteristics. **(iii) Interface composition** Transforming the set of interfaces abstracting the real-time requirements of individual subcomponents into an interface abstracting the requirement of scheduling them together according to a given *intercomponent scheduling* strategy.

Motivation and problem: Multiprocessors are entering the realm of embedded systems, namely safety-critical and hard real-time systems as those employed in the aerospace and the automotive industries. However, the multiprocessor capabilities are routinely not exploited, because of a lack of support in terms of verification and certification [6]. Due to these industries' prevalent use of and interest in TSP systems, compatibility between TSP and platforms with multiple processors, both identical and non-identical, is highly desired. The ARINC 653 specification, a standard for TSP systems in civil aviation and aerospace, shows limited support thereto [7]. To allow reusing the obtained results in a wider range of systems and applications, we approach this problem through compositional analysis of hierarchical scheduling frameworks (HSF), of which TSP systems are a special case.

Contributions: In [1], we presented the problem of compositional analysis of HSFs on uniform multiprocessors. In this fast abstract, we overview the progress made on the problem since then, and some ongoing work. For a detailed description with proofs, the reader is referred to [2]. This is the first work explicitly dealing with compositional analysis of HSFs on uniform multiprocessor platforms. Our contributions so far [2] are: **(i)** the uniform multiprocessor resource (UMPR) model to serve as a component interface for compositional analysis; **(ii)** a sufficient local schedulability test for sporadic task workloads using global EDF (gEDF) on the UMPR resource model; **(iii)** component abstraction guidelines to select the platform for the UMPR interface; We also briefly describe how we are dealing with the remaining open point — interface composition/intercomponent scheduling.

Related work

Compositional analysis: There is no previous literature explicitly dealing with compositional analysis of HSFs on uniform multiprocessor platforms. Approaches for identical multiprocessors include the multiprocessor periodic resource (MPR) model; an MPR interface (Π, Θ, m) abstracts the provision of Θ processing units over every period with Π time units length over a virtual platform consisting of m identical unit-speed processors [8]. The UMPR extends the MPR for uniform multiprocessors; we chose the MPR because of its simplicity and compositionality potential; other approaches present a less pessimistic approach, at the expense of less simple abstractions specifying the individual contribution of each processor in the virtual platform [9]–[11].

gEDF on dedicated uniform multiprocessors: Schedulability analysis of gEDF on uniform multiprocessors was introduced by Funk et al. [12]. Baruah & Goossens [13] provide a sufficient gEDF-schedulability test for constrained-deadline sporadic task sets of uniform multiprocessors; we extend their approach, since their analysis does not take into account that the platform may be at times partially or totally unavailable.

II. System model

Task model: A component $\mathcal{C}$ comprises a workload (task set) $\mathcal{T}$, with n constrained-deadline sporadic tasks $\tau_i \stackrel{\text{def}}{=} (T_i, C_i, D_i)$. We denote by $\delta_{\max}(\mathcal{T}) \stackrel{\text{def}}{=} \max_{\tau_i \in \mathcal{T}} \frac{C_i}{D_i}$ the maximum density among all tasks in $\mathcal{T}$. The *demand bound function* $\text{DBF}(\tau_i, t)$ gives an upper bound to the maximum cumulative execution requirement by jobs of sporadic task τ_i which have both their arrival and deadline times within any time interval with length t [13].

Platform and scheduling model: We assume a uniform multiprocessor platform with m processors, defined and represented as $\pi \stackrel{\text{def}}{=} \{s_i\}_{i=1}^{m}$, with $1.0 \geq s_i \geq s_{i+1} > 0.0$ for all $i < m$. Each s_i represents a processor's schedulable utilization; this value corresponds to the amount of processor capacity units it provides within one time unit; the total capacity of the platform is expressed as $S_m(\pi) \stackrel{\text{def}}{=} \sum_{i=1}^{m} s_i$. We also make use of the *lambda* parameter, $\lambda(\pi) \stackrel{\text{def}}{=} \max_{\ell=1}^{m-1} \frac{\sum_{j=\ell+1}^{m} s_j}{s_\ell}$, which and abstracts how close is π to an identical multiprocessor platform [12]. We also assume a work-conserving global EDF scheduling strategy with unrestricted migration; without loss of generality, we refer to it simply as gEDF.

This paper partly reports progress on the problem presented at RTSOPS 2012 [1], and summarizes results described in [2]. This work was partially supported by FCT/Égide (PESSOA programme), through the transnational cooperation project SAPIENT; by the EC, through project IST-FP7-STREP-288195 (KARYON); and by FCT, through project PTDC/EEI-SCR/3200/2012 (READAPT), multiannual funding to LaSIGE (UI 408), and Doctoral Grant SFRH/BD/60193/2009.

Resource model — UMPR: To solve the described problem, we propose the uniform multiprocessor periodic resource model, extending and generalizing the MPR model. An interface expressed with the *uniform multiprocessor periodic resource* (UMPR) model $\mathcal{U} \stackrel{\text{def}}{=} (\Pi, \Theta, \pi)$ specifies the provision of Θ units of resource over every period of length Π over a virtual uniform multiprocessor platform π. The supply bound function $\text{SBF}(\mathcal{U}, t)$ expresses the minimum resource supply $\mathcal{U}$ guarantees over *any* time interval of length t (see [2] for its definition).

III. Local schedulability test

We extend Baruah and Goossens's [13] "busy interval" analysis, taking into consideration that the processors available to the component at each instant may not be the fastest processors in the platform. Due to this, we introduce some pessimism regarding the upper bound on the number of jobs that carry in some execution into the "busy interval" (ν).

