
MERASA: MULTICORE EXECUTION OF HARD REAL-TIME APPLICATIONS SUPPORTING ANALYZABILITY

THE MERASA PROJECT AIMS TO ACHIEVE A BREAKTHROUGH IN HARDWARE DESIGN, HARD REAL-TIME SUPPORT IN SYSTEM SOFTWARE, AND WORST-CASE EXECUTION TIME ANALYSIS TOOLS FOR EMBEDDED MULTICORE PROCESSORS. THE PROJECT FOCUSES ON DEVELOPING MULTICORE PROCESSOR DESIGNS FOR HARD REAL-TIME EMBEDDED SYSTEMS AND TECHNIQUES TO GUARANTEE THE ANALYZABILITY AND TIMING PREDICTABILITY OF EVERY FEATURE PROVIDED BY THE PROCESSOR.

••••• An increasing need for safety, comfort, services, and lower emissions from current and future automotive, aerospace, and production equipment requires higher performance than today's embedded processors can deliver. However, in developing safety-related real-time embedded systems, timing requirements must be met.

Multicore processors are the contemporary solution used for improving performance without increasing clock frequency. In embedded systems, multicores allow execution of mixed-criticality application workloads comprised of hard real-time (HRT) and non-HRT (NHRT) applications. This offers various benefits, including improving reliability and reducing weight and cost through decreased package count. However, mainstream multicores are designed for high average performance, resulting in nonanalyzable timing behavior (or extremely pessimistic worst-case timing estimates) that can render them unusable in the domain of safety-related real-time embedded systems.

Intertask (we use the terms "thread" and "task" interchangeably) interferences are the main difficulty in analyzing the timing behavior of multicores. Intertask interferences appear when two or more tasks sharing a given hardware resource try to access it at the same time. To prevent conflicts, an arbitration must select which task is granted access to such shared resources. This can delay the execution time of other tasks and therefore increase their worst-case execution times (WCETs). This makes a task's WCET dependent on a set of intertask interferences introduced by the concurrently running tasks, which is undesirable because a change in one task requires reanalysis of the complete task set. Hence, a major objective for the MERASA multicore architecture is to make the analysis of each task independent from the coscheduled tasks. Another objective is to design a multicore architecture that allows safe and tight WCET estimations. MERASA achieves both objectives by

The MERASA Project
Barcelona Supercomputing
Center
Honeywell International
Rapita Systems
University of Augsburg
Université Paul Sabatier

isolating task execution and bounding the effect on WCET estimations of intertask interferences in hardware and system software when shared resources need to be accessed.

The MERASA project uses two types of WCET tools: a static WCET tool (Open Tool for Adaptive WCET Analyses [OTAWA]¹) and a measurement-based WCET tool (RapiTime²). As part of the MERASA project, we enhanced both tools and used them to validate the hardware techniques proposed. Moreover, we invested significant effort in the interoperability of OTAWA and RapiTime and in crafting coding guidelines³ for WCET analyzability of applications. The MERASA project started in November 2007 and lasts 36 months. We structured the project in three main phases, which correspond roughly to a spiral development model:

- a design-space exploration phase using a high-level multicore simulator,
- an architectural refinement phase based on a detailed SystemC multicore simulator, and
- field-programmable gate array (FPGA) prototyping of the developed MERASA multicore and pilot studies.

We investigate moderately scalable embedded multicore architectures with up to 16 cores, which are themselves single- or multi-threaded by using simultaneous multi-threading (SMT) techniques. The project's reference platform is Infineon Technologies' TriCore architecture. However, the developed techniques are applicable to any embedded processor.

MERASA project achievements

Along with our top-level objective to provide a high-performance multicore design that allows bounded, tight WCET estimates, we also aim to ease the WCET analysis process. Even for multicore executing multiple HRT tasks, the WCET analysis should be similar to the single-threaded case. To that end, bounding task interferences is the key to providing a timing-analyzable system, which by design allows computing tight WCET estimations. Tasks can be isolated at the core level, whereas for shared resources (namely bus,

The MERASA Project

Members of the MERASA project include:

- Theo Ungerer, Sascha Uhrig, Mike Gerdes, Irakli Guliashvili, Florian Kluge, Stefan Metzloff, Jörg Mische, and Julian Wolf from the University of Augsburg;
- Francisco J. Cazorla, Eduardo Quiñones, and Marco Paolieri from the Barcelona Supercomputing Center;
- Pascal Sainrat, Hugues Cassé, and Christine Rochange from the Université Paul Sabatier;
- Guillem Bernat and Michael Houston from Rapita Systems; and
- Zlatko Petrov from Honeywell International

shared cache, and main memory), bounding of interferences is mandatory because such resources cannot be isolated.

