Embedded Systems: Many Cores – Many Problems

(Invited Paper)

Reinhard Wilhelm and Jan Reineke
Informatik
Saarland University
D-66123 Saarbrücken
Email: {wilhelm | reineke}@cs.uni-saarland.de

Abstract

The embedded-systems industry is about to make a transition to multi-core platforms. This is a highly risky step as several essential problems are not yet solved. In particular, the performance analysis problem has no viable solution for the existing multi-core designs. The main culprit is the interference on shared resources. Several alternative approaches both in architecture and system design and in analysis methods are discussed.

1. Introduction

The embedded-systems industry introduces multi-core architectures for their good performance-energy ratio. This trend coincides with the transition from *federated* to *integrated system architectures* in the automotive and the aeronautics industries. In federated architectures, many computers—most of the time one per application—are used, connected by buses. In integrated architectures, such as the *Integrated Modular Avionics* (IMA) [1] and the *AUTomotive Open System ARchitecture* (AUTOSAR)[1], many applications—often of mixed criticality— are integrated on powerful platforms. This is expected to improve systems in several dimensions, weight, space, energy consumption, and maintainability. *Composability* of the resource behavior is the main design goal as it will support *incremental cerification*.

Multi-core platforms seem to be ideally suited for these integrated software architectures. However, the transition to multi-core platforms for embedded system is problematic. Many problems are still awaiting their solutions. Integrated software architectures on top of multi-core platforms introduce single-point-of-failures for possibly a whole set of application instances and therefore require new redundancy concepts.

Mapping a set of (instances of) applications to a set of cores of several multi-core platforms satisfying all performance and redundancy requirements is a highly complex problem, which currently does not seem to have found a satisfactory solution.

As a last point, the question of how to derive performance guarantees for embedded system implemented on multi-core platforms is unsolved.

1. http://www.autosar.org

2. The Timing-Analysis Problem

We concentrate on the timing-analysis problem.

2.1. Single-Core Systems

Sound guarantees for the timing behavior of threads are needed as soon as some of the applications to be integrated on a multi-core platform are time-critical. *Total task isolation*, also with respect to resource consumption, is the method currently used to exclude the influence of non-critical applications on critical applications. it is implemented using *temporal* and *spatial partitioning*.

The timing-analysis problem for uninterrupted execution on single-core platforms has long been solved [2]. The necessary analyses for the determination of preemption costs are available [3]. The effort needed by sound static timing-analysis methods for single core platforms is large, but tolerable. Extending the methods to multiple threads executed on several cores will either lead to an unacceptable increase in the analysis effort or to an unacceptable decrease in precision of the results.

Static analysis methods based on abstract interpretation have the, in principle positive, property that they allow to trade efficiency of the analyses for precision of the results. But trying to reduce the analysis effort this way might possibly reduce the achievable precision to unacceptable degrees.

2.2. Interference on Shared Resources

The challenge is caused by interferences on resources shared between several threads running on different cores. We observe two kinds of interferences: Inherent interferences and virtual interferences.

Inherent interferences on a shared resource can actually be observed in a run of the system. Such interferences might increase the actual execution times of tasks and therefore, inherently, the WCET bounds of those tasks, too.

Virtual interferences are introduced by abstraction of the system, i.e., loss of information about the system. Although an interference might never happen in a concrete run of the system, the analysis may not be able to prove this, as it can only rely on its incomplete, static information. For instance, if the timing analysis for task T completely abstracts from concurrently running tasks, it has to assume an interference by another task T' every time T makes an access to a shared resource. This information loss is caused by a (total) abstraction from the set of tasks and the system's task scheduling policies, which restrict what can actually happen concurrently. It is an open problem how to limit the information loss about concurrently running tasks by suitable abstractions. Hence, limiting inherent interferences must be a high-priority design goal: If there can be no interferences *at all* in the concrete system, it is easy for an analysis to exclude interferences even when abstracting completely from other tasks. A rather radical proposal is presented in Section 3.4.