Theorem 1 (See proof in [2])**.** *Let $\nu = m - 1$. A component $\mathcal{C}$ comprising a constrained-deadline sporadic task set $\mathcal{T}$ is schedulable under gEDF using a UMPR interface $\mathcal{U} = (\Pi, \Theta, \pi)$, if for all tasks $\tau_k \in \mathcal{T}$ and all $A_k \geq 0$,*

$$\sum_{i=1}^{n} \text{DBF}(\tau_i, A_k + D_k) + (\nu + \lambda(\pi)) \cdot (A_k + D_k) \cdot \delta_{\max}(\mathcal{T}) \leq \text{SBF}(\mathcal{U}, A_k + D_k) \ . \tag{1}$$

IV. Component abstraction

For the (identical) multiprocessor resource model, the technique presented in [8] to generate the component interface for $\mathcal{C}$ consists of **(i)** assuming Π is specified by the system designer; **(ii)** computing the values of Θ and m so that $\mathcal{C}$ is schedulable with the least possible resource bandwidth (Θ/Π); for the computation of the schedulability to become tractable, the SBF is replaced by a linear lower bound. For the UMPR, component abstraction is not this simple, because the notions of numbers of processors and total capacity are no longer represented together as only m (and consequently the number of candidate virtual platforms for each component explodes). On the other hand, the available physical platform may impose restrictions on this. As such, we assume both period Π and platform π are specified (for instance, as a requirement presented to the component's developers by their contractor, which will in the end integrate the multiple components), and only Θ needs to be computed to guarantee schedulability. We nevertheless provide important analytically proven guidelines for the selection of platform π.

1) For the same number of processors and total capacity, UMPR interfaces with non-identical uniform multiprocessor platforms are better than those with identical multiprocessor platforms. This is coherent with previous findings in the literature [13].
2) UMPR interfaces with platforms providing the same total capacity with a lower number of faster processors are better than those with a greater number of slower processors.

Our experiments with randomly generated task sets are consistent with these guidelines.

V. Ongoing: Intercomponent scheduling and interface composition

We have proven experimentally [2] and analytically that, in the presence of uniform multiprocessor platforms, classical gEDF scheduling does not guarantee compositionality. We currently have the definition of a scheduler, umprEDF, that guarantees that the effective supply that each component receives is consistent with the interface derived for it. We are working on formally proving its properties, and on interface composition (deriving a global-level resource interface).

Acknowledgments. *The authors would like to thank I. Shin and A. Easwaran for being available to answer our doubts about specific aspects of their work, and the organizers and attendees of RTSOPS 2012 for the fruitful discussions there.*

References

[1] J. P. Craveiro and J. Rufino, "Heterogeneous multiprocessor compositional real-time scheduling," Fast Abstract in RTSOPS 2012, Jul. 2012.
[2] ——, "Compositional hierarchical scheduling frameworks on uniform multiprocessors," University of Lisbon, DI–FCUL, Tech. Rep. TR-2012-07, 2012, revised January 2013). [Online]. Available: http://hdl.handle.net/10455/6891
[3] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *RTSS '98*, Dec. 1998.
[4] J. Rufino, J. Craveiro, and P. Verissimo, "Architecting robustness and timeliness in a new generation of aerospace systems," in *Architecting Dependable Systems VII, LNCS 6420*, A. Casimiro, R. de Lemos, and C. Gacek, Eds. Springer, 2010, pp. 146–170.
[5] I. Shin and I. Lee, "Compositional real-time schedulability analysis," in *Handbook of Real-Time and Embedded Systems*, I. Lee, J. Y.-T. Leung, and S. H. Son, Eds. Chapman & Hall / CRC, 2007.
[6] J. Anderson, S. Baruah, and B. Brandenburg, "Multicore operating-system support for mixed criticality," in *CPS Week 2009 Workshop on Mixed Criticality*, San Francisco, CA, USA, Apr. 2009.
[7] J. Craveiro, J. Rufino, and F. Singhoff, "Architecture, mechanisms and scheduling analysis tool for multicore time- and space-partitioned systems," *ACM SIGBED Rev.*, vol. 8, no. 3, Sep. 2011, ECRTS '11 WiP.
[8] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *ECRTS '08*, Jul. 2008.
[9] E. Bini, G. Buttazzo, and M. Bertogna, "The multi supply function abstraction for multiprocessors," in *RTCSA '09*, Aug. 2009.
[10] E. Bini, M. Bertogna, and S. Baruah, "Virtual multiprocessor platforms: Specification and use," in *RTSS '09*, Dec. 2009.
[11] A. Burmyakov, E. Bini, and E. Tovar, "The Generalized Multiprocessor Periodic Resource interface model for hierarchical multiprocessor scheduling," in *RTNS '12*, Nov. 2012.
[12] S. Funk, J. Goossens, and S. Baruah, "On-line scheduling on uniform multiprocessors," in *RTSS '01*, Dec. 2001.
[13] S. Baruah and J. Goossens, "The EDF scheduling of sporadic task systems on uniform multiprocessors," in *RTSS '08*, Nov./Dec. 2008.

Exploiting Period Compatibility for Partitioned Rate-monotonic Scheduling using Circular Statistics

Dirk Müller, Matthias Werner
Operating Systems Group, Chemnitz University of Technology
D-09111 Chemnitz, Germany
{dirkm,mwerner}@cs.tu-chemnitz.de

1 Introduction

The choice of periods in a periodic task set has a significant impact on its utilization bound under Rate-monotonic Scheduling (RMS). It ranges from the Liu/Layland bound [3] in the worst, and thus general, case to the full uniprocessor utilization of 1 in the case of simply periodic (aka harmonic) task sets.

First, it is a challenge to exploit period values in order to obtain better uniprocessor utilization bounds also for non-extreme cases. A better grouping has then the potential of improving partitioned RMS on a multiprocessor system. Finally, based on such an understanding, period modification suggestions can be given in a rapid system design process.

2 Circular Statistics

Standard statistics is mainly based on scalar data. Opposed to this, directional statistics makes use of unit vectors with the information contained in their directions. In two dimensions, directional statistics specializes to circular statistics, see, e.g., [2]. Here, the angular direction is taken modulo 360°. A full circle is a period.

The example of determining an average *wind direction* will show the need for circular statistics. The average of two data points, Northeast (NE) and Northwest (NW), shall be calculated. Taking the numerical average of the corresponding azimuth (north-based, clockwise) values 45° and 315° results in an average wind direction of 180°, i.e. South (S). But the only reasonable result is North (N), the opposite direction. This example already indicates that linear statistics is not appropriate for such types of data.