MERASA core

The general MERASA multicore architecture is based on SMT cores (see Figure 1) and is capable of running a mixed application workload, comprising both HRT and NHRT threads. Each MERASA core⁴ consists of two pipelines—an integer and an address pipeline—and implements the TriCore instruction set.⁵ Each core provides up to four thread slots (separate instruction windows and register sets per thread), which allow execution of one HRT and three NHRT threads.

In the issue stage, thread scheduling is performed by hardware that dispatches the instructions from the different thread slots by priority. The HRT thread, which is completely isolated from the other NHRT threads running in the same core,⁶ receives the highest priority. To enable the isolation, we implement multicycle instructions as interruptible microcode sequences and split memory read accesses into two phases (access and write back). A prioritization privileges the HRT thread in the fetch stage, the real-time issue stage, and the intracore real-time arbiter (see Figure 2). Consequently, the core executes the HRT thread as if it were the only running thread, so we can apply common techniques for static WCET analysis used for a single-threaded core.

We dedicate two local scratchpad memories to the HRT thread to reduce the WCET for fetches and data accesses, because any memory access that is handled locally does not use the bus and does not

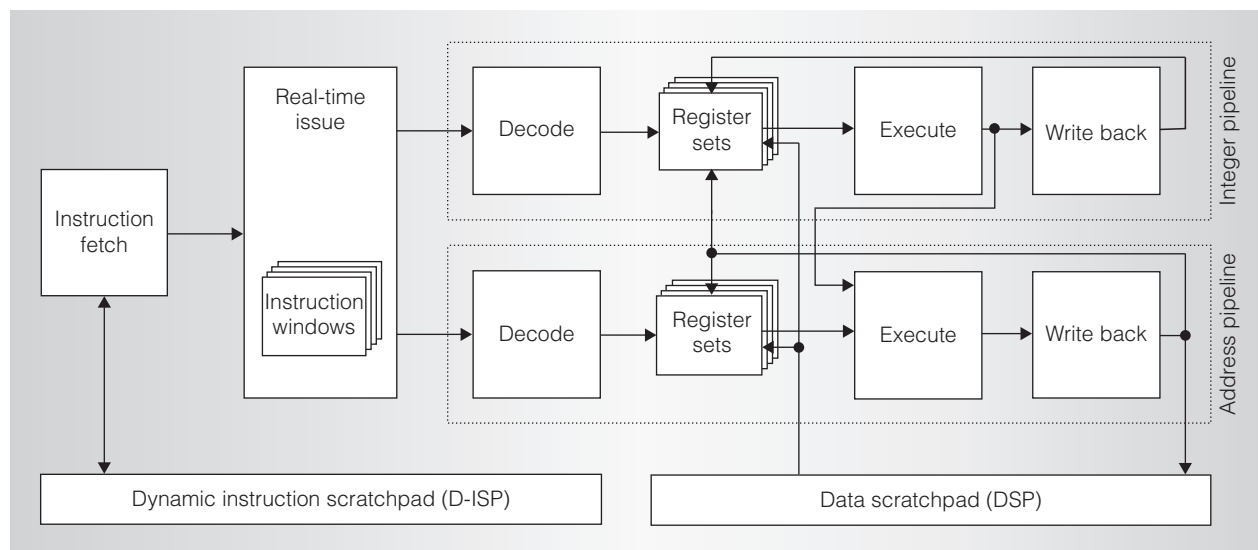


Figure 1. The general MERASA multicore architecture is a simultaneous multithreaded (SMT) core architecture with a five-stage dual pipeline. It can execute both hard real-time (HRT) and non-HRT (NHRT) threads concurrently.

suffer interthread interference. A dynamic instruction scratchpad (D-ISP)⁷ loads the complete code of an activated function dynamically on call and return. Furthermore, we use a data scratchpad (DSP) for the stack. Additionally, we provide data and instruction caches for NRT threads (not shown in Figure 1).

General MERASA multicore

Intertask interferences might appear in a multicore when tasks try to simultaneously access the same shared hardware resources. The MERASA processor design (Figure 2) takes intertask interference into account. We propose a multicore architecture in which the maximum waiting time for an HRT task to get an access to a shared resource is bounded. Particularly, at the multicore level, we focus on three main shared resources: the bus that connects cores to the shared cache, the shared cache itself, and the memory controller.

For the bus, we propose an interference-aware bus arbiter that bounds the intertask bus interference delays that HRT tasks might suffer. Our proposal splits the bus arbiter into two hierarchical components⁸: an intercore bus arbiter, which arbitrates requests from different cores; and several intracore bus arbiters, one per core, that arbitrate among requests from the same core.

This design ensures that the delay a task can suffer due to requests coming from other tasks is bounded by a fixed amount of time.

