Using Multi-Core Architectures to Execute High Performance-Oriented Real-Time Applications

C. AUSSAGUES[a,1], E. OHAYON[a], K. BRIFAULT[a] and Q. DINH[b]
[a] *CEA LIST, Embedded Real Time Systems Laboratory*
[b] *Dassault Aviation, DGT/DPR (General Technical Dir. / Future Research Business)*

Abstract. This paper presents a method and the associated tools to design and implement embedded real-time systems that execute high-performance-oriented applications on multi-core architectures. After presenting the OASIS design, compilation and execution framework, the paper focuses on the principles of the modifications performed on the safety-oriented real-time kernel to execute transparently and consistently multitasking applications on multi-cores. It then gives the first results of benchmarking performed on a Dassault Aviation's HPC application called "2D-tracking algorithm". It concludes on the future works towards the introduction of data-parallelism in the OASIS parallel time-triggered programming and underlying execution models.

Keywords. Embedded high-performance computing, real-time, multi-cores.

Introduction

Recently, major processor manufacturers introduced support for multi-processors on a single chip (Multi-Core), and many hardware vendors started to develop and market Multi-processor system-on-chip (MPSoC). These multiprocessors become fairly popular and attractive for embedded systems as they bring more performance in offering higher parallel computation capability and lower power consumption [1].
This tendency accelerates the emergence of high-performance embedded applications and emphasizes the use of parallel programming models integrating different levels of parallelism where a trade-off between expressed parallelism and automatic parallelization is necessary. So, embedded real-time (RT) applications are starting to run on Symmetric Multi-Processing (SMP) systems. But, extending an existing Operating System (OS) to support SMP systems is not a trivial task [2]. To take advantage of a SMP architecture, an OS needs to take into account the shared memory facility, the migration & load-balancing between processors, and the communication between tasks. Besides, implementing all the SMP mechanisms and considering how they interfere is particularly laborious when the chosen OS is a hard real-time one [3], raising the issues of synchronization and predictability. For embedded real-time

[1] CEA LIST, Embedded real time systems laboratory, Point Courrier 94, Gif-sur-Yvette, F-91191 France, E-mail: christophe.aussagues@cea.fr

applications, a key problem is to be able to bridge the gap between parallelism expressed in design (*modularity*) and those available in multi-core based architecture (*performance*). Use of multi-core architectures in embedded systems should impact as less as possible the software design and the run-time behaviour, especially when real-time constraints are at stake. Like in High-Performance Computing (HPC), the concern for embedded high-performance real-time systems is to allow increasing performance from single-core architectures to multi-core ones. But, it means that executing embedded applications on single-core or multi-core systems should provide the same results/outputs in real time and that it should be possible to add new functionalities (e.g. real-time tasks) to existing embedded applications with all real-time constraints still fulfilled.
After reviewing in section 1 related work on several common Real-Time Operating Systems (RTOS) and their extensions to multi-core architectures, we present our research work to extend the OASIS (see section 2) hard real-time kernel for HPC-oriented embedded systems. The design of the OASIS kernel to take full advantage of SMP multi-cores will then be presented in section 3. Before concluding and addressing future works, section 4 will provide the results of the first experiments with an HPC-based 2D-object tracking application provided by Dassault Aviation and ported to OASIS-SMP.

1. Related works

Several approaches exist to design a SMP RTOS. A first idea consists in proposing a monolithic kernel, in which the entire operating system is run in kernel space as supervisor mode. All OS services run along with the main kernel thread, residing in the same memory area. An example is VxWorks, where the kernel and tasks run in one address space. This allows tasks switching to be very fast and eliminates the need for system call traps. Monolithic systems are easier to design and implement than other solutions, and are extremely efficient when well written. However, the main disadvantages of monolithic kernels are the dependencies between system components, and the fact that large kernels become very difficult to maintain (lack of modularity).
A second approach consists in running the RTOS on top of a micro-kernel, which only implements core services such as process scheduling, inter-process communication, low-level network communication, and interrupt dispatching. All other services, such as device drivers or file systems, are implemented as user processes making the kernel smaller and faster. Some examples are the QNX Neutrino or L4Linux. These user-space implementations are easier to debug, while memory accesses violations can be detected and isolated. Another advantage is scalability. However, performance can be a weak point since micro-kernel architectures place an heavy load on inter-process communication and context switching.
A third idea consists in modifying a SMP kernel to dedicate some processors to real-time tasks. Those asymmetric kernels, such as APRIX[4], ASMP-Linux [8] or ARTiS [5], consider real-time processes and related devices as privileged and shield them from other system activities. Piel et Al. [6] extend a standard SMP Linux kernel with a real-time scheduler, and propose a migration mechanism in order to allow flexibility in the processor allocation. Even though their work provides a comprehensive performance evaluation, it considers an event-driven real-time system, not suitable for hard real-time operations, while simultaneously addressing the cohabitation matter (i.e. mixing real-time and generic purpose tasks within the same system).Both problems addressed