Due to the local approximation of the Earth's surface by a plane, geography is a relevant application field of circular statistics as could be seen in the wind direction example. Other appropriate applications of circular statistics are the analysis of time series with respect to repetitive time events like *time of day* or *day of the week*. More general, all data resulting from *modulo* operations are suitable for an analysis by circular statistics. The time period (the divisor) is then mapped to a full circle of 360°.

3 Circular Range for Measuring Period Compatibility under Uniprocessor Preemptive RMS

A utilization bound taking all period values into account was first proposed by Burchard *et al.* [1]. There, so-called S values are defined as fractional parts (modulo 1) of the binary logarithms of the periods. Then, β is defined as the linear range of all obtained S values. The case $\beta = 0$ means a simply periodic task set, while a $\beta \to 1$ corresponds to a most-difficult-to-schedule situation according to Burchard's test.

Utilization bounds based on β turned out to be pessimistic [4] [5]. The simple dispersion measure *linear range* is not appropriate for the modulo-based S values and leads to the paradox situation of scale-dependent (i.e., dependent upon the choice of time unit) utilization bounds, cf. Fig. 1. As mentioned in the introduction, modulo-based data shall be treated as circular data. Hence, *circular range* coined β' instead of β shall be used. The problem with the old β approach is that it maps zero to powers of two. Thus, periods slightly below and sligthly above a power of two, say 15 and 17, are considered to be incompatible which is wrong. As shown in [4] and [5], this small modification leads to both less pessimism and a simpler utilization bound formula based on the Burchard bound [1].

The change of perspective from linear to circular statistics was crucial for these improvements. An at-first-glance tiny cosmetic operation turned out to be conceptually correct and to increase the obtained utilization bound which is a classic goal of the design of this kind of schedulability tests.

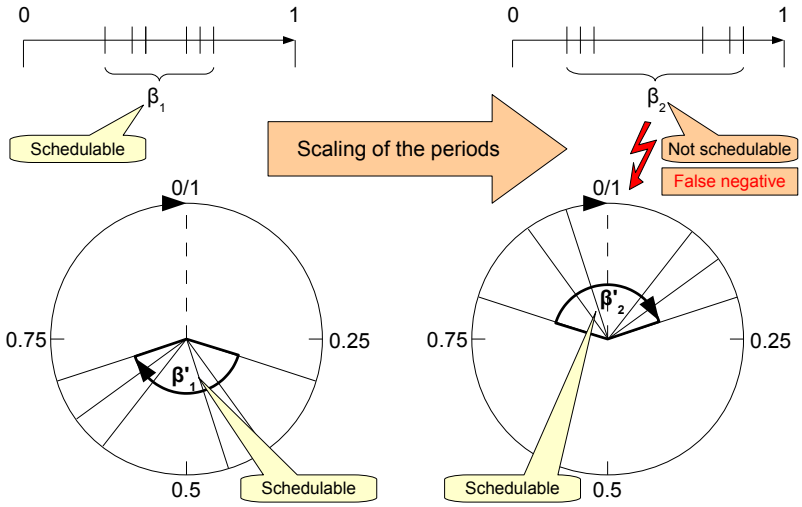

Figure 1: Period Scale Invariance of Circular Range of S values (below), $\beta'_1 = \beta'_2$, vs. Period Scale Dependency, $\beta_1 < \beta_2$, of Linear Range of them (above); based on [4]

4 Challenges

1. The grouping of tasks with compatible periods based on the linear S value range β was successful in the RMST algorithm [1] which tackles partitioned multiprocessor RMS. What is the gain of a change from linear to circular range there? How beneficial is it for hierarchical scheduling with RMS?

2. A typical situation in system design is the question for reasonable period modifications in order to make a task set schedulable. A viable approach for reducing circular range could be *circular autocorrelation*. The result is then a subset of tasks to be scaled and the scaling factor. How useful is such an approach and how does it compare to other system design methods?

5 Summary

It was shown that period compatibility estimation under RMS can be improved using circular instead of linear statistics which leads to increased uniprocessor utilization bounds and removes scale dependency. Since the uniprocessor schedulability test is one of the cornerstones of partitioned RMS, we ask for the usefulness of the approach for partitioned multiprocessor scheduling.

References

[1] Almut Burchard, Jörg Liebeherr, Yingfeng Oh, and Sang H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *Computers, IEEE Transactions on*, 44(12):1429–1442, December 1995.

[2] S. Rao Jammalamadaka and A. Sengupta. *Topics in Circular Statistics*. World Scientific Pub Co Inc, Singapore, har/dskt edition, 2001.

[3] Chung Laung Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[4] Dirk Müller. Period Fitting for Rate-monotonic Scheduling Using a Circular Similarity Measure. In *Proc. of the 18th IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA 2012), WiP Session*, pages 427–430, Seoul, Korea, August 2012.

[5] Dirk Müller and Matthias Werner. Comment on "New Strategies for Assigning Real-Time Tasks to Multiprocessor Systems". *Computers, IEEE Transactions on*, PP(99):1, 2012. to appear, http://dx.doi.org/10.1109/TC.2012.244.

Reality Check: the Need for Benchmarking in RTS and CPS

Marco Di Natale
Scuola Superiore S. Anna
email: marco@sssup.it

Chuansheng Dong
McGill University
email: chuansheng.dong@mail.mcgill.ca

Haibo Zeng
McGill University
email: haibo.zeng@mcgill.ca

Compared with other embedded systems research communities, including electronic design automation (EDA), compilers, or signal processing, the real-time systems (RTS) and the more recent cyber-physical systems (CPS) communities suffer from the lack of uniformly accepted benchmarks and even models on which to evaluate algorithms and solutions. Correspondingly, and not surprisingly, there is lack of consensus on what is *realistic* or *applicable*.

The problem is not minor as most researchers in the area continue to experience such a difficulty. It is also quite fundamental, touching upon very well established results, models, and techniques. We shortly review (and possibly challenge) some methods, models, and tools, and then move to a proposal for a shared (and public) repository of test cases and benchmarks.