The shared cache can suffer from both bank and storage interferences. To avoid these interferences, we propose a *dynamically partitioned cache*,⁸ which assigns a private subset of cache banks to each HRT task such that no other task has access to it. The operating system sets the cache partition assigned to each core by modifying special hardware registers. Moreover, we keep each task's data and instructions separate to prevent additional interferences. Therefore, each HRT task has access to its local D-ISP and DSP, private instruction, and data cache partitions, whereas NHRT tasks have access to the first-level instruction, data caches, and a private cache partition, which works as a second-level cache that is shared among all NHRT tasks.

Finally, we propose the *analyzable real-time memory controller* (AMC),⁹ designed to minimize the impact of memory intertask interferences on the WCET. AMC allows analysis of such impact based on the generic timing constraints that define the timing behavior of the JEDEC-compliant DDRx synchronous DRAM (SDRAM) memory devices. This allows the effect of different DDRx SDRAM devices on WCET

estimations to be quantified so the designer can choose the most suitable device from a WCET estimation viewpoint rather than based on the average case performance (which usually occurs).

MERASA system software

The MERASA system-level software represents an abstraction layer between the application software and the embedded hardware.¹⁰ It provides the basic functionalities of a real-time operating system as a foundation for application software running on the MERASA processor. The challenge in this software field is to guarantee the isolation of memory and I/O resource accesses of various HRT threads that are running on different cores to avoid mutual and possibly unpredictable interferences. This isolation should also enable a tight WCET analysis of application code. The resulting system software executes HRT threads in parallel on different cores of the MERASA multicore processor.

To provide thread isolation, we apply a hardware-based real-time scheduler implemented in the real-time issue stage, and we devise a thread control block (TCB) interface for the system software. Hardware uses the TCB interface to schedule a fixed number of HRT threads (one per core) and an arbitrary number of NHRT threads into the available hardware thread slots. Additionally, a software scheduler complements the hardware scheduling by optimizing the thread scheduling.

WCET techniques and tools

We implemented support for the MERASA architecture in OTAWA, using static analysis techniques and in RapiTime, which is based partly on measurements. Inputs to OTAWA include the application's binary code and a description of flow facts (such as loop bounds) determined from the source code by the oRange tool.¹¹ OTAWA extracts the control flow graph (CFG) from the binary code. Several analysis steps, some of which we designed to fit the components of the MERASA multicore architecture (D-ISP, DSP, the instruction partition in the dynamically partitioned cache, real-time bus, and analyzable real-time memory controller), successively annotate the CFG

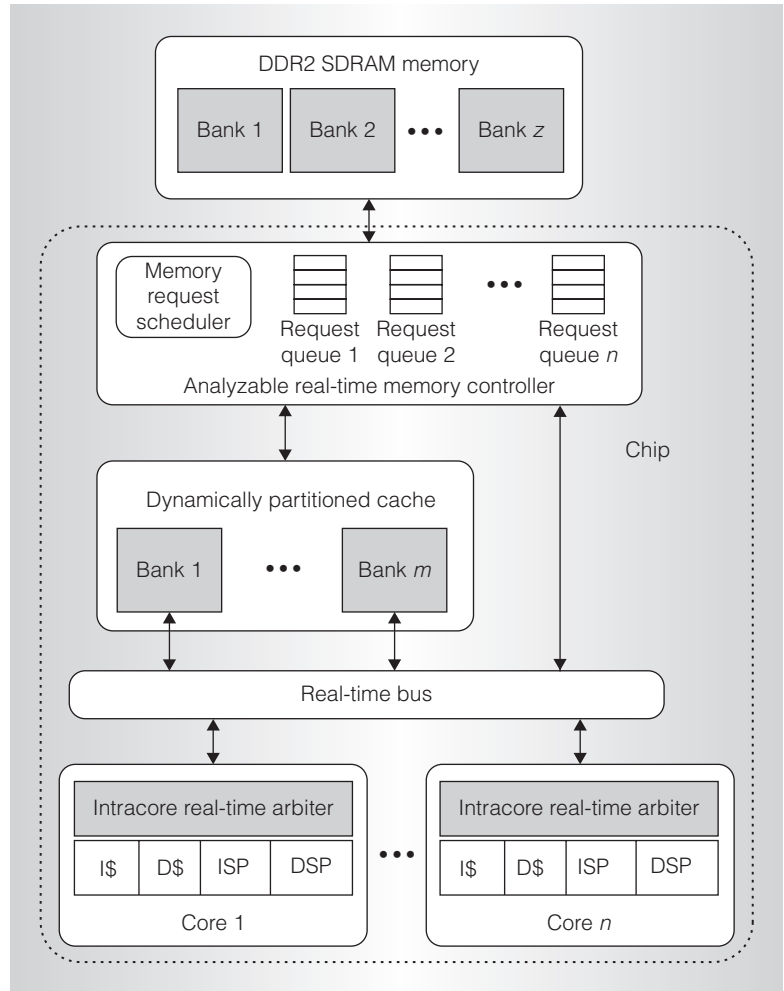


Figure 2. Block diagram showing the general MERASA multicore architecture. The main blocks are the cores, the real-time capable bus, real-time capable cache, real-time capable memory controller, and the synchronous DRAM (SDRAM) memory device.

