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 task model is an important step towards formally defining a task model for parallel real-time tasks [8]. 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.

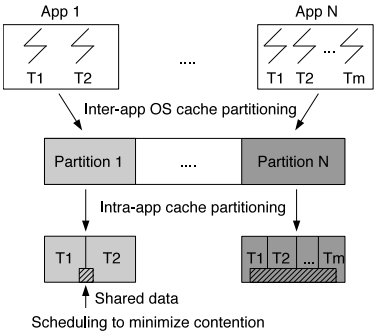

Figure 1. Proposed envisioned solution architecture.

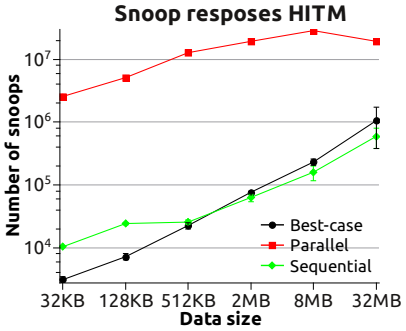

Figure 2. HPCs evaluation on Intel Q9550 processor: snoops HITM.

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 [9].

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] 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 Syst.*, under review.