# Impact of Resource Sharing on Performance and Performance Prediction: A Survey

Andreas Abel, Florian Benz, Johannes Doerfert, Barbara Dörr, Sebastian Hahn, Florian Haupenthal, Michael Jacobs, Amir H. Moin, Jan Reineke, Bernhard Schommer, and Reinhard Wilhelm

Saarland University, Saarbrücken, Germany

**Abstract** Multi-core processors are increasingly considered as execution platforms for embedded systems because of their good performance/energy ratio. However, the interference on shared resources poses several problems. It may severely reduce the performance of tasks executed on the cores, and it increases the complexity of timing analysis and/or decreases the precision of its results. In this paper, we survey recent work on the impact of shared buses, caches, and other resources on performance and performance prediction.

## 1 Introduction

Multi-core processors are increasingly considered as execution platforms for embedded systems since they offer a good energy-performance tradeoff and seem to support transitions from federated to integrated system architectures in the automotive and avionics domains. Many applications implemented on such multi-core platforms are safety- and some also time-critical. A critical issue 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 caches, which fall into the class of *storage resources*. Most of the treatments of the interferences on shared resources found in the literature consider the detrimental effect of interferences. In the case of shared caches, however, the interference of one application $A_1$ on another co-running application $A_2$ could even speed up $A_2$ if $A_1$ would perform the right cache prefetching for $A_2$.

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. For that reason, currently, no sound timing-analysis method for multi-core platforms with shared resources exists.

P.R. D'Argenio and H. Melgratti (Eds.): CONCUR 2013, LNCS 8052, pp. 25–43, 2013.

This survey considers several aspects of the execution of sets of tasks on multi-core platform 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. The other is how to compute bounds on the slow-down in order to derive guarantees for the timing behavior. A major problem is the increased complexity of this task compared to the single-task single-core case.

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.

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.

### 1.1 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 on a multi-core system (without co-running tasks), existing techniques for timing estimation could be applied. In case of parallel workloads (with co-running tasks), a sound approach for timing analysis of multi-core systems has to take into account the interferences, as described in detail in Sections 2 and 3.

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

- Approaches achieving *performance isolation* by hardware and/or software techniques, e.g. by employing TDMA arbitration of busses. Performance isolation implies *timing composability* and permits the use of standard single-core timing analysis techniques with minor modifications. While this makes timing analysis comparatively easy, the challenge in such approaches is to make efficient use of shared resources by partitioning them appropriately for the given workload.
- Approaches analyzing the mutual effects of co-running tasks on each other's execution time. Such approaches require new timing analysis techniques that differ greatly from those employed in the single-core single-task case.

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.