with information about the expected timings of instructions. OTAWA then derives the execution time of basic blocks based on parametrized execution graphs.¹² From these annotations, it builds and solves an integer linear program to get the application's WCET.¹³

The simulators and the FPGA prototype record timing traces of instrumented HRT thread execution on the MERASA multicore. RapiTime uses the extracted traces to estimate the WCET. Improvements to RapiTime have let us reduce the amount of instrumentation. The latest release can operate with zero-overhead instrumentation, thus removing dependence on inserted output

instructions and allowing more accurate measurements.

We also developed coding guidelines to support a WCET analysis and defined an interchange format used for the interoperability of the WCET tools based on a common definition of code units (basic blocks and measured sequences). Expected improvements to OTAWA from RapiTime outputs include the possibility of comparing models of the target architecture to measurements and the identification of areas in which static analysis is too pessimistic. Conversely, outputs from OTAWA might provide proven upper bounds to RapiTime, as well as indications on code areas that should be further tested to improve WCET estimations. Because the MERASA architecture was changing rapidly while we were exploring design parameters, we could use RapiTime to both quickly assess the impact of modifications and verify that the static model matches the design's final implementation. Please note that static WCET analysis requires a model of the real system, whereas measurement-based WCET analysis operates on the real system.

Evaluations and pilot study

Because of the overall objectives of high performance and a tight WCET computation, we focused our evaluations on WCET computability and comparison of WCET estimates of OTAWA and RapiTime. The WCET results are critical for the MERASA multicore evaluation. We used the simulator results to investigate the tightness of the WCET analysis. Because of space limitations, we only include an extract of the evaluations here.

Detailed analysis of on-chip interferences

To the best of our knowledge, this is the first time that researchers have compared the estimations for a multicore by a static-analysis tool (OTAWA) and a measurement-based tool (RapiTime). For this kind of experiment, we must use the same settings for the MERASA multicore simulator and for the model of the multicore processor used for the static WCET analysis with OTAWA. We use the multicore simulator to compute the maximum observed execution times (MOETs) and extract data for the WCET estimations of

RapiTime. Moreover, to achieve the best code coverage, we based the measured WCETs of RapiTime on the execution of instrumented benchmarks over several iterations.

For the evaluation of interferences on shared resources, we propose using a multi-programmed workload. We used benchmarks from the EEMBC Autobench benchmark suite as HRT tasks and Honeywell International's collision-avoidance algorithm as a larger industrial application benchmark. We performed this evaluation on a cycle-accurate SystemC simulator in which we have modeled the MERASA processor and the effects of the real-time bus and the D-ISP on WCET estimations. For this experiment, we do not consider the dynamically partitioned cache and the first-level caches. In particular, we model a quad-core MERASA processor with 16 Kbytes DSP and D-ISP. We use a round robin with prioritization real-time bus policy⁸ and five cycles as memory latency for loading 64 bits of data, assuming a 200-MHz embedded processor accessing SDRAM.

Table 1 shows the results of quantitative evaluations for a selection of the EEMBC automotive benchmarks and the collision-avoidance application. To assess tightness of WCET calculations, we normalize the WCETs of RapiTime and OTAWA to the MOETs on the SystemC simulator. The SystemC simulator takes into account the maximum possible interference on accessing shared resources. Each time a thread accesses the memory, it is delayed by an upper-bound delay. For RapiTime estimates, a WCET computation mode in the simulator performs this action.⁸ The RapiTime and OTAWA WCET estimations show a relative tightness of 1.04 to 2.83 compared to the MOETs. In other words, the WCET estimations of both WCET tools are close to the measured MOETs of the selected EEMBC benchmarks. However, the measured MOET values are for given input data sets (that is, they do not necessarily represent the longest path). Thus, the difference between the MOETs and the RapiTime and OTAWA WCETs is not only caused by WCET overestimation but also by WCET analysis, which considers all possible paths. Although we can compute

Table 1. Results of EEMBC automotive benchmarks and the collision-avoidance application with maximum possible interferences.

Benchmark	SystemC simulator MOET (normalized)	Normalized measured WCET (RapiTime)	Code coverage (%) (RapiTime)	Normalized computed WCET (OTAWA)
Bitmnp	634,100 (1.0)	1.44	75.68	1.45
Canldr	5,146 (1.0)	1.04	33.04	1.09
Puwmod	9,637 (1.0)	1.32	98.82	1.41
Rspeed	4,356 (1.0)	1.16	88.46	1.21
Ttsprk	78,048 (1.0)	2.61	90.59	2.83
Collision avoidance	20,623,985 (1.0)	2.50	98.80	2.83

tight WCET estimations with both WCET tools, the collision-avoidance industrial application is highly data dependent (that is, it depends on the position and number of obstacles in the input map). Thus, we over-estimated OTAWA’s WCET estimations (2.83), but were still close to the computed measurement-based WCET (2.50). In conclusion, we can compute a tight WCET for benchmarks and industrial applications running on the MERASA multicore processor with both WCET tools.