Of course it is not by chance that our community lacks consensus on benchmarks or realistic system models. The main reason is probably the need for high-level system models, which are often application-specific (as opposed to general-purpose and well established algorithms that make up most of the benchmark libraries for other communities). Any application-specific model is often considered as part of the company IP (if industrial) and name masking is typically not enough to remove concerns. Hence, the only option is to patiently wait until the product (line) gets out of the market. A secondary reason is that citation-oriented metrics are inevitably leading to diminishing returns from realistic system-level (applicable) research and possibly to more limited interaction(s) with the industrial world.

Even if available, an industrial task model is probably not the best benchmark. Industrial task models are often the product of a designer's interpretation of the code-level solution to a functional problem. In many cases, the designer will seek a simplified task model to reduce concurrency and simplify its job (dealing with race conditions). The task model should be considered as the product of a design synthesis or optimization activity and the functional model, when existing, should always be the preferrable input.

I. System (task and message) models

Several methods and algorithms that are developed in literature are often validated against randomly generated task sets, where the task set complies with a defined activation, synchronization (dependency) and communication model. Quite often task sets are treated as abstract entities and uniform coverage of the space of attributes is sought. For example, confuting the analysis in [6] that stated that the average least upper bound for fixed-priority rate monotonic scheduling is 89%, the UUniFast algorithm [1] was developed to randomly generate task sets with a more accurately *uniform* distribution of task periods and utilizations. In reality, task periods are not arbitrarily selected, but are often the result of oversampling and undersampling at fixed ratios, giving rise to (pseudo-)harmonic sets or constranied by the processing rates of commercial off-the shelf smart sensors.

Also, the community has analyzed several task models as logical evolution of existing ones or as a possible representation of program-level constructs. Several of these did not originate from an application problem defined as a set of tasks but as a possible variation/extension of previous models. The models took a life of their own and researchers are often dealing with the backward work of finding a real-world motivation or match to the task model (and scheduling problem) they know how to solve.

In many real-world cases, the communication and synchronization problems are either much simpler (simple communication by sampling), or not easily represented by using the existing models (such as the activation modes in the Controller Area Network interaction layer, and the scheduling problems defined by synchronous state machine models [8]). As a further example, the model of linear transactions (possibly with offsets) is much more studied but much less common than a more general model of tasks and messages interacting in a graph pattern.

A. Realism and Details

In general, any engineering model should be detailed enough to capture the system attributes of interest to provide an accurate analysis of selected properties. Although aspects like the impact of cache, context switch, interrupt handling delays, and other architectural aspects can play a significant roles in some cases, they are often hidden by considering them (usually after upper-bounding) as part of other system parameters (such as the WCET of tasks) or neglected altogether, under the assumption that their impact is negligible.

Common examples include the assumption that context switch costs are much smaller than the execution times of tasks. However, measures of typical context switch times

for real operating systems on real execution platforms are not often readily available to validate this assumption and experimental results on real OS are required.

Similarly, tradeoffs between policies should be defined by the actual times that are used to perform operations. One example is the comparison between locked-based multiprocessor synchronization protocols like MPCP and MSRP. Unfortunately, a large body of study on this topic has assumed that the critical sections can be larger than $500\mu s$ or even $1ms$, which is probably not realistic for critical sections implementing access to shared memory buffers. Even with a very large data buffer (around 1k byte), the access time is measured to be in the range of $50\mu s$ on a 120MHz embedded microprocessor. Examples of (possibly longer) critical sections in (real-world) applications are required.

B. Random Task/System Generation

When a randomly generated system configuration is acceptable, standarized tools are needed to make experiments repeatable. Among the available generators, TGFF (Task Graph For Free) [2] and SMFF (System Model For Free) [7] are the most popular.

TGFF [2] is widely used in research domains other than real-time systems. It generates random task models, including the task parameters (periods and WCETs), the communication topology, and application-level timing constraints (end-to-end deadlines). However, TGFF does not represent the execution platform and the mapping of tasks. To partially address this issue, SMFF [7] covers the description of the entire system, including the hardware architecture, the software applications, and their mapping onto the platform with the associated scheduling and timing parameters.

Of course, the relevance to reality of these random task/system generation tools is largely decided by the parameter settings. It is also restricted by the underlying assumption in the random generation algorithm. For example, TGFF assumes that the task graph is directed acyclic (which is not suitable for tasks captured by cyclic graphs such as finite state machines), while SMFF assigns task activation periods assuming a uniform distribution between the specified minimum and maximum values.

C. Available Benchmarks

EEMBC [3] is one of the earliest efforts towards developing performance benchmarks for use in embedded systems. It organizes the test suites by specific focus of embedded systems hardware and software development, for application domains including automotive, consumer electronics, and telecommunications, or to address specific design concerns such as energy and floating point performance.

MiBench [4], collected in the early 2000s, follows the model of EEMBC by dividing a set of 35 applications (available as C code) into 6 categories, including automotive and industrial control, consumer devices, office automation, networking, security, and telecommunications.

For a specific purpose of evaluating WCET analysis methods and tools, the Mälardalen WCET research group maintains a benchmark [5], containing 35 programs (provided in C source files) collected from several different research groups and tool vendors around the world.

However, these benchmarks have their limitations. EEMBC is not freely available, access requires a membership with the associated cost. MiBench (besides its out-of-date) and Mälardalen benchmark are collections of programs rather than entire systems with a defined task structure.

II. A website for benchmark

Of course, this lack of benchmarks asks for concrete action. We present our project for the construction of a website [9] meant to store, classify, manage, and provide access to tools, tests and examples constructed and accessible using an open text-based format.

The website (or portal if preferred) is organized with sections for tools (for random generation) and real-world examples or case studies, classified in turn along two dimensions: type of models (Functional, describing functions and signals; Task, with tasks and messages; and Platform models, with physical architectures and possibly OS, device driver or communication stack models), and according to the execution platform (single-core, multicore, or distributed).

Examples are available and can be provided in an open text format, according to the description provided in the format section. We started collecting message sets and application descriptions from automotive systems. The website will be hosted at McGill University [9], with the collection of the above mentioned tools and benchmarks. Please feel invited to contribute with representative test cases and links.

References

[1] E. Bini and G. Buttazzo. Measuring the Performance of Schedulability Tests. *Real-Time Sys.*, 30(1-2):129–154, May 2005.