2.2.1. Observed and Worst-Case Interference. The inherent interference on shared resources reduces the performance of tasks running on the individual cores. A number of experiments have tried to find out what the inherent interference of concurrently executed benchmark programs or specially constructed worst-case programs can be. The former experiments choose a set of programs, measure their performance when executed on one core in isolation, then run them in combination on a number of cores to determine the slowdown compared to the single-core execution. [4] describes a Thales case study where several benchmark programs are run concurrently on two cores. A worst slowdown of 60% was observed when the same program was run on the two cores. In such a scenario, the accesses of the two programs to shared resources are perfectly synchronized, and thus maximally interfere.

For the second class of experiments, programs are designed that stress a particular shared resource to determine the worst-case interference on that resource. Nowotsch and Paulitsch [5] report a slowdown by a factor of 20 on an 8-core platform, when threads are concurrently run that maximally stress a shared DDR SDRAM. Radojkovic et al. [6] have constructed a set of benchmark programs, one for each resource, maximally stressing this resource on a given platform. The program under analysis is then run on one core, the resource-stressing programs each on another core. The authors claim that this produces an upper bound on the slowdown that the program under analysis will suffer.

The latter experiments exhibit an amount of inherent interference that programs not designed to stress resources will rarely show. On the other hand, efficient static analysis methods will use abstraction to arrive at acceptable effort. Abstraction in general introduces imprecision in the form of virtual interference. So, on the one hand static analyses of real-life programs will most likely not exhibit the worst-case inherent interference. On the other hand, they will produce upper bounds that may be quite pessimistic. The goal is to arrive at efficient analyses with high precision.

2.2.2. Increased Analysis Complexity. State-of-the-art analyses integrate the analysis of all architectural features, i.e., the pipeline, bus, and cache state, precisely tracking their interactions. For caches, compact, yet precise, abstract domains exist: sets of concrete cache states can be efficiently represented by abstract cache states, which can be joined where control flow joins, and which can be precisely and efficiently updated upon memory accesses. Unfortunately, such abstract domains have not been found for pipeline states. As a consequence, current timing analyses need to maintain large sets of pipeline states and–due to timing anomalies [7], [8]–they need to follow all possible cases, whenever several successor states are possible. Even for complex single-core processors this makes timing analysis expensive, yet still at an acceptable level.

The increased analysis complexity for inadequate multi-core designs—unfortunately all existing designs— results from the interference on shared resources of different threads executed on different cores. These threads, if running asynchronously, may access the shared resources in many different interleavings,

possibly resulting in different execution states and in different timing behaviors. The number of different interleavings for a frequently accessed resource such as a shared asynchronous bus is larger than what can be exhaustively explored. Two threads that are run on two cores with n and m accesses to a shared resource allow for $(n+m)!/(n! \cdot m!)$ different interleavings. When proving the correctness of concurrent programs, global variables are the shared resources, and n and m are typically rather small. Still, most attempts to do the correctness proof encounter a severe state-space explosion problem. In the case of timing analysis for threads running on multi-core platforms with shared resources such as buses, n and m are huge!

So, either interference needs to be eliminated, by eliminating shared resources as discussed in Section 3, or a fundamentally different analysis approach is required.

2.2.3. Improved Analysis Methods. One angle to attack the above problem is to reduce the set of interleavings that have to be considered by analysis. The model-checking community has invented ingenious reduction techniques. The most adequate seems to be partial-order reduction [9]. It would reduce the state space by exploiting the commutativity between concurrently executed instructions of the different threads. However, different interleavings of instructions need not lead to the same execution states and need not have the same timing. An efficiently computable equivalence relation on interleavings needs to be developed that provides for an architecture-specific reduction technique.

Another, more radical departure from current approaches, would be *compositional timing analysis*: instead of analyzing the contributions of all architectural components in an integrated fashion, one would separately analyze the contributions of various components to the overall execution time, such as the pipeline, buses, and caches, summing them up in the end.

This would dramatically reduce analysis time, as such analyses would not have to deal with the product of the state spaces of the various components. In particular, the contribution of interference on shared resources could be calculated separately. Approaches to bound cache-related preemption cost [3] are already pursuing this direction.

Sometimes, "timing accidents" in different components, such as pipeline stalls and cache misses overlap, causing overall execution time to be smaller than the sum of the timing penalties incurred in the different architectural components. Compositional analyses will lose precision compared with integrated analyses as, by design, they cannot detect such scenarios. On the other hand, compositional analyses can gain precision: by focussing on individual architectural components, much more precise abstractions can be employed that may exclude timing accidents that a stronger abstraction in an integrated analysis would have to consider possible.

