Efficient Execution of Mixed Application Workloads in a Hard Real-Time Multicore System

Marco Paolieri[1,2], Eduardo Quiñones[1], Francisco J. Cazorla[1], and Mateo Valero[1,2]

[1] Barcelona Supercomputing Center,
Barcelona, Spain,
marco.paolieri@bsc.es
[2] DAC - Universitat Politècnica de Catalunya,
Barcelona, Spain

Abstract. In this paper we present a multicore architecture that introduces a novel hardware shared-resource management policy, called *Worst-Case Resource Management* (WC-RM), that allows executing efficiently mixed application workloads composed by hard real-time and non real-time applications in a multicore platform. Our multicore architecture forces hard real-time tasks to be executed close to their worst-case execution time, leaving more free shared resources that can be used by the non real-time tasks. Our WC-RM policy improves the performance of NHRTs up to 10% compared to a resource management policy in which hard real-time tasks access the shared resources as soon as they are available.

1 Introduction

Nowadays, hard real-time embedded systems require more performance than what are currently provided in embedded processors[3]. Being able to achieve the desired performance will allow hard real-time embedded systems to increase safety, comfort and, number and quality of services.

Multicore processors are increasingly being considered as an effective solution due to their low cost and good performance per watt ratio, while maintaining a relative simple processor design without suffering timing anomalies [2]. Moreover, multicore processors ideally enable co-hosting mixed application workload composed by applications with different requirements (e.g. high data processing demand or stringent time constraints). Co-hosting non-safety and safety critical applications on a common powerful multicore processor is extremely beneficial for the embedded system market, because it allows scheduling a higher number of tasks on a single processor, maximizing the hardware utilization while reducing cost, size, weight and power requirements.

However, even if multicore processors may offer many benefits to the embedded system designers, it is much harder to perform timing analysis for multicore

processors than for single-core processors due to *inter-thread interferences* accessing hardware shared resources[3], and providing a tight *Worst-Case Execution Time* (WCET) analysis is of paramount importance for hard real-time systems.

Inter-thread interferences appear when two or more tasks that share a resource try to access it at the same time, so that an arbitration mechanism is required. As a consequence, the execution time of a task, and so its WCET, may increase depending on the other tasks running simultaneously inside the processor.

In order to consider inter-thread interferences when performing the timing analysis of *Hard Real-time Tasks* (HRTs), we have recently proposed a multicore architecture [1] in which the maximum delay that a request to a shared resource can suffer due to inter-thread interferences is bounded. That is, our multicore processor *guarantees* by design that a request from a HRT cannot be delayed longer than a given *Upper Bound Delay* (UBD).

However, under this scenario, *Non Hard Real-time Task* (NHRTs) can suffer severe performance degradations because HRTs have higher priority when accessing shared resources to cope with their stringent timing constraints, i.e. their deadlines. As a consequence HRTs can keep accessing shared resources preventing NHRTs from accessing them, and, in order to execute efficiently mixed application workloads composed by safety and non-safety critical applications in a multicore processor it is not only required to ensure that HRTs meet their deadlines; but it is also important not to degrade the performance of NHRTs.

To this end, in this paper we propose a novel shared-resource management policy, the *Worst-Case Resource Management* (WC-RM) policy, in which HRTs are forced to be executed close to their WCET by delaying each shared resource access by UBD cycles. By doing this, more *free* shared resources are then available for NHRTs, and so their performance may not be degraded. In order to show the effectiveness of our technique, we compare our WC-RM policy with a resource management policy, called *Average-Case Resource Management* (AC-RM), in which HRTs access the shared resources as soon as they are available. Note that the AC-RM policy may improve considerably the performance of HRTs compared to WC-RM policy. Our results show that our WC-RM policy improves the performance of NHRTs up to 10% compared to AC-RM policy, allowing to execute efficiently mixed application workloads.

2 An Analyzable Multicore Processor

The research presented is this paper is based on an analyzable multicore architecture [1] that provides a WCET estimation in which the worst case inter-thread interference scenario is considered. To do so, the proposed architecture *guarantees* by design that the maximum delay that a request from a HRT accessing a shared resource can suffer by any other task is bounded by a previously computed *Upper Bound Delay* (UBD). In other words, requests to shared resources cannot be delayed longer than UBD cycles.

[3] In this paper, by default, the term resources refers to hardware resources

Concretely, [1] presents a four-core processor in which each core, that has its private data and instruction L1 cache, is connected to a multi-banked L2 shared cache through a shared bus. In such multicore architecture the on-chip inter-thread interferences that can potentially increase the WCET estimation of HRTs are: *bus, bank* and *storage interferences*.