[2] R. Dick, D. Rhodes, and W. Wolf. TGFF: task graphs for free. Available at http://ziyang.eecs.umich.edu/ dickrp/tgff/

[3] The Embedded Microprocessor Benchmark Consortium. Website: http://www.eembc.org/

[4] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A free, commercially representative embedded benchmark suite. Available at http://www.eecs.umich.edu/mibench/

[5] WCET benchmarks. Available at http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

[6] J.P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. 10th Real-Time Systems Symposium*, 1989.

[7] M. Neukirchner, S. Stein, and R. Ernst. SMFF: System Models for Free. Available at http://smff.sourceforge.net/

[8] H. Zeng and M. Di Natale. Schedulability Analysis of Periodic Tasks Implementing Synchronous Finite State Machines. *Proc. 23rd Euromicro Conference on Real-Time Systems*, 2012.

[9] An Open Repository for Real-Time Benchmarks. Available at http://www.ece.mcgill.ca/hzeng3/benchmarks.html

Schedulability Analysis with an Extended Timing Definition Language

Tomasz Kloda Bruno d'Ausbourg Luca Santinelli
ONERA Toulouse, name.surname@onera.fr

I. Introduction : the time triggered languages context

The development of embedded software is a highly platform dependent process. The main difficulty lies in both formulating the functional specification of the system and correctly determining its temporal behavior. While the former is facilitated by the high-level programming languages which abstract from many hardware aspects, getting the expected temporal characteristic of the system involves usually much more efforts due to the implementation of scheduling policies and synchronization or inter-processes communication protocols.

To answer the problem of managing efficiently these two crucial aspects to the correctness of the system features, time triggered languages for embedded programming were devised. These languages are based on a clear separation between the functional part of applications and their timing definition. The basic functional unit in these languages are tasks that periodically execute some piece of code. Each task τ_i is characterized by its worst case execution time C_i. Several concurrent tasks make up a *mode*. A mode is characterized by its period that is a common multiple of the periods of tasks that make up the mode. Tasks are invoked within the mode at a declared frequency F_i that determines their period T_i. Tasks can be removed or added by switching from one mode to another. Tasks communicate with each other, as well as with sensors and actuators, only by means of ports.

Time triggered languages provide a programmer's abstraction which was firstly introduced within the *Giotto* programming language [5], [6], [7]. Giotto assigns a Logical Execution Time (LET_i) [8] to each task τ_i. Periodic invocation of tasks, reading of sensor values, writing of actuator values, completions of tasks and mode switches are all triggered by real time. They take place in the LET interval assigned to the task. Thus, tasks are invoked and sensors are read at the start instant of this LET interval while actuators are updated and tasks are completed at the end instant of the LET. Mode switches may occur only at instants defined by the common multiples of the LET of all the tasks belonging to that mode when they are all completed. It must be noted that in these languages the LET of a task is restricted to be equal to its period : $\forall \tau_i \; LET_i = T_i$. T_i and LET_i time intervals are then equal. This means considering that a task τ_i is released at the start instant of T_i and is completed at the end instant of T_i.

The LET abstraction and its time-triggered semantics enables efficient reasoning about the time behavior of applications. This allows control designers and developers to focus on the control systems aspects instead of the platform (hardware and operating system) without being interested in where, how and when tasks are actually scheduled: this may be on platforms with a single CPU or with many CPUs, on platforms with a preemptive priority scheduling or not. The compiler is in charge of generating the timing code for the given platform and needs to ensure that the LET semantics is preserved. To this end, it must guarantee the schedulability for a specific platform and so is faced to a possibly complex schedulability problem because the actual scheduling scheme of the operating system and the schedulability test of the compiler must be compatible.

Based on the concepts developed by Giotto, the Timing Definition Language (TDL) [2], [3], [14] introduced additional features together with appropriate tools for structuring large real-time systems and analyzing their schedulability, and proposed new syntactical and semantic modular structures for describing the temporal behavior of applications. In particular a set of periodic tasks and the set of the different modes of execution of these tasks may be declared inside a module definition. An application can be devised as a set of modules that are the actual distribution unit in TDL. Tasks of modules are concurrently activated. Nonetheless, each module can execute only the tasks of its current mode. Mode switches may occur only at specific time instants that are hyperperiods of tasks in the current mode. Figure 1 depicts a sample physical execution pattern of the exemplary TDL system composed of three different modules: M_1, M_2, M_3. Module M_1 can execute tasks of modes m_{11} or m_{12}, M_2 those of modes m_{21} or m_{22} and M_3 these of modes m_{31} or m_{32}.

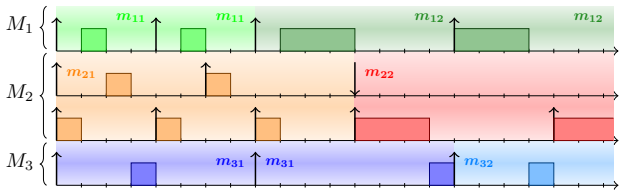

Fig. 1: Sample execution pattern of TDL system scheduled with fixed priority

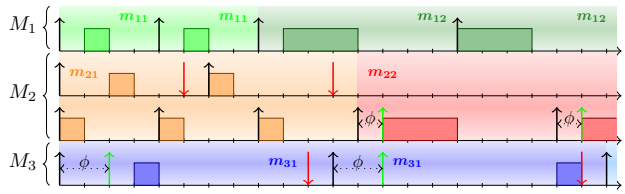

Fig. 2: Sample execution pattern of TDL-extended system scheduled with fixed priority

II. The problem statement

We feel that Giotto and TDL framework may be extended in three ways. Firstly, we suggest that the LET of a task and its period denote in fact two different concepts. Consequently, we allow that LET assumes values less than task period: $LET_i \leq T_i$. Differently from what was suggested in section I, the LET_i interval is included in the T_i interval. Then a task may be released after the beginning of its period and may be completed before the end of this period. Therefore, this means extending the TDL task definition $\tau_i = (C_i, LET_i)$ with an initial offset ϕ_i and a $LET_i \leq T_i$ $\tau_i = (\phi_i, C_i, LET_i, T_i)$. Figure 2 represents a sample physical execution

pattern of 3 TDL modules with these new extensions. The actual deadline D_i of task τ_i is given by $D_i = \phi_i + LET_i$. Secondly, we suggest that an application may be heterogeneous and may combine time triggered components (with their timing definition) and classical event triggered components (without timing definition). Thirdly, we suggest that mode changes should be made more reactive by allowing mode switches not only at instants that are hyperperiods but also at some possible other idle (or made idle) instants in the mode. As it can be seen on Figure 2 there is an idle interval in module M_3 between the deadline (end of its LET) of the only task of mode m_{31} (red arrow) and the end of the m_{31} mode (black arrow). Therefore, to provide better reactivity, it might be considered to advance mode change instant instead of waiting idly until the end of the mode period. Moreover, when a mode change is requested, some tasks whose completion is not vital for the consistency of the system, might be aborted [10].