Impact of intertask memory interferences on WCET estimations

Figure 3 shows the RapiTime WCET estimations for the Honeywell application considering the maximum delay it can suffer when running in concert with three (Figure 3a) and one (Figure 3b) memory-intensive applications. For these experiments, we consider a quad-core MERASA processor connected to a JEDEC-compliant 256 Mbyte \times 16 DDR2-400 SDRAM device, with a CPU to SDRAM clock ratio of 4. To do so, we activate the WCET computation mode⁸ in the simulator when computing the WCET estimation. We compute WCET estimations under different scenarios:

- We assume a private DDR SDRAM memory controller for each task, with intertask interferences occurring only in on-chip shared resources⁸ (“PR MC” in the figure), and
- We consider both on-chip and memory interferences using AMC⁹ shared among HRT tasks and controlling on-chip resources (“AMC” in the figure).

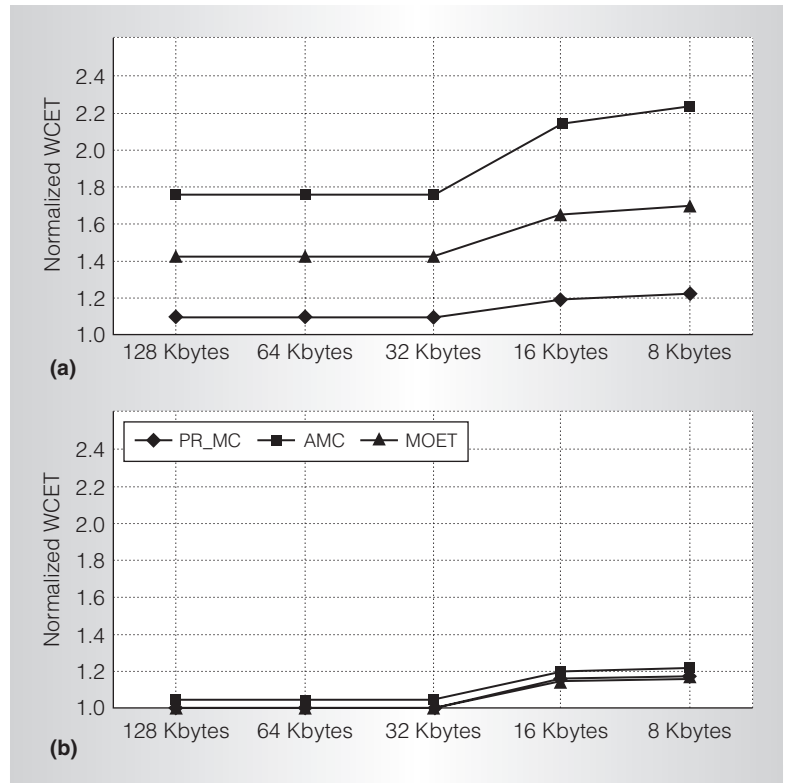


Figure 3. Normalized WCET estimation for the Honeywell application when using a JEDEC-compliant 256 Mbyte \times 16 DDR2-400 SDRAM device: Honeywell application running with three HRT tasks (a) and Honeywell application running with one HRT task (b).

We also vary the private cache partition size assigned to the Honeywell application (from 128 to 8 Kbytes) to see how AMC behaves as the pressure on the memory system increases.

Moreover, to evaluate whether the WCET estimations when using AMC are tight, we also measure the MOET of the HRT task

when running as part of a memory-intensive workload composed of several instances of the *opponent* benchmark. The opponent benchmark is a memory-intensive synthetic benchmark in which each instruction is a store that systematically misses in the last-level cache and, as a result, always accesses the main memory.

As expected, memory interferences have a significantly higher impact on the WCET estimation than on-chip interferences. Despite the high impact of memory interferences, AMC allows a tight WCET estimation. When comparing the Honeywell application's MOET in the most memory-intensive workload (that is, assigning a cache partition of 8 Kbytes and 3 HRT opponents [Figures 3a] with its computed WCET estimation for the corresponding workload), we observe (with RapiTime) an increase of only 29 percent (from 1.72 to 2.22). The curves are steady between 128 Kbytes and 32 Kbytes because the analyzed application's data footprint fits in 32 Kbytes. The performance is worse with a smaller cache, but assigning a bigger cache partition does not provide any performance benefit.