jointly raise major resource management issues such as the denial of service problem, not resolved in their approach.
A fourth approach is based on virtualized clusters, where each processor runs a modified version of the Linux kernel, and where the kernels cooperate in a virtual high-speed and low-latency network. An example is Adeos nano-kernel [7]. In a similar approach Kagström et Al. [9] address the SMP extension of an existing OS. A micro-kernel is added to every slave processor and the main single-processor system is kept almost unchanged. Benchmarks results along with a minimal kernel hacking are provided. Their work is close to our, as they consider asymmetric SMP OS.

2. OASIS Principles

OASIS [10][12] defines a method and a set of tools to design and implement multitasking deterministic real-time systems. It consists in a programming language, named ψC and an execution environment, for the hard real-time embedded systems, that uses a fully parallel time-triggered execution model beyond the classical TT paradigm [13]. ψC allows a formal description of the different real-time tasks (called agents) in an application, where pure algorithmic parts written in ANSI C are combined with an observable temporal behaviour described in a declarative way. The execution environment offers a safety-oriented real-time kernel (for single-core architectures) and an associated tool chain for the implementation of the OASIS model.

The OASIS model brings some novelties in comparison with the classical so-called TT approach: it allows asynchronism by the coexistence of different time scales between the agents in order to make the application design easier. It facilitates the decomposition of the application into parallel agents defined in elementary activities (see [12]). Time management in the OASIS kernel is entirely lock-free to avoid the overhead, contention problems, deadlocks and the possibility of processes priority inversion.
The OASIS communication model is composed of two mechanisms: periodic data flows (called temporal variables), and message boxes. These non-blocking communication mechanisms are completely described in an interface that links the programming model to the underlying execution environment.
The OASIS model implies no assumptions on the scheduling algorithm. As it has a TT approach, there are neither precedence constraints nor resource protocols, and only temporal constraints are required. The Earliest Deadline First (EDF) algorithm [17] can be used as it is a simple, sufficient and optimal one [11].

3. Design of OASIS-SMP

3.1. Objectives – Target architecture

The aim of the OASIS-SMP extension is to allow the programmer to benefit *transparently* from the computation power increase supplied by the multi-processors. The OASIS time-triggered execution model is preserved, and the issue of task dispatching between execution cores is hidden from the programmer: *no modification*

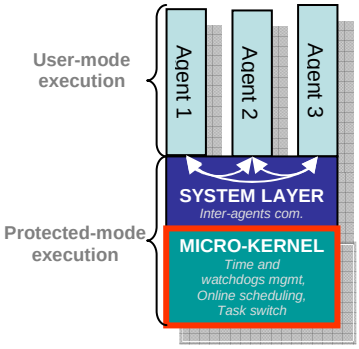
is required to the OASIS programming model, and therefore to the application ΨC source code. Ultimately, the new kernel is even able to run on different CPUs multiple agents of an OASIS application that was compiled for the original single-core version.
The IA-32 OASIS kernel has been modified to be able to run on Intel SMP 32bit platform. This is a very popular architecture in the mass-market of personal computers, where the different identical cores share a single memory bus for a Uniform Memory Access topology. The choice was motivated by the commonness of the architecture, and the fact that a robust IA-32 single-core version of OASIS is already available.

3.2. Scheduling policy

Obviously, the choice of the scheduling algorithm is critical when designing a "hard" real-time operating system. In order to map automatically and transparently the different agents of an application to the available cores, the chosen scheduling policy for the SMP version is a fairly intuitive extension of EDF – Global-EDF [18]. The priority index of a task is still defined by the time remaining before its deadline[2], only if *n* cores are available, then the *n* first tasks with the highest priority are distributed among the CPUs.
In the general case, Global-EDF is ***not*** an optimal multi-processor algorithm [19]. Although other algorithms may be later considered (see [16]), it provides a fast, easy to implement policy, and it is totally transparent to the user.