The question is now: what new schedulability problems does the compiler of the timing definition language face under these three new hypotheses ?

III. Current state

A schedulability analysis of TDL definitions, where tasks $\tau_i = (C_i, LET_i)$, was developed in [2]. The author proposes schedulability tests for Fixed Priority (FP) and Earliest Deadline First (EDF). Because $\forall \tau_i \ LET_i = T_i = D_i$ and the critical instant occurs at simultaneous start of the tasks from all the modules in this TDL scheme, the EDF schedulability analysis is performed by evaluating the utilization factor : the sum of maximal modules utilization factors cannot exceed 1. FP policy schedulability analysis is based on the evaluation of the worst-case response time taking into account mode changes. The line of reasoning founding this analysis follows the general idea detailed in [10].

IV. Open schedulability problems arising in our extended framework

Our objective is to establish schedulability tests for FP and EDF under the three new hypotheses that define an extended TDL model and that were discussed in section II : disjointing LET from task period ($LET_i \leq T_i$), performing schedulability analysis with the assumption that processor resources can be shared between timing defined tasks and not timing defined ones, and finally improving the reactivity of the mode change.

Relaxing constraint which was fixing LET to the task period, in the domain of scheduling boils down to the analysis of real-time tasks described by the four parameters model (Φ_i, C_i, D_i, T_i) calculated according to the principles exposed in section II. The notion of modes and modules implies that the system taskset is composed of tasksets from all modules of the system and each one of these tasksets is allowed to interchange old mode tasks with the new ones at precisely defined mode switch instants. Selecting these instants only at common multiple of all tasks periods within given mode determines that the mode change is synchronous within the module because the old mode tasks are necessarily completed when the new ones are released. Nonetheless, as the mode change within some module can occur during the mode execution of another module, its interference should be properly examined during schedulability analysis. Moreover, the time interval between starts of different modes belonging to distinct modules can assume the values depending on these modes periods and their predecessors periods. The values of these time intervals can be examined in the similar manner as the tasks offsets in [11].

To handle heterogeneous systems we feel that it could be worthy to combine into the scheduling analysis the *Delivery Curves* and the *Remaining Delivery Curves* as introduced in [15]. In this way it would be possible to evaluate how much of processor a time defined modules demand in a time interval Δ and then to check that this demand remains compatible with the given processor consumption of all the other not time defined elements. To this end, if mode periods in different modules are selected so that their start instants are not correlated and, as a consequence, the highest demand intervals of both modes can occur at the same time, the approaches for EDF proposed by Phan in [12] and Fisher in [4], based on the *processor demand criterion* [1], may be used and may give satisfactory results. Otherwise, a method can be proposed where the allocation of the processor resources to modules is not only a function of the the time interval but also of the specific time instant this interval starts. The problem of scheduling FP tasks is concerned by the similar difficulties, however, as the approaches based on the *response time* analysis as well as *Real Time Calculus* provide sufficient conditions, a correct choice should be made.

As regards mode changes, in [9] Martinek presents some new protocols for *Giotto* aiming to increase their promptness. Once our previous questions and problems will be answered, we should like to find some improvements that allow more reactive mode changes in our timing definitions. Particularly, its impact on the time interval between start instants of modes in distinct modules is a key point that should be examined.

References

[1] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor. *Real-Time Syst.*, 2(4):301–324, Oct. 1990.

[2] E. Farcas. *Scheduling Multi-Mode Real-Time Distributed Components*. PhD Thesis, Department of Computer Science, University of Salzburg, July 2006.

[3] E. Farcas, C. Farcas, W. Pree, and J. Templ. Transparent Distribution of Real-Time Components Based on Logical Execution Time. In *LCTES*, pages 31–39, 2005.

[4] N. Fisher and M. Ahmed. Tractable Real-Time Schedulability Analysis for Mode Changes under Temporal Isolation. In *ESTImedia*, pages 130–139, 2011.

[5] T. A. Henzinger. Composable Code Generation for Distributed Giotto. In *LCTES*, pages 21–30, 2005.

[6] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Embedded Control Systems Development with Giotto. In *LCTES/OM*, pages 64–72, 2001.

[7] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A Time-Triggered Language For Embedded Programming. In *EMSOFT*, pages 166–184, 2001.

[8] C. M. Kirsch and A. Sokolova. The Logical Execution Time Paradigm. In *Advances in Real-Time Systems*, pages 103–120, 2012.

[9] N. F. Martinek and W. Pohlmann. Mode Switching in GIA – An ADA Based Real-Time Framework. Department of Scientific Computing, University of Salzburg.

[10] P. Pedro and A. Burns. Schedulability Analysis for Mode Changes in Flexible Real-Rime Systems. In *ECRTS*, pages 172–179, 1998.

[11] R. Pellizzoni and G. Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems*, 30(1-2):105–128, 2005.

[12] L. T. X. Phan, I. Lee, and O. Sokolsky. Compositional Analysis of Multi-Mode Systems. In *ECRTS*, pages 197–206, 2010.

[13] W. Pree and J. Templ. Modeling with the Timing Definition Language (TDL). In *ASWSD*, pages 133–144, 2006.

[14] J. Templ. TDL Specification and Report, 2003. Department of Computer Science, University of Salzburg.