Bus and bank interferences appear when two requests from different cores access the L2 cache through the bus at the same time. Our proposed interference-aware bus arbiter *guarantees* that a request of a HRT cannot be delayed longer than $UBD = N_{HRT} \cdot max\{L_{bus}, L_{bank}\} - 1$, where N_{HRT} is the number of HRTs running simultaneously inside the processor and L_{bus} and L_{bank} are the latencies of the bus and bank respectively.

Storage interferences appear when one task evicts valid data from the L2 cache of another task. To avoid this, the shared L2 cache implements a cache partitioning technique, called *bankization*, in which each core can dynamically assign a private subset of the total number of banks that no other core can use. Note that, by using *bankization*, bank interferences are also completely eliminated since two cores are not allowed to access the same bank, being the shared bus the only on-chip resource that may cause interferences.

Thus, by considering that every request to the shared bus has a maximum delay of UBD cycles, it is possible to compute a safe and tight WCET estimation that considers the worst case inter-thread interference scenario. To this end, our multicore processor introduces a hardware feature called *WCET Computation Mode*, in which the processor is set when computing the WCET estimation. In this new execution mode, each HRT is run in isolation and every request to the shared bus is artificially delayed by UBD cycles. By doing this, the resulting execution profile of running in the *WCET computation mode* considers the maximum possible inter-thread interference scenario, providing a safe WCET estimation that is independent of the workload the analyzed HRT is going to be co-scheduled with. Once the WCET analysis has been completed, the processor is set back to the *Standard Execution Mode*.

Therefore, our multicore architecture allows computing a safe and tight WCET estimation of HRTs when executing in a mixed application workload. Our experiments show that the WCET increment due to inter-thread interferences of a collision avoidance algorithm provided by Honeywell, is up to 25% with respect to the WCET estimation without considering interferences (see [1] for further details).

An important benefit of our multicore architecture is that, since the WCET estimation computed applying the WCET computation mode is independent from the rest of applications that compose the workload, it is possible to change, after the integration phase of the system, any task of the workload without requiring a re-estimation of the WCET for all the remaining tasks. This reduces considerably the cost of analyzability as only the task involved in the change need to be re-analyzed, as opposed of having to re-analyze the whole system.

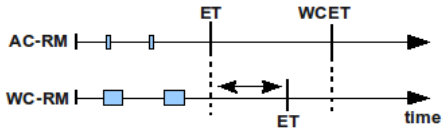

Fig. 1. Execution time (ET) of a HRT with two requests (rectangles) when using the WC-RM and AC-RM policies. In case of WC-RM, the accesses to shared resources take longer, increasing the ET of the HRT and being closer to its WCET

3 Considering NHRT Performance

As previously described, in order to execute efficiently a mixed application workload it is not only required to provide analyzability to HRTs, but also to maximize the performance provided to NHRTs. That is, since HRTs have to accomplish stringent timing constraints, the access to shared resources is prioritized for HRTs over NHRTs, allowing NHRTs to access shared resources only if there are no requests from HRTs ready to be served. Thus, it may happen that, in case of HRTs with high demands of shared resources, the performance of NHRTs are degraded since HRTs prevent the NHRTs from accessing the shared resources. It is generally the case, in fact, the use of a hardware resource management policy in which requests to shared resources are processed as soon as they are generated by HRTs without providing any performance guarantee to NHRTs. We call this policy *Average-Case Resource Management* (AC-RM).

Thus, considering that it is also important to guarantee a certain performance level to NHRTs, in this paper we propose a novel resource management policy, called *Worst-Case Resource Management* (WC-RM), that provides an efficient shared resource assignment still ensuring that HRTs meet their deadlines. With this novel policy, every access from a HRT to shared resources is stalled by UBD cycles. This forces HRTs to be executed closer to their WCET, so more resources are available for NHRTs during the HRT execution.

Note that, unlike the WC-RM policy, the AC-RM policy improves the performance of the HRTs in the average case. Instead, the WC-RM policy considers that every HRT request is executed under the worst possible inter-thread interference scenario, delaying the execution time of the task. The main requirement for HRTs it is, in fact, to meet the deadline but they do not get any benefit if they are executed faster than their WCET (in the average case), as opposed to always execute as close as possible to the WCET.

The idea behind WC-RM policy is shown in Figure 1 where a HRT accesses twice a shared resource. When applying the AC-RM policy, requests are served as soon as possible. Instead, by applying the WC-RM policy, each request is delayed by UBD cycles, effectively enlarging the execution time of the task, i.e. getting closer to the WCET. Therefore, more accesses to the shared resources can be provided to NHRTs, potentially increasing the performance of the NHRTs.