3. Taking Constructive Influence

Several remedies have been proposed. A rather radical proposal ends this section.

3.1. Smart Configuration of Existing Multi-Cores

Currently available multi-cores were not developed with WCET analysis in mind. Consequently, they exhibit timing anomalies, poorly analyzable cache replacement policies, or fully shared memory. Leaving all those average-case performance-enhancing features enabled renders static timing analysis almost inapplicable.

[4], [10] show how to configure multi-core processors in a way such that static timing analysis is made easier. They do this for the MPC5668G, an automotive processor, and the MPC8641D, an avionics processor.

3.2. Deterministic Access Protocols

The number of interleavings and the resulting uncertainty about the timing behavior can be reduced by introducing deterministic access protocols for the shared resources [11] and then safely bounding the access delay to the shared resources [12], [13], [14], [15]. The disadvantage may be the encountered performance loss. While the gain in predictability is clear, the ultimate analysis of the expected performance loss is still missing.

Deterministic access protocols can be computed for the accesses to shared resources from the access patterns to these shared resources [12] . This deterministic protocol will allow to control the worst-case length of an access delay and to derive safe and precise bounds on the overall execution times.

Cumulative approaches use upper bounds on the resource consumption by interfering tasks to determine safe bounds on the access delays [16].

Resource access arbitration with bounded delays use arbitration schemes that guarantee bounds on the delays for the accesses to shared resources. The derivation of such bounds depends on the arbitration protocol of the bus used to access shared resources. In [15] such

an arbitration scheme is complemented with a resource front-end to completely isolate the temporal behavior of tasks accessing a predictable shared resource.

3.3. Resource Partitioning

Deterministic bus protocols such as TDMA partition the bus bandwidth allocating slots to different threads. Similarly, space resources can be partitioned and parts exclusively allocated to different threads to eliminate interferences. This has been done in particular for 2nd-level caches by cache partitioning, i.e., by allocating disjoint partitions of a shared cache to different threads running on different cores [17]. Partitioning of a shared L2 cache does, however, not solve the interference problem if accesses to the cache or to memory still may collide on a shared bus. In [18], temporal and spatial resource partitioning are combined to eliminate any interference between different threads accessing a shared DDR2 SDRAM device: each DRAM bank is privately allocated to an individual hardware thread, and access to the banks is granted in a time-triggered fashion.

3.4. The PROMPT Approach

The more radical alternative is to abolish resource sharing as far as possible [19], [20].

Design Principles. The PROMPT (PRedictability Of Multi-Processor Timing) architecture design principles, see [19], aim at embedded hard real-time systems in the automotive and the aeronautics industry requiring *efficiently predictable good worst-case performance*.

The small amount of sharing existing in the set of applications allows to design a target architecture with little interference on shared resources and thus little variance of execution times and high predictability. Our principle is *Architecture follows Application*. The goal of this design discipline is to *improve the worst-case performance* and to *make the derivation of reliable and precise timing guarantees efficiently feasible*. This design discipline will support the composability goal of the IMA and AUTOSAR movements in the aeronautics and the automotive industries. We conjecture that without this or a similar design discipline the required modular development process will not be realizable without an unacceptable loss of guaranteed performance.

The architecture is designed in a multi-phase process. It starts with the design or the selection of fully timing-compositional cores as discussed in [21].

Then the set of applications is considered:

- *Hierarchical privatization* will decompose the set of applications according to their sharing characteristics on the shared global state. The resulting partitioning of the set of applications could be used to define an isomorphically structured target architecture with no more shared resources than required by the set of applications.
- *Sharing of lonely resources* would introduce sharing of costly and infrequently accessed resources. Input/output devices will most likely have to be shared, for cost and space reasons.
- *Controlled socialization* would try to satisfy cost constraints with an acceptable loss of predictability. It would introduce sharing while controlling the loss in predictability.

Ways to determine safe and sufficiently small delays for the access to shared resources have been discussed in Section 3.2.

Conclusion

The timing-analysis problem for embedded systems running on multi-core platforms is still unsolved. Architectures with better timing predictability, most likely with fewer shared resources or more deterministic access control are needed.