[15] L. Thiele, S. Chakraborty, and M. Naedele. Real-Time Calculus for Scheduling Hard Real-Time Systems. In *ISCA*, volume 4, pages 101–104, 2000.

Improvements to Static Probabilistic Timing Analysis for Systems with Random Cache Replacement Policies

Robert I. Davis
Real-Time Systems Research Group, Computer Science Department, University of York, UK
rob.davis@york.ac.uk,

I. INTRODUCTION

Critical real-time systems such as those deployed in space, aerospace, transport, and medical applications require guarantees that the probability of the system failing to meet its timing constraints is below an acceptable threshold (e.g. a failure rate of less than 10^{-9} per hour). Advances in hardware technology and the large gap between processor and memory speeds, bridged by the use of cache, make it difficult to provide such guarantees without significant over-provision of hardware resources. The use of deterministic cache replacement policies means that pathological worst-case behaviours need to be accounted for, even when in practice they may have a vanishingly small probability of actually occurring. Further, the quality of deterministic WCET estimates for such systems can be highly sensitive to missing information, making them overly pessimistic. Random cache replacement policies negate the effects of pathological worst-case behaviours while still achieving efficient average-case performance, hence they provide a means of increasing guaranteed performance in hard real-time systems [6].

Determining the timing behaviour of applications running on a processor with a random cache replacement policy requires probabilistic analysis of worst-case execution times. This can be achieved using Static Probabilistic Timing Analysis (SPTA) to compute an upper bound on the exceedance function (1 - CDF) for the probabilistic Worst-Case Execution Time (pWCET) of a program. An example exceedance function is given in Figure 1, taken from [3]. From the exceedance function, it is possible to read off for a specified probability, an execution time that has that probability of being exceeded on any single run. Static Probabilistic Timing Analysis (SPTA) has been developed for single processor systems assuming both evict-on-access [2], [1] and evict-on-miss random cache replacement policies [3].

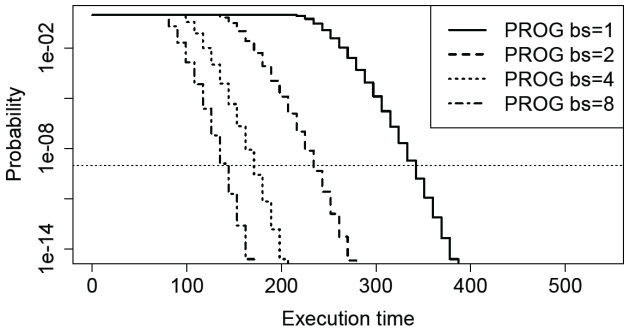

Figure 1: pWCET distributions (1-CDF) for different memory block sizes

II. EXISTING STATIC PROBABILISTIC TIMING ANALYSIS

We now recap on SPTA for an evict-on-miss random cache replacement policy [3] for the instruction cache and no data cache. With the evict-on-miss policy, whenever an instruction is requested and is not found in the cache, then a randomly chosen cache line is evicted and the memory block containing the instruction is loaded into the evicted location. We assume an N-way associative cache, and hence the probability of any cache line being evicted on a miss is $1/N$.

For simplicity, we assume a single path program comprising a fixed sequence of instructions. We represent these instructions via the memory blocks they access; with a superscript indicating the *re-use distance k.* (The re-use distance is the maximum number of evictions since the last access to the memory block containing that instruction, and is omitted if it is infinite). For example, $a, b, a^1, c, d, b^3, c^2, d^2, a^5$. For each instruction, it has been shown [3] that the probability of a cache hit is lower bounded by:

$$P_{hit}(k) = \left(\frac{N-1}{N}\right)^k \tag{1}$$

provided that $k < N$, otherwise $P^{hit}(k) = 0$ (details of this latter restriction are given in [3]). Given fixed costs for the cache-hit latency (e.g. $H = 1$) and the cache miss latency (e.g. $M = 10$), then an upper bound pWCET distribution of a program can be computed as the convolution ($\otimes$) of the probability mass functions (PMFs) of each instruction. For example, given two instructions with PMFs with cache hit probabilities of 0.8 and 0.7 respectively, we get a pWCET distribution for the 'program' (comprising the two instructions) that has a probability of the execution time being 2 on any given run of 0.56, a probability that it will be 11 of 0.38, and a probability that there will be two cache misses and hence an execution time of 20 of 0.06.

$$\begin{pmatrix} 1 & 10 \\ 0.8 & 0.2 \end{pmatrix} \otimes \begin{pmatrix} 1 & 10 \\ 0.7 & 0.3 \end{pmatrix} = \begin{pmatrix} 2 & 11 & 20 \\ 0.56 & 0.38 & 0.06 \end{pmatrix}$$

We note that for larger numbers of instructions, the probability of a large number of cache misses quickly becomes vanishingly small, as illustrated by the graphs of 1-CDFs or exceedance functions in [3] one of which is reproduced in Figure 1.

We note that the SPTA given for evict-on-miss [2], [1] and evict-on-access [3] policies are somewhat *pessimistic*. This pessimism arises because when computing the probability of a hit for a particular instruction, the analysis assumes that all of the intervening instructions that could potentially be misses are in fact misses, which is a pessimistic assumption. For example, for the sequence *a, b,* a^1*, c, d,* b^3*,* c^2*,* d^2*,* a^5, when computing the probability of a hit for the third occurrence of '*a*' i.e. 'a^5', it is assumed that the five intervening instructions that could potentially be misses are all misses. The corresponding memory blocks are *c, d,* b^3*,* c^2*,* d^2; however, the probability that b^3*,* c^2*,* d^2 are all misses is actually very small, for example if N=256, then this probability is no greater than $7.1\text{x}10^{-7}$.