Few hardware modifications are required to implement the WC-RM policy: it is in fact required only a saturated down-counter initialized to UBD when a shared resource is accessed and decremented every cycle. Then, instead of

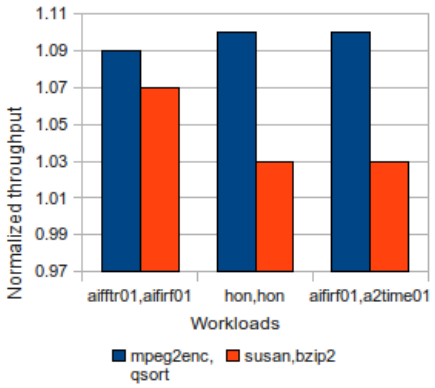

Fig. 2. Performance improvement of NHRTs when using WC-RM policy, taking as a baseline the AC-RM policy

freeing the corresponding entry on the queue as soon as the request is served, the arbiter stalls all HRT requests until the down-counter reaches zero. Instead, NHRT requests can access it, independently of the down-counter value.

4 Experimental Setup

All experiments presented in this paper use an in-house cycle-accurate execution-driven simulator derived from CarCore[4] and compatible with Tricore ISA [5] binaries and with RapiTime WCET analysis tool [6].

The WC-RM policy has been evaluated using six different mixed application workloads composed by 2 HRTs and 2 NHRTs. Regarding HRTs, we use a real hard real-time application provided by Honeywell Corporation based on a collision avoidance algorithm for 3D path planning algorithm used in autonomous driven vehicles (*hon*), and three EEMBC Autobench benchmarks with different shared resource demands: one with low demands (*a2time01*), and two high demands (*aifirf01* and *aifftr01*). Regarding NHRTs, we used two pair of benchmarks with different shared resource demanding: two with high demands (*mpeg2enc* and *qsort* from MediaBench II and MiBench Automotive benchmark suites) and two with low demands (*bzip2* and *susan corner* for SPEC CPU2006 and MiBench Automotive benchmark suites). We have simulated 100M of instructions for each HRT in the workload.

5 Results

Figure 4 shows the total throughput of the two pairs of NHRTs (*mpeg2enc-qsort* and *bzip2-susan corner*) when running in three different mixed application workload composed by different pairs of HRTS (*aifftr01-aifirf01*, *hon-hon* and *aifirf01-a2time01*) and using our WC-RM policy. The throughput is normalized

to a baseline that considers the same mixed application workloads but using a AC-RM policy.

The performance impact of WC-RM depends on the amount of accesses to shared resources that NHRTs have. Thus, in case of *mpeg2enc-qsort* NHRTs with high demands, the performance improves by 9%, 10% and 10% when running with the different pairs of HRTs; while in case of *bzip2-susan corner* NHRTs with lower demands, the performance only increase by 3%, 3%, 7%.

6 Conclusions and Future Work

In this paper we present a multicore architecture designed from a WCET point of view that implements a novel hardware shared-resource management policy, called *Worst-Case Resource Management* (WC-RM), that improves the performance level of NHRTs when running a mixed application workload. The WC-RM policy provides an efficient shared resource assignment in which every request from a HRT is delayed by UBD cycles. This forces HRTs to be executed closer to their WCET, so more shared resources are available for NHRTs during the HRT execution. Our experiments shows that by using the WC-RM policy the performance of NHRTs improve up to 10% in comparison of using a AC-RM policy.

To sum up, our multicore architecture provides an efficient way of executing mixed application workload because it provides analyzability to HRTs while maximizing the performance level of NHRTs. As future work, we plan to extend our multicore architecture to guarantee certain quality of service (QoS) for soft real-time applications.

7 Acknowledgements

This work has been supported by the Ministry of Science and Technology of Spain under contract TIN-2007-60625, by the HiPEAC European Network of Excellence and by the MERASA STREP-FP7 European Project under the grant agreement number 216415. Marco Paolieri is supported by the Catalan Ministry for Innovation, Universities and Enterprise of the Catalan Government and European Social Funds.

References

1. M. Paolieri, E. Quinones, F.J. Cazorla, G. Bernat and M. Valero. Hardware Support for WCET Analysis of Hard Real-Time Multicore Systems. In *ISCA*, Austin, TX, USA, 2009.
2. T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In *RTSS*, Phoenix, AZ, USA, 1999.
3. *MERASA EU-FP7 Project: www.merasa.org*, 2007.
4. S. Uhrig, S. Maier, and T. Ungerer. Toward a processor core for real-time capable autonomic systems. In *Proc. ISSPIT*, Athens, Greece, 2005.
5. Infineon. *Tricore 1. 32-bit Unified Processor Core v1.3*, 2005.
6. *RapiTime: Worst-case execution time analysis. User Guide. Rapita Systems. Ltd.*, 2007.