Impact of Resource Sharing on Performance and Performance Prediction

(Invited Paper)

Jan Reineke and Reinhard Wilhelm
FR Informatik
Universität des Saarlandes
Saarbrücken, Germany
Email: {reineke/wilhelm}@cs.uni-saarland.de

EXTENDED ABSTRACT

Multi-core processors are increasingly considered as execution platforms for embedded systems because of their good performance/energy ratio. Many applications implemented on multi-core platforms are safety- and some also time-critical. A critical issue for these applications is the reduced predictability of such systems resulting from the interference of different applications on shared resources. These interferences can be at least of two kinds: Several applications may request a resource at the same time, but the resource can only admit one access at a time. As a consequence, an arbitration mechanism may delay the request of all but one application, thus slowing down the other applications. This is the case of resources like buses, typically called *bandwidth resources*. On the other hand, one application may also change the state of a shared resource such that another application using that resource will suffer from a slowdown. This is the case with shared memories, such as shared caches and shared dynamic random-access memories, which fall into the class of *storage resources*.

Interference on shared resources makes worst-case execution time (WCET) analysis of applications more difficult since a task or a thread can no longer be analyzed for its timing behavior in isolation. All potential interferences slowing down (or speeding up) the task under analysis have to be considered. This leads to a combinatorial explosion of the analysis complexity, as all possible interleavings of different threads have to be analyzed.

The survey [1] considers several aspects of the execution of sets of tasks on multi-core platforms that have to do with the interference of the tasks on shared resources. One question is how the actual performance of tasks is slowed down by other co-running tasks. Another is how to compute bounds on the slow-down in order to derive sound guarantees for the timing behavior. A major problem is the increased complexity of this task compared to the single-task single-core case. This has led to the situation that industry is developing embedded systems for multi-core platforms while there exist no timing-analysis methods and tools that are both sound and precise.

A. Storage resources

Caches are a particular case of *storage resources*. Several approaches exist for the treatment of shared caches in attempts to derive timing guarantees. Cache partitioning eliminates the interference between tasks. Static analysis of non-partitioned shared caches attempts to safely bound the interference. The definite comparison between these two approaches has yet to be done.

B. Bandwidth resources

Buses are instances of *bandwidth resources*. Several protocols exist for the arbitration of shared buses, which can be classified as either time-driven, event-driven, or hybrid combinations of both. Static analysis can be used to determine good slot assignments in time-driven protocols like TDMA, and it can be used to determine bounds on the access delays in event-driven state-based protocols like FCFS and round robin.

C. Ways to derive guarantees

In order to guarantee the timeliness of tasks in a hard real-time system, one needs upper bounds on the execution times of the tasks. As long as a task executes in isolation, i.e., without (interference on) shared resources, on a multi-core system, existing techniques for timing estimation can be applied. In case of an execution with co-running tasks using shared resources, a sound approach for timing analysis has to take into account the interferences on the shared resources.

Approaches to determine upper bounds on execution times of tasks on multi-core platforms can be classified into two groups:

- Approaches achieving *performance isolation* by eliminating interference using hardware and/or software techniques. Performance isolation implies *timing composability* and permits the use of standard single-core timing analysis techniques with minor modifications.
- Approaches analyzing the mutual effects of co-running tasks on each others execution time. Such approaches require new timing analysis techniques that differ greatly from those employed in the single-core single-task case.

a) Achieving Performance Isolation: In single-core architectures, integrated modular avionics (IMA) according to ARINC 653 [3] attempts to achieve performance isolation by *temporal and spatial partitioning* [4]. Spatial partitioning protects the memory space (both code and data) of one application

978-3-9815370-2-4/DATE14/©2014 EDAA

against access by another application. It is realized by avoiding sharing of memory resources. Temporal partitioning has to ensure that the activities in one (activation of an) application do not affect the timing of the activities in (an activation of) another application. It is implemented by static periodic scheduling and by flushing resources, such as caches, between invocations of different tasks.

However, such a partitioning solution is inefficient and cannot easily be extended to multi-core architectures, in which different tasks execute concurrently. In such cases, hardware support for *resource partitioning* is required. Examples are TDMA arbitration of busses and partitioning of caches. In addition, not all shared resources can (easily) be flushed.

The challenge in these approaches is to make efficient use of shared resources by partitioning them appropriately for the given workload. In the survey [1], we review software- and hardware-based mechanisms to partition shared caches and to determine good partition sizes.

b) Analyzing the Impact of Interference: Different methods have been proposed or are pursued to derive guarantees for the timeliness of sets of tasks in a parallel workload setting when performance isolation is not given. First, there is the classification according to whether the software is analyzed or executed.

