

Design and implementation of a multi-core embedded real-time operating system kernel

4th April 2014 - ACTRISS 2014 - OPRTC-ULB

Olivier Desenfans, Antonio Paolillo, Vladimir Svoboda, Ben Rodriguez, Joël Goossens, Dragomir Milojevic
PARTS Research Centre, Université Libre de Bruxelles and Mangogem S.A.
{odesenfa, antonio.paolillo, vladimir.svoboda, brodrigu, joel.goossens, dragomir.milojevic}@ulb.ac.be

I. THE SUBMITTED EXTENDED ABSTRACT

A. Introduction

Real-time theory has been focusing on the use of multi- and many-core chips in embedded systems for the past 2 decades [1]. While the subject has matured in the literature, the industry still widely relies on real-time operating system concepts created during the era of single processor platforms with simple interconnect and simple memory access architectures. Very few of these systems actually take advantage of multi-core systems and those that do rely on algorithms with proven poor performance [2]. *HIPPEROS*, standing for *HIgh Performance Parallel Embedded Real-time Operating Systems*, is a multi-core real-time operating system (RTOS) developed at the *PARTS* research centre. Its main objective is to address the challenge of exploiting modern multi-core platforms efficiently by filling this 20 year gap between theory and practice while maintaining the highest possible reliability in safety-critical systems. While this has been attempted in the past by patching various general purpose operating systems, *HIPPEROS* is designed and developed from scratch in a bottom-up rationale to avoid known issues due to legacy code. We adopted this radical and innovative approach willing to go as far as possible to test the limits of current research and to see how it can be implemented in a functional RTOS aiming to develop it into a commercial product. This extended abstract introduces the challenges encountered when developing such a real-time operating system on modern multi-core systems-on-chip. Moreover, it presents the *HIPPEROS* team research activities that will contribute to the state-of-the-art on a variety of topics such as: kernel architectures to provide real-time software support to applications, efficient implementations of new scheduling algorithms, techniques to improve the scalability of real-time applications on multi-/many-core platforms and scheduling algorithms aware of multi-dimensional constraints such as power and thermal metrics.

B. Open problems

This section explicitly presents three research questions addressed at the *PARTS* research centre: 1) *how should the real-time scheduling theory be applied in real-world embedded systems?* 2) *How well do these algorithms scale on many-core platforms?* 3) *How can we minimize the environmental footprint of embedded real-time systems?*

1) *Testing multi-core real-time scheduling algorithms in run-time embedded systems:* Researchers often turn towards Linux when it comes to testing new multi-core real-time algorithms [3], [4]. This approach has several merits such as the reuse of an existing operating system code base and a kernel that are already tested and validated by millions of users worldwide, but it also has one severe drawback: *Linux is not designed to handle hard real-time constraints*. As it is the case for the vast majority of software systems, a large effort has been made to optimise its *average-case* execution time while hard real-time systems have to provide determinism and repeatability for the *worst-case* behaviour. Moreover, Linux is monolithic and does not provide any guarantee of deadlock absence inside the complex layers of operating system mechanisms. This complexity of the Linux kernel makes the evaluation of the kernel-related overheads very difficult. This means that most of the new real-time multi-core scheduling algorithms have not been tested in a strict hard real-time environment yet. As stated by Brandenburg in [2]:

Ideally, [...], worst-case kernel overheads [...] should be determined analytically. However, for the foreseeable future, this will likely not be possible in complex kernels such as Linux. Instead, it would be beneficial to develop (or extend existing) μ -kernels of much simpler design with LITMUS^{RT}-like functionality.

Such a kernel would have to be built from the ground up with hard real-time and multi-core constraints integrated as parts of the base design. This will allow for simpler, finer-grained measurements of the overheads introduced by different implementations of the various scheduling policies the literature has to offer.

Furthermore, while popular scheduling policies have already been evaluated in a real run-time environment [2], the practical implementation of more sophisticated and powerful algorithms (e.g. RUN [5], U-EDF [6] or power- and thermally-aware algorithms) remains unexplored. As a consequence the applicability and the overheads related to these scheduling policies which present interesting theoretical algorithmic properties are unknown.

2) *The scalability issue*: The multi-core paradigm is not only interesting in terms of performance; it allows to reduce the actual number of individual uniprocessor boards in an embedded system by running functionally independent features on a single multi-core chip. However, this regrouping tends to drastically increase both the number of tasks and number of cores managed by the kernel. This has a large impact on kernel overheads. For example, the U-EDF scheduling algorithm, which is *theoretically* optimal i.e. introduces no algorithmic capacity loss, requires to execute a routine with a complexity proportional to the number of cores multiplied by the number of tasks *at each job arrival* [6].

Moreover, it is now well understood that the software architecture of multi-core kernels influences how these run-time overheads impact the overall system. A recent study [7] shows that the standard symmetric kernel architecture does not scale well on many-core platforms. The main solution devised in this study is to arbitrarily dedicate one of the cores to process all scheduling decisions. This core is called the *master core*. Other cores, called *slave cores*, are under control of the *master* and put its scheduling decisions to action.

It is possible to extend this approach to the whole kernel by introducing the notion of *remote system call*. Not only scheduling decisions but all system calls having a potential impact on other processes should be implemented using message passing. This is the approach we are taking in the design of the *HIPPEROS* kernel. The main challenge is to implement sound communication protocols between cores but removes the need for complex kernel locks and inter-core interactions which are the main causes behind poor scalability.