It is worth noting that the Honeywell application's WCET estimation is independent of the context in which the application runs. Using context-dependent information in the computation would reduce upper-bound delays, hence, achieving higher WCET tightness. However, obtaining such context-dependent information is costly, and it would make the WCET of each application context dependent.

FPGA and pilot studies

We performed a pilot study with Honeywell International using a MERASA quad-core processor implemented on an FPGA. To yield chip space estimations for the FGPA implementation, we estimated the hardware effort of different configurations of the MERASA multicore (that is, hardware requirements and maximum frequency on the Altera Stratix II EP2S180F1020C3 device).¹⁴ The number of adaptive lookup tables (ALUTs) and register bits grows nearly linearly with the number of cores and hardware threads.

$$\begin{aligned} & \text{logic utilization [in \%]} \\ & \leq (6 + (5 \cdot \text{slots})) \cdot \text{cores} \end{aligned} \quad (1)$$

Equation 1 shows the relation between the logic utilization of different multicore configurations. Each thread slot requires about 6,000 ALUTs and 2,000 registers and the base processor adds about 9,000 ALUTs and 3,000 registers. The D-ISP adds around 1,500 ALUTs and 700 registers per core. Also, the maximum frequency decreases when the number of thread slots increases. The frequency decrease is not linear because a higher number of slots requires a more complex scheduling logic. For the pilot study, we implemented a MERASA quad-core with two hardware thread slots, DSP, D-ISP, and the real-time bus running at around 25 MHz clock frequency.

Our pilot study implements a 3D path planning algorithm on the MERASA multicore FPGA prototype. This algorithm uses Laplace's equation for airborne collision avoidance. This technique constructs paths $r(t)$, through a 3D domain by assigning a potential value of $v(r) = 0$ for r on any boundaries or obstacles, and a potential of $v(r) = -1$ for r on the goal region. The collision-avoidance algorithm solves Laplace's equation in the interior of the 3D region, guaranteeing no local minima in the domain's interior, leaving a global minimum of $v(r) = -1$ for r on the goal region, and global maxima of $v(r) = 0$ for r on any boundaries or obstacle. We construct a path from any initial point $r(0)$ to the goal by following the negative gradient of the potential v . The pilot study shows the feasibility of the MERASA multicore targeting an autonomously flying vehicle and applying a 3D path planning algorithm using a Laplacian's solver.

The collision-avoidance algorithm gets its inputs from a commercial flight controller simulator used in the Honeywell Advanced Technologies Laboratories. It returns the estimated vehicle position and velocity vector to the flight controller simulator that displays the vehicle's movement on a screen. We have successfully demonstrated the single-threaded version of the collision-avoidance algorithm and will demonstrate a parallelized version in the near future.

Our current work focuses on additional pilot studies with industrial applications from space and automotive domains

Related Work in Real-Time Capable Designs

The first hard real-time-capable multithreaded designs mainly used scalar architectures. For example, the multithreaded Komodo processor provides several hard real-time tasks by time-sharing within a constant period of 100 cycles, but uses a special instruction set (Java byte code) and assumes scratchpad memories.¹ Jamuth, a commercial derivative of the Komodo processor, supports slower memories for soft real-time tasks.²

Schoeberl provides a time-predictable computer architecture for real-time systems that supports worst-case execution time (WCET) analysis and evaluates it using the Java optimized processor (JOP), which is a real-time Java processor.³ For performance enhancements, Schoeberl proposes a multicore system that features time-sliced arbitration of the main memory access to enable analyzability.

The real-time virtual multiprocessor (RVMP)⁴ issues multiple instructions from multiple tasks to multiple pipelines, but it assumes multiple identical pipelines. Therefore, it can execute multiple hard real-time tasks, but the throughput does not increase because other tasks cannot use idle pipeline slots dynamically.

For a predictable execution of multiple tasks in parallel, Edwards and Lee developed the concept of precision timed (PRET) machines.⁵ This approach uses a thread-interleaved pipeline to guarantee a precise timing without a loss of throughput, however a simplified WCET analysis remains possible. If a thread is stalled, another thread cannot use the cycle because the PRET architecture supports only precisely timed hard real-time tasks. No other tasks with softer timing demands can be executed to increase throughput.

Predator is another European Commission project reconciling efficiency with predictability.⁶ Its architectural approach uses analyzable caches, a compiler-controlled memory management, and simple cores with analyzable behavior. It uses predictable kernel mechanisms to cover the main challenge for a real-time operating system, which is the variability in task execution times caused by interference.

In the real-time operating system field, many commercial solutions exist. Popular examples include the QNX Neutrino (<http://www.qnx.com/products>) and the Enea OSE (<http://www.enea.com>), which recently added support for multicore processors. However, our approach also focuses on timing analyzability. Moreover, our solutions work in concert with the MERASA hardware scheduler, which guarantees an isolated execution of hard real-time tasks.