Acknowledgment

The authors would like to thank the members of the chair and long-time collaborators at AbsInt for valuable contributions and discussions.

References

[1] C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to integrated modular avionics," in *26th AIAA/IEEE Digital Avionics Systems Conference (DASC)*, 2007.

[2] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm, "Reliable and precise WCET determination for a real-life processor," in *EMSOFT*, ser. LNCS, vol. 2211, 2001, pp. 469–485.

[3] S. Altmeyer, C. Maiza, and J. Reineke, "Resilience analysis: Tightening the CRPD bound for set-associative caches," in *LCTES '10: Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems*. New York, NY, USA: ACM, April 2010, pp. 153–162. [Online]. Available: http://rw4.cs.uni-saarland.de/~reineke/publications/ResilienceAnalysisLCTES10.pdf

[4] D. Kästner, M. Schlickling, M. Pister, C. Cullmann, G. Gebhard, C. Ferdinand, and R. Heckmann, "Timing predictability of multi-core processors," 2012, submitted for publication.

[5] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," in *EDCC*, 2012.

[6] P. Radojkovic, S. Girbal, A. Grasset, E. Quiñones, S. Yehia, and F. J. Cazorla, "On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments," *TACO*, vol. 8, no. 4, p. 34, 2012.

[7] T. Lundqvist and P. Stenström, "Timing anomalies in dynamically scheduled microprocessors," in *RTSS '99: Proceedings of the 20th IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1999, p. 12.

[8] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker, "A definition and classification of timing anomalies," in *Proceedings of 6th International Workshop on Worst-Case Execution Time (WCET) Analysis*, July 2006.

[9] A. Valmari, "A stubborn attack on state explosion," in *CAV*, ser. Lecture Notes in Computer Science, E. M. Clarke and R. P. Kurshan, Eds., vol. 531. Springer, 1990, pp. 156–165.

[10] D. Kästner, M. Schlickling, M. Pister, C. Cullmann, G. Gebhard, R. Heckmann, and C. Ferdinand, "Timing Predictability of Multi-Core Processors," *Embedded World Congress*, 2012.

[11] H. Kopetz, "On the design of distributed time-triggered embedded systems," *JCSE*, vol. 2, no. 4, pp. 340–356, 2008.

[12] J. Rosen, A. Andrei, P. Eles, and Z. Peng, "Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, ser. RTSS '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 49–60. [Online]. Available: http://dx.doi.org/10.1109/RTSS.2007.13

[13] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, "Worst case delay analysis for memory interference in multicore systems," in *DATE*. IEEE, 2010, pp. 741–746.

[14] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo, "Worst-case response time analysis of resource access models in multi-core systems," in *DAC*, S. S. Sapatnekar, Ed. ACM, 2010, pp. 332–337.

[15] B. Akesson, A. Hansson, and K. Goossens, "Composable resource sharing based on latency-rate servers," in *Euromicro Symposium on Digital Systems Design*. IEEE, August 2009, pp. 547–555.

[16] R. Pellizzoni and M. Caccamo, "Toward the predictable integration of real-time COTS based systems," in *Real-Time Systems Symposium (RTSS)*, 2007.

[17] N. Guan, M. Stigge, W. Yi, and G. Yu, "Cache-aware scheduling and analysis for multicores," in *Proceedings of the seventh ACM international conference on Embedded software*, ser. EMSOFT '09. New York, NY, USA: ACM, 2009, pp. 245–254. [Online]. Available: http://doi.acm.org/10.1145/1629335.1629369

[18] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee, "PRET DRAM controller: Bank privatization for predictability and temporal isolation," in *CODES+ISSS '11: Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, October 2011, pp. 99–108. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/851.html

[19] R. Wilhelm, C. Ferdinand, C. Cullmann, D. Grund, J. Reineke, and B. Triquet, "Designing predictable multicore architectures for avionics and automotive systems," in *Workshop on Reconciling Performance with Predictability (RePP)*, 2009.

[20] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, S. Wegener, and R. Wilhelm, "Predictability considerations in the design of multi-core embedded systems," *Ingénieurs de l'Automobile*, vol. 807, pp. 36–42, September 2010.

[21] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand, "Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 28, no. 7, pp. 966–978, 2009.