3) *Power and thermal constraints in high-end processors*: Power constraints are inherent to embedded systems, the most obvious reason being that most devices are powered from a limited power supply. Thus it is of interest to optimise the performance/power ratio. Most power reduction techniques for real-time systems use standard Dynamic Voltage and Frequency Scaling (DVFS) [8], [9] and scheduling algorithms tailored to reduce the average power consumption [10].

In high-end devices the temperature profile of the chip must also be taken into account. A high power consumption can induce high local peak temperatures inside the chip (i.e. hot spots). Such a rise in temperature can potentially damage the chip if no action is taken. While this is a non-issue on most single core embedded systems today, as the number of cores rises this will lead to the issue of dark silicon [11]: a significant portion of a chip (i.e. number of cores, cache sections [12]) has to be powered off at any time to act as a heat sink for the other processing cores. On general purpose devices (e.g. smartphones), this issue is addressed by using *race to idle* strategies. Cores are to enter a deep sleep power mode after executing their attributed workload at maximum speed. However this introduces long wake-up latencies and can strongly contribute to the Worst-Case Execution Time of all tasks. Recent research [13], [14] provide more nuanced solutions.

Once again, we wish to implement and evaluate these solutions in a real system. These techniques may induce potentially large overheads and latencies which may be impractical in a hard real-time context.

II. THE TALK

This section presents information about the talk at the ACTRISS day.

This url will lead you to the slideset:

- <http://bit.ly/1ebUcs5>.

You can also use the QR Code given on Figure 1.

Here are some additional references used during the preparation of this presentation:

- EDF implementations: [15], [16], [17], [18]
- Scaling global scheduling: [7]
- U-EDF, P-fair: [6]
- Dark silicon: [11]



Fig. 1: URL of the presentation.



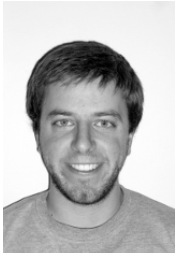



<p>Parallel Architectures for Real-Time Systems</p> <p>MangoGem S.A.</p> <p>Antonio Paolillo</p>	  	<p>http://parts.ulb.ac.be/</p> <p>http://www.mangogem.com/</p> <p>http://antonio.paolillo.be/</p>	  
--	---	--	---

Fig. 2: Research group and speaker information.

REFERENCES

- [1] J. Parkhurst, J. Darringer, and B. Grundmann, "From single core to multi-core: preparing for a new exponential," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 2006, pp. 67–72.
- [2] B. Brandenburg, "Scheduling and locking in multiprocessor real-time operating systems," Ph.D. dissertation, The University of North Carolina, 2011.
- [3] J. M. Calandrino, H. Leontyev, A. Block, U. Devi, and J. H. Anderson, "Litmus^{RT}: A testbed for empirically comparing real-time multiprocessor schedulers," in *27th IEEE International Real-Time Systems Symposium*. IEEE, 2006, pp. 111–126.
- [4] D. Faggioli, M. Trimarchi, F. Checconi, M. Bertogna, and A. Mancina, "An implementation of the earliest deadline first algorithm in Linux," in *24th Annual ACM symposium on Applied Computing*. ACM, 2009, pp. 1984–1989.
- [5] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "RUN: Optimal multiprocessor real-time scheduling via reduction to uniprocessor," in *IEEE 32nd Real-Time Systems Symposium*, Nov 2011, pp. 104–115.
- [6] G. Nelissen, V. Berten, V. Nelis, J. Goossens, and D. Milojevic, "U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks," in *24th Euromicro Conference on Real-Time Systems*, July 2012, pp. 13–23.
- [7] F. Cerqueira, M. Vanga, and B. Brandenburg, "Scaling global scheduling with message passing," in *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2014.
- [8] V. Nelis, J. Goossens, R. Devillers, and D. Milojevic, "Power-aware real-time scheduling upon identical multiprocessor platforms," in *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, 2008.
- [9] V. Hanumaiah and S. Vrudhula, "Temperature-aware DVFS for hard real-time applications on multicore processors," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1484–1494, October 2012.
- [10] D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 7, pp. 686–700, July 2003.
- [11] H. Esmailzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th International Symposium on Computer Architecture*, 2011.
- [12] X. Fu, K. Kabir, and X. Wang, "Cache-aware utilization control for energy efficiency in multi-core real-time systems," in *23rd Euromicro Conference on Real-Time Systems*. IEEE, 2011, pp. 102–111.
- [13] N. Fisher, J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling on multicore systems," in *Real-Time and Embedded Technology and Applications Symposium*, April 2009.
- [14] L. Thiele, L. Schor, I. Bacivarov, and H. Yang, "Predictability for timing and temperature in multiprocessor system-on-chip platforms," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 48, March 2013.
- [15] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973. [Online]. Available: <http://doi.acm.org/10.1145/321738.321743>
- [16] G. C. Buttazzo, "Rate monotonic vs. edf: Judgment day," *Real-Time Syst.*, vol. 29, no. 1, pp. 5–26, Jan. 2005. [Online]. Available: <http://dx.doi.org/10.1023/B:TIME.0000048932.30002.d9>
- [17] G. Buttazzo and P. Gai, "Efficient implementation of an edf scheduler for small embedded systems," *Proc. OSPERT*, 2006.
- [18] M. Short, "Improved task management techniques for enforcing edf scheduling on recurring tasks," in *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, ser. RTAS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 56–65. [Online]. Available: <http://dx.doi.org/10.1109/RTAS.2010.22>