References

1. J. Kreuzinger et al., "Real-time Scheduling on Multithreaded Processors," *Proc. 7th Int'l Conf. Real-Time Computing Systems and Applications*, IEEE CS Press, 2000, pp. 155-159.
2. S. Uhrig and J. Wiese, "Jamuth—An IP Processor Core for Embedded Java Real-Time Systems," *Proc. 5th Int'l Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 07)*, ACM Press, 2007, pp. 230-237.
3. M. Schoeberl, "Time-predictable Computer Architecture," *EURASIP J. Embedded Systems*, vol. 2009, article ID 758480, 2009.
4. A. El-Haj-Mahmoud et al., "Virtual Multiprocessor: An Analyzable, High-Performance Architecture for Real-Time Computing," *Proc. Int'l Conf. Compilers, Architectures and Synthesis for Embedded Systems*, ACM Press, 2005, pp. 213-224.
5. B. Lickly et al., "Predictable Programming on a Precision Timed Architecture," *Proc. Int'l Conf. Compilers, Architectures and Synthesis for Embedded Systems (CASES 08)*, ACM Press, 2008, pp. 137-146.
6. R. Wilhelm et al., "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, 2009, pp. 966-978.

to further demonstrate the MERASA achievements. We also plan to evaluate the scalability of MERASA solutions for between 8 and 16 cores. The challenges for the last year of the project focus on providing analyzability for parallelized HRT tasks. In this case, we must bound the interferences between the threads of the same application.

To that end, we implemented the commonly used POSIX-compliant mechanisms for thread synchronization, such as mutex, conditional, and barrier variables, in a time-bounded fashion. We have shown how to compute the WCET for these functions.¹⁰ Furthermore, we enhanced the memory controller with a synchronization technique to support the system software's synchronization

functions. To enable the MERASA multicore to execute multiple threads that might share some resources, we introduced an atomic read-write instruction for synchronization of parallel HRT threads running on different cores based on the atomic swap instruction of the TriCore instruction set. We designed our atomic read-write to be performed in the same time as a usual burst access to memory. Thus, no additional latency is introduced so it does not affect the overall WCET.

Open questions are the scalability of the MERASA techniques above four cores, the analyzability support for parallel HRT tasks, and a fair performance and WCET comparison to standard multicores that do not feature an HRT-capable design.

MICRO

Acknowledgments

The MERASA Project provided most of the funding for this work. The project is a specific targeted research project (STREP) funded by the European Community's Seventh Framework Programme under grant agreement no. 216415 and performed by the partners University of Augsburg (coordinator), Barcelona Supercomputing Center, Université Paul Sabatier, Rapita Systems, and Honeywell International. Marco Paolieri is partially supported by the Catalan Ministry for Innovation, Universities and Enterprise of the Catalan Government, and European Social Funds. Eduardo Quinones is partially funded by the Spanish Ministry of Science and Innovation under the grant Juan de la Cierva JCI2009-05455.

References

1. C. Ballabriga et al., "OTAWA: an Open Toolbox for Adaptive WCET Analysis" *IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS)*, Springer, 2010, pp. 35-46.
2. "RapiTime White Paper," Rapita Systems, 2008, <http://www.rapitasystems.com/system/files/RapiTime-WhitePaper.pdf>.
3. A. Bonenfant et al., *Coding Guidelines for WCET Analysis Using Measurement-based and Static Analysis Techniques*, tech. report IRIT/RR-2010-8-FR, Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier, 2010; <ftp://ftp.irit.fr/IRIT/TRACES/IRIT-RR-2010-8-FR.pdf>.
4. J. Mische et al., "How to Enhance a Superscalar Processor to Provide Hard Real-Time Capable In-Order SMT," *Proc. 23rd Int'l Conf. Architecture of Computing Systems (ARCS 10)*, Springer-Verlag, 2010, pp. 2-14.
5. *TriCore 1 User's Manual*, v.1.3.8, Infineon Technologies, 2008.
6. J. Mische et al., "Exploiting Spare Resources of In-order SMT Processors Executing Hard Real-time Threads," *Proc. IEEE Int'l Conf. Computer Design (ICCD 08)*, IEEE CS Press, 2008, pp. 371-376.
7. S. Metzloff et al., "Predictable Dynamic Instruction Scratchpad for Simultaneous Multithreaded Processors," *Proc. 9th Workshop Memory Performance (Medea 08)*, ACM Press, 2008, pp. 38-45.
8. M. Paolieri et al., "Hardware Support for WCET Analysis of Hard Real-Time Multicore Systems," *Proc. 36th Int'l Symp. Computer Architecture (ISCA 09)*, ACM Press, 2009, pp. 57-68.
9. M. Paolieri et al., "An Analyzable Memory Controller for Hard Real-Time CMPs," *Embedded Systems Letters*, vol. 1, no. 4, Dec. 2009, pp. 86-90.
10. J. Wolf et al., "RTOS Support for Parallel Execution of Hard Real-Time Applications on the MERASA Multicore Processor," *Proc. 13th IEEE Int'l Symp. Object/Component/Service-oriented Real-time Distributed Computing (ISORC 10)*, IEEE CS Press, 2010, pp. 193-201.
11. M. de Michiel et al., "Static Loop Bound Analysis of C Programs Based on Flow Analysis and Abstract Interpretation," *Proc. IEEE Int'l Conf. Embedded and Real-Time Computing Systems and Applications (RTCSA 08)*, IEEE CS Press, 2008, pp. 161-166.
12. C. Rochange and P. Sainrat, "A Context-Parameterized Model for Static Analysis of Execution Times," *Trans. High-Performance Embedded Architectures and Compilers (HiPEAC)*, vol. 2, no. 3, Springer-Verlag, 2007, pp. 109-128.
13. Y.-T.S. Li and S. Malik, "Performance Analysis of Embedded Software using Implicit Path Enumeration," *Proc. Workshop Languages, Compilers, and Tools for Real-time Systems*, ACM Press, 1995, pp. 88-98.
14. Stratix II DSP Development Board, *Reference Manual*, Altera Corp., Aug. 2006, http://www.altera.com/literature/manual/mnl_stx2_pro_dsp_dev_kit_ep2s180.pdf.

