Towards a Ravenscar Extension for Multi-Processor Systems

José F. Ruiz

AdaCore
46 rue d'Amsterdam
75009 Paris, France

Phone: +33 (0)1 49 70 67 16
Fax: +33 (0)1 49 70 05 52

ruiz@adacore.com

Abstract

The Ravenscar profile, originally designed for single processors, has proven remarkably useful for modelling verifiable real-time single-processor systems. New software demands increasing processing power, and multi-processor platforms are spreading as the answer to achieve the required performance. Embedded real-time systems are also subject to this trend, while keeping the need for the founding principles of Ravenscar: simplicity, efficiency, reliability, predictability, and analyzability. This paper proposes a simple extension to the Ravenscar profile to support multi-processor systems using a fully partitioned approach. The implementation of this scheme is simple, and it can be used to develop applications amenable to schedulability analysis.

1 Introduction

The Ravenscar model for single processors defines a deterministic and analysable tasking model for single processors which can be supported with a run-time system of reduced size and complexity. It supports accurate analysis of real-time behavior using Rate Monotonic Analysis (RMA) [7] and Response Time Analysis (RTA) [6]. In recent years, research on scheduling theory for multi-processor systems [5, 3] has paved the way to timing analysis in multi-processor systems.

According to the allocation of priorities to tasks, there are either static off-line scheduling algorithms, or dynamic policies, where priorities are calculated at run time. Dynamic-priority scheduling algorithms for multiprocessors, such as pfair scheduling [4], can achieve better processor utilization than static-priority ones. However, the higher complexity of dynamic algorithms, their much higher run-time overhead, and their lower predictability and robustness in overload situations make then less attractive for high-integrity systems.

In terms of relationship between tasks and processors, the spectrum goes from global scheduling, where any task can be executed on any processor at any time, to partitioned scheduling, where each task is allocated for its whole lifetime to concrete processors. The schedulability of neither approach is strictly better than the other [1] (there are task systems that are feasible using a global partitioning that cannot be scheduled in a partitioned system and vice versa). However, the partitioned approach has some interesting advantages: 1) it can rely on well-known optimal single-processor priority-assignment schemes and timing analysis techniques (local RMA on each processor), and 2) the run-time support is simpler.

This partitioned approach simplifies also development and testing, and eventually certification, by the physical separation between tasks executing in different processors (a concept largely exploited by the Integrated Modular Avionics approach).

According to the Ravenscar principles of simplicity, reliability, and predictability, fully partitioned scheduling, using static-priority policy, appears as the natural extension of the single-processor Ravenscar profile. There are tools and techniques supporting the allocation of tasks to processors, the assignment of task's priorities, and the timing analysis of the resulting systems. The major drawback of such scheduling mechanism is that the maximum worst-case utilization is a third the capacity of the platform [1].

Note that finding an optimal assignment of tasks to processors is a NP-hard bin-packing problem that needs to be solved off-line, although there exist heuristic algorithms which are reasonably good.

The following sections will describe the specific additions to the existing single-processor Ravenscar profile to support multi-processor systems using a fully partitioned approach, with each task allocated statically to a concrete