3.3. Software parallelization analysis

The following figure shows the general software architecture of an OASIS system, separated in three layers.



The user agents: each agent is a real-time task defined by the programmer. The OASIS execution model forbids memory sharing: inter-task communications are "services" handled only by the kernel (see below). Therefore, synchronization between agents –e.g. resource sharing– is ensured by their temporal behaviour: no locks are allowed. The OASIS design ensures that agents may safely be executed simultaneously (or interleaved, on the single-core version). This also explains why the application source code requires no modification for execution on the SMP version.

The system layer: this part of the OASIS kernel ensures safety-oriented sanity checks and inter-agents communication. The system layer and especially the communication mechanisms were originally written to be entirely interruptible. Therefore, executing instances of the system layer for different agents on different cores simultaneously does not raise any new problem[3].

[2] The lower the priority index, the more the task is priority.

[3] This property is true thanks to the memory consistency model of the Intel processors, which is strong enough for this application.

The micro-kernel: it includes among other things the scheduler and the context-switcher. It also updates the global real-time clock and handles the watchdogs for time-triggered events. All these modules may not be executed simultaneously and require atomic execution to ensure the consistency of the micro-kernel data structures. This property is implemented by a fair active-waiting spinlock protecting the whole micro-kernel.

3.4. Time management – Asymmetric kernel architecture

Global time management and time-triggered events are handled by a specific CPU, chosen arbitrarily at boot-time, referred further as CPU0. Conversely, software system calls (or "traps") may happen on any CPU, leading to the execution of the scheduler. A re-scheduling arms a new watchdog that will be handled by CPU0. Therefore, most of the micro-kernel code – the scheduler and the context switcher – may be executed on any CPU. Only time management parts run on CPU0.
The SMP architecture is made asymmetric by this choice, since CPU0 suffers from a micro-kernel execution overhead. However this choice is a fair compromise between:

- a symmetric architecture, where each CPU would update its local time, requiring a synchronization protocol;
- a completely asymmetric architecture, where one control-CPU would be devoted to handling all micro-kernel operations for the other computation-CPUs.

4. Running a HPC application on OASIS-SMP

4.1. Parallelizing Dassault Aviation's tracking algorithm

The HPC application provided by Dassault Aviation is a sequential implementation of a tracking algorithm, relying on Bayesian's probabilistic methods to detect moving objects in a two-dimensional field based on real-time radar data. The implementation consists of a global loop refining iteratively a cloud of particles, used for objects detection. The application is so compute-intensive on single-core architectures that its present usage is only as a background computation, i.e. to be run only when there are enough computer resources left. On the other hand, this code is locally and globally highly parallelizable.
OASIS provides a safe and real-time execution environment for this application, where parallelism can be expressed at a task level in order to improve performances.
Therefore, porting Dassault's implementation to a ΨC application basically requires:

1. Defining a temporal behaviour of the application, i.e. splitting the code into different elementary activities with an associated deadline;
2. Expressing task-parallelism by defining several communicating agents that share the computation load and are executed simultaneously on different cores by the OASIS-SMP kernel.

A master-slave approach was used to split the application into different agents: one master agent runs the code, discharging punctually a part of its work to one or more slave agents. As seen before, the OASIS kernel requires that communication between the master and its slaves go through specific kernel mechanisms, which ensure memory isolation between the agents. Since shared memory is prohibited, all communications

require copying data between source and destination buffers. For that reason, communication overheads are expected. In order to minimize them, only the heaviest computation section within the loop is parallelized, while the sole master agent executes the rest sequentially.
The ΨC version of Dassault Aviation's application defines 5 temporal synchronization points and 2 actual data exchanges between the master and its agents. The deadlines associated to each time window are based on performance measurement of the original C code executed in a Linux environment. The sum of each of these deadlines defines a global end-to-end real-time constraint inferior to 1 second per iteration, following Dassault's requirements.
Note that when the master is executing sequential portions of the code, the slave agents have basically nothing to do, and are automatically put at rest by the scheduler. See *Figure 1 – Architecture of the ΨC application*.