III. Optimistic Static Probabilistic Timing Analysis

In an attempt to remove the source of pessimism described above, an alternative formula is given in [5], for computing the probability of a cache hit as follows:

$$P_{hit} = \left(\frac{N-1}{N}\right)^{\sum P_{miss}} \tag{2}$$

where the summation in the exponent is over the probabilities of misses of the intervening instructions. No proof is given for this formulation [5]. Below, we show that it is optimistic. We first illustrate what led us to look very carefully at this formula: the fact that it can produce irrational numbers. This was suspicious given that probabilities must necessarily be rational, as in this case, each probability is computed by counting the number of scenarios that result in a particular outcome and then dividing by the total number of possible scenarios. As an example, we assume that $N = 2$ and the summation is over just one instruction that has a probability of being a hit of 1/2 (i.e. the second 'b' in the sequence *a, b,* a^1*,* b^1*),* hence we have:

$P_{hit} = \left(\frac{1}{2}\right)^{1/2} = 1/\sqrt{2}$, which is irrational.

Counter example:

We now show that (2) is optimistic. For simplicity we consider the same sequence of four instructions *a, b,* a^1*,* b^1, now with a cache size $N = 4$. Further, we assume that the latency of a cache hit is 1 and the latency of a cache miss is 10. In our example, the first two instructions are certain misses, so using (1), the probability distributions for the first three instructions are as follows: $\begin{pmatrix}10\\1\end{pmatrix}, \begin{pmatrix}10\\1\end{pmatrix}, \begin{pmatrix}1 & 10\\0.75 & 0.25\end{pmatrix}$ According to (2), the probability of the 4th instruction being a hit is then $\left(\frac{3}{4}\right)^{0.25} = 0.9307$

So the overall pWCET according to [5] is $\begin{pmatrix}10\\1\end{pmatrix}\otimes\begin{pmatrix}10\\1\end{pmatrix}\otimes\begin{pmatrix}1 & 10\\0.75 & 0.25\end{pmatrix}\otimes\begin{pmatrix}1 & 10\\0.9306 & 0.0694\end{pmatrix} = \begin{pmatrix}22 & 31 & 40\\0.69795 & 0.2847 & 0.01735\end{pmatrix}$

Now let us consider the only two possible scenarios separately and compute the exact pWCET distribution.

Case 1: the second '*a*' is a cache hit. This scenario has a probability of occurrence of 0.75 (as the first '*b*' is a certain cache miss and has a probability of 0.75 of *not* evicting '*a*' from the cache). Given that the second '*a*' is a cache hit, then the second '*b*' is also certain to be a cache hit, hence the partial pWCET for this scenario is $\begin{pmatrix}10\\1\end{pmatrix}\otimes\begin{pmatrix}10\\1\end{pmatrix}\otimes\begin{pmatrix}1\\0.75\end{pmatrix}\otimes\begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}22\\0.75\end{pmatrix}$

Case 2: the second '*a*' is a cache miss. This scenario has a probability of occurring of 0.25 (as the first '*b*' is a certain cache miss and has a probability of 0.25 of evicting '*a*' from the cache). Given the second '*a*' is a cache miss, the second '*b*' has a probability of 0.75 of being a cache hit, hence the partial pWCET is: $\begin{pmatrix}10\\1\end{pmatrix}\otimes\begin{pmatrix}10\\1\end{pmatrix}\otimes\begin{pmatrix}10\\0.25\end{pmatrix}\otimes\begin{pmatrix}1 & 10\\0.75 & 0.25\end{pmatrix} = \begin{pmatrix}31 & 40\\0.1875 & 0.0625\end{pmatrix}$

Combining Case 1 and Case 2, we have an overall pWCET of $\begin{pmatrix}22 & 31 & 40\\0.75 & 0.1875 & 0.0625\end{pmatrix}$

Notice that the exact pWCET distribution derived above by examining all possible scenarios is different from that obtained using the formula (2) from [5]. The precise calculation gives a higher probability of 0.0625 (versus 0.01735) of the absolute WCET of 40 occurring. Thus the formula from [5] does not deliver a valid upper bound pWCET distribution, instead it provides an optimistic pWCET that is unsafe to use. (Formally, we may say that a pWCET distribution (describing a random variable Y) is a valid upper bound on the exact pWCET distribution (describing a random variable Z) if $P\{\text{Y} \le x\} \le P\{\text{Z} \le x\}$ for any x).

We can also compute an upper bound pWCET for our example using the analysis given in [3] i.e. using (1) as follows: $\begin{pmatrix}10\\1\end{pmatrix}\otimes\begin{pmatrix}10\\1\end{pmatrix}\otimes\begin{pmatrix}1 & 10\\0.75 & 0.25\end{pmatrix}\otimes\begin{pmatrix}1 & 10\\0.75 & 0.25\end{pmatrix}=\begin{pmatrix}22 & 31 & 40\\0.5625 & 0.375 & 0.0625\end{pmatrix}$ Notice that in this case the pWCET obtained is not itself exact, but it is a valid upper bound on the exact distribution.

IV. Open Problem

In this short paper, we have refuted the SPTA formula for the probability of a cache hit given in [5] used to compute pWCET distributions. We have also shown that previous SPTA methods do not compute an exact pWCET distribution even for the simplest of programs. Computation of an exact pWCET distribution appears to require enumeration and composition of all the different possible scenarios. While this was possible for our simple example, in general it would lead to a combinatorial (exponential) number of cases to consider, hence rendering such an approach intractable even for moderately sized examples.

The open problem that we propose is therefore how to improve upon the simple SPTA analysis [2], [1], and [3] that exists today, so as to obtain tighter bounds on the actual pWCET distribution while keeping the amount of computation required within acceptable limits.

References

[1] F.J. Cazorla, E. Quinones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. “Proartis: Probabilistically analysable real-time systems”, ACM TECS, 2013.

[2] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. J. Cazorla. “Measurement-Based Probabilistic Timing Analysis for Multi-path Programs”. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), 2012.

[3] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. “Analysis of probabilistic cache related pre-emption delays”. In proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), 2013.

[4] J. Lopez, J. L. Diaz, J. Entrialgo, D. Garca. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. Real-time Systems, 40(2), 2008.

[5] L. Kosmidis, J. Abella, E. Quinones, F. J. Cazorla, “A Cache Design for Probabilistic Real-time Systems” In proceedings Design, Automation, and Test in Europe (DATE) 2013.

[6] E. Quinones, E. Berger, G. Bernat, and F. Cazorla. “Using Randomized Caches in Probabilistic Real-Time Systems”. In Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS), pages 129–138, 2009.