- The *static analysis* of a whole set of concurrently executed applications may deliver a sound and precise guarantee for the timing behavior. The problem is the huge complexity of this approach.
- *Measurement-based* methods are in general not able to derive guarantees, neither in the single-core nor in the multi-core case.

The particular contribution to the execution-time bounds of the interference on shared resources can be dealt with in different ways:

1. The *Murphy* approach assumes maximal interference on each access to a shared resource. This assumption can be easily integrated into existing single-core timing analysis techniques. The *Murphy* approach will clearly give sound, but the most pessimistic execution-time bounds.

2. The *slowdown factor* approach attempts to explicitly quantify the worst-case impact of the interferences on shared resources on the timing of a task caused by co-running tasks. The obtained slowdown factor could then be used to obtain an estimate on the execution times of a task in a parallel workload from an estimate in the isolated case. Existing approaches aim at quantifying the slowdown of a task in the worst case by measurement-based techniques. These measurement-based approaches employ so-called *resource-stressing benchmarks*, which are constructed for particular resources to produce the maximal slowdown on co-running tasks due to conflicts on this resource [6]. In the survey [1], we claim that attempts in this direction may be both unsound and overly pessimistic. Resource-stressing benchmarks are, in general, independent of the application that is slowed down by co-running these benchmarks. Therefore, one single resource-stressing benchmark can hardly slow down the application in the worst way. The fact that resource-stressing benchmarks might not be sound is demonstrated by an imaginary application-specific *worst companion*.

3. Finally, *static analysis* of a whole set of concurrently executed applications may deliver a sound and precise guarantee for the timing behavior. The problem is the huge complexity of this approach. To reduce analysis complexity, existing static analysis approaches separate analysis into two phases: The first phase determines a bound on the execution time of each task in isolation and a characterization of its resource-access behavior. The second analysis phase then uses this characterization to bound the impact of interference on the execution times of all tasks. The sum of the two bounds for each task then yields an estimate of the task's worst-case execution time. For soundness, all of these approaches rely on *timing compositionality* [5], which permits to account for the cost of interference in such a compositional way. Unfortunately, many existing hardware architectures exhibit domino effects and timing anomalies and are thus out of the scope of such an approach.

Different abstraction levels for characterizing the resource behavior may yield different precision of the result and may require different analysis effort. One popular abstraction is the *superblock task model*, where tasks are structured as a sequence of superblocks for which bounds on the execution time and on the resource behavior are known. A major problem with this approach is that the resource behavior, e.g. bus accesses for cache reloads, may depend on the interference of the tasks on the shared cache (if this is not partitioned), which depends on the relative speed of the different tasks, which again depends on the cache performance.

ACKNOWLEDGMENT

The authors would like to thank Andreas Abel, Florian Benz, Johannes Dörfert, Barbara Dörr, Sebastian Hahn, Florian Haupenthal, Michael Jacobs, Amir H. Moin, Bernhard Schommer, our coauthors on the survey [1].

Work reported herein has been supported by the Transregional Research Center AVACS of the Deutsche Forschungsgemeinschaft.

REFERENCES

[1] Andreas Abel, Florian Benz, Johannes Doerfert, Barbara Dörr, Sebastian Hahn, Florian Haupenthal, Michael Jacobs, Amir H. Moin, Jan Reineke, Bernhard Schommer, and Reinhard Wilhelm. Impact of resource sharing on performance and performance prediction: A survey. In D'Argenio and Melgratti [2], pages 25–43.

[2] Pedro R. D'Argenio, Hernán C. Melgratti, editors. *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8052 of *Lecture Notes in Computer Science*. Springer, 2013.

[3] Airlines electronic engineering committee (AEEC), Design Guidance for Integrated Modular Avionics (ARINC specification 651). ARINC, Inc., 1991.

[4] John Rushby. Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance SRI International, Menlo Park, 2000.

[5] Sebastian Hahn, Jan Reineke, Reinhard Wilhelm. Towards Compositionality in Execution Time Analysis - Definition and Challenges. CRTS 2013, Vancouver, Canada, December 3, 2013.

[6] Radojkovic, P., Girbal, S., Grasset, A., Quinones, E., Yehia, S., Cazorla, F.J. On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments. ACM Trans. Archit. Code Optim. (2012)