Theo Ungerer is chair of systems and networking in the Computer Science Department at the University of Augsburg, Germany, and the coordinator of the European Commission project MERASA. He has a doctoral degree in mathematics and a habilitation degree in computer science from the University of Augsburg.

Francisco J. Cazorla leads the group on interaction between computer architecture and operating systems (CAOS) at the Barcelona Supercomputing Center. He has a doctoral degree in computer architecture from the Universitat Politècnica de Catalunya. He led the architecture work package of MERASA.

Pascal Sainrat is a professor at the Université Paul Sabatier de Toulouse and head of the Traces team at the Institut de Recherche en Informatique de Toulouse. He has doctoral degrees in computer science from the University of Toulouse.

Guillem Bernat is the founder and CEO of Rapita Systems, a spin-off company from the University of York, transferring measurement-based worst-case execution time analysis technology into commercial practice. He has a doctoral degree in computer science from the Universitat de les Illes Balears.

Zlatko Petrov is a technical leader of the Platform system group at Honeywell Advanced Technology Europe. He has master of science degrees in automation and control systems from the Technical University of Varna, Bulgaria, and in embedded systems design from the University of Lugano, Switzerland.

Hugues Cassé is an associate professor at Université Paul Sabatier de Toulouse and member of the Traces team of Institut de Recherche en Informatique de Toulouse. He has a doctoral degree in computer science from the University of Toulouse.

Christine Rochange is an associate professor at Université Paul Sabatier de Toulouse and member of the Traces team of Institut de Recherche en Informatique de Toulouse. She has a doctoral degree in computer science from the University of Toulouse.

Eduardo Quiñones is a senior researcher at the Barcelona Supercomputing Center. He has a doctoral degree in computer architecture from the Universitat Politècnica de Catalunya.

Sascha Uhrig is a senior researcher at University of Augsburg, Germany, at the chair of systems and networking. He has a doctoral degree in computer science from the University of Augsburg.

Mike Gerdes is a researcher and PhD student at University of Augsburg, Germany, at the chair of systems and networking. He

has a diploma in applied computer science from the University of Augsburg.

Irakli Guliashvili is a researcher at the University of Augsburg, Germany, at the chair of systems and networking. He has a diploma in applied computer science from the University of Augsburg.

Michael Houston is a senior software engineer at Rapita Systems. He has an ME in computer systems and software engineering from the University of York.

Florian Kluge is a senior researcher in the Computer Science Department at the University of Augsburg, Germany. He has a doctoral degree in computer science from the University of Augsburg.


Stefan Metzloff is a researcher and PhD student at the University of Augsburg, Germany, at the chair of systems and networking. He has a diploma in computer science from the University of Magdeburg.

Jörg Mische is a researcher and PhD student at the University of Augsburg, Germany, at the chair of systems and networking. He has a diploma in computer science from the University of Augsburg.

Marco Paolieri is a PhD student in the Computer Architecture Department at the Barcelona Supercomputing Center. He has a master of science degree in embedded systems design at the Advanced Learning and Research Institute, the University of Lugano, Switzerland.

Julian Wolf is a researcher and PhD student at the University of Augsburg, Germany, at the chair of systems and networking. He has a diploma in applied computer science from the University of Augsburg.

Direct questions and comments to Theo Ungerer, University of Augsburg, Universitätsstr. 6a, 86159 Augsburg, Germany, ungerer@informatik.uni-augsburg.de.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.