4.2. Benchmarks results

The OASIS-SMP micro-kernel implements utility system calls to retrieve easily the *actual* CPU time used by an agent during its last temporal slot (independently from the associated deadline). In order to compare relevant execution times, Dasssault Aviation's application was ported into two ΨC versions:

a) a first task-parallelized version following the master/slave software architecture described in the previous section, with one master agent and one slave agent.
b) A sequential version made of a single agent, with the same temporal behaviour as the master agent of the previous application (i.e. defining the same temporal synchronization points).

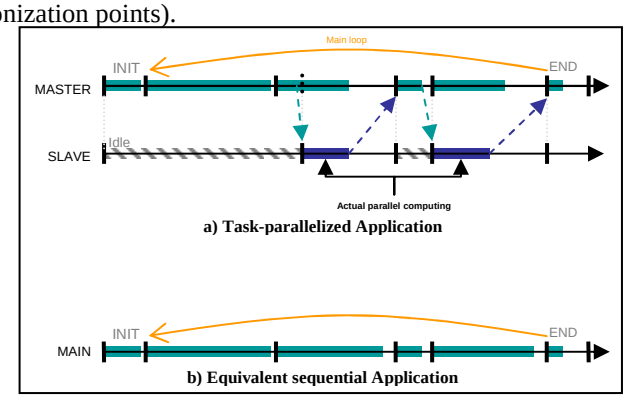


Figure 1 – Architecture of the ΨC application

A first observation is that both applications have the exact same real-time behaviour, as expected with the OASIS runtime environment.
The following figure 2 (left chart) shows the global speedup of the master/slave version against the sequential version for the first loop iterations. It appears that performances are globally not impacted by parallelization. The slight performance variations are due mainly to cache filling effects for the first iterations, then to the fluctuant complexity of the floating-point computations.

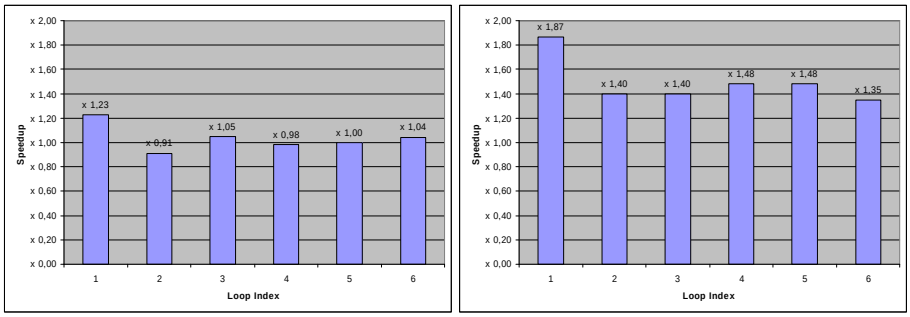


Figure 2 – **Left**: global speedup with parallel ΨC application
Right: local speedup, focusing on parallelized sections only – communication are not considered

The figure 2 (right chart) shows a measured speedup comparison on purely parallelized elementary activities, i.e. on time slots where no communication is made between the master and the slave, only computation (parallel versus sequential). The parallel version shows significant performances improvement, close to optimum (x2).

In conclusion, communications between the master and the slave show overheads, as expected: indeed, one OASIS communication system call requires three memory moves or more and two MMU switches. However, the performance improvement observed on purely parallel section is very promising and opens the road to more efficient parallelization methods.

5. Conclusions and future works towards data-parallelism

This paper presented the principles of extending a safety-oriented real-time kernel, OASIS, to take full advantage of multi-processors on a single chip in terms of performances. The extension of OASIS runtime environment to exploit parallelism at the kernel level allows supporting the execution of multitasking embedded applications on multi-cores in a transparent and consistent way: the same application is running on single- and multi-cores with the same real-time behaviour, but allowing the integration of more real-time tasks in an efficient manner. The modifications of the micro-kernel were presented, as well as the first results of the OASIS-SMP kernel use on a high-performance oriented real-time application provided by Dassault Aviation.
The results presented in section 4 are consistent with the expectations, in both temporal behaviour and global performances perspectives. They also show that task-parallelism is not always sufficient to take fully advantage of a multi-core architecture.
Due to their safeness based on memory isolation, OASIS's inter-agents communication mechanisms are also costly. In order to workaround this limitation, an agent should be able to split into several execution threads sharing the same memory context. A single agent, splitting on demand into multiple memory sharing execution threads, would then replace the master-slave software architecture.
A new extension of the OASIS-SMP kernel implementing such capability is currently being tested and already shows very promising results. Future directions to improve performances include implementing the extension of the ΨC language to express and take advantage of the data-parallelism. Another direction could be the evaluation of the performance impacts of OASIS-SMP on specific embedded parallel applications from the points of view of cache/memory accesses or inter-processor interrupts.

6. Acknowledgements

The research presented in this paper has partially been supported by the French Ministry for Economy, Industry and Employment (DGCIS) through the ITEA2 project "ParMA"[4] (n°06015, June2007--May 2010). The authors would like to thank V. David and M. Lemerre for their involvement in these works and their relevant comments.

References

[1] R. Sasanka, S.V. Adve, Y-K. Chen and E. Debes, The energy efficiency of CMP vs. SMT for multimedia workloads, *in ICS'04: Proc. of the 18th annual international conference of Supercomputing* (2004), 196—206.

[2] W. Wolf, The future of Multiprocessor Systems-on-Chips, *in DAC'04: Proceedings of the 41st annual conference of Design Automation* (2004), 681—685.

[3] J. A. Stankovic, R. Rajkumar. *Real-Time Operating Systems*. Kluwer Academic Publishers, 2004.

[4] M. Seo, H. Seok Kim, J. Chan Maeng, J. Kim and M.Ryu, An Effective Design of Master-Slave Operating System Architecture for Multiprocessor Embedded Systems, in *Advances in Computer Systems Architecture*, Lecture Notes in Computer Science, Berlin, vol. 4967 (2007).

[5] P. Marquet, E. Piel, J. Soula and J-L. Dekeyser, Implementation of ARTiS, an asymmetric real-time extension of SMP Linux, *in Proc. of the 6th Real-time Linux Workshop* (2004).

[6] E. Piel, P. Marquet, J. Soula, J-L. Dekeyser, Real-time systems for multiprocessor architectures, *in IPDPS'06 : Proc. of the 20th international parallel and distributed processing Symposium* (2006), 8—16.

[7] K. Yaghmour, A practival approach to Linux Clusters on SMP hardware, *Tech. report of ADEOS Project* (2002), 1—12.

[8] E. Betti, D.P. Bovet, M. Cesati, and R. Gioiosa, *EURASIP Embedded Systems Journal*, Hard real-time performances in multiprocessor-embedded systems using ASMP-Linux, Hindawi Publishing Corp. 2008.

[9] S. Kagström, H. Grahn, L. Lundberg, *Softw. Pract. Exper. Jounral*, The application kernel approach – a novel approach for adding SMP support to uniprocessor operating systems, John Wiley & Sons Inc., 2006.

[10] V. David, C. Aussaguès, C. Cordonnier, M. Aji and J. Delcoigne, OASIS: a new way to design safety critical applications", in *21st IFAC/IFIP Workshop on Real-Time Programming* (WRTP'96), 1996.

[11] C. Aussaguès. V. David, A method and a technique to model and ensure timeliness in safety critical real-time systems, *in Intl Conference of Engineering of Complex Computer Systems* (1998), 2—12.

[12] D. Chabrol, G. Vidal-Naquet, V. David, C. Aussaguès, and S. Louise, OASIS: a chain of development for safety-critical real-time systems, *in Proc. of Embedded Real-Time Software Conference (ERTS'04)*, (2004).

[13] H. Kopetz, The Time-Triggered Model of Computation, *in RTSS'98: Proc. of the 19th real-time Systems Symposium* (1998), 168—178.

[14] H. Kopetz, The Time-Triggered Architecture, *in ISORC'98: Proc. of the 1st IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (1998), 22—29.

[15] C. Liu, J. Layland. *Journal of the ACM*, Scheduling algorithms for multiprogramming in a hard real-time environment, ACM Publisher, 1973.

[16] Lemerre, M., David, V., Aussaguès, C., and Vidal-Naquet, G. 2008. Equivalence between Schedule Representations: Theory and Applications. *In Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium - Volume 00* (April 22 - 24, 2008). RTAS. IEEE Computer Society, Washington, DC, 237-247.

[17] Lamport, L. 1990. Concurrent reading and writing of clocks. *ACM Trans. Comput. Syst.* 8, 4 (Nov. 1990), 305-310

[18] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. *In Hand-book on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004.

[19] Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). *In STOC '97: Proc. of the 29th annual ACM symposium on theory of computing, pages 140–149*, New York, NY, USA, 1997. ACM

[4] http://www.parma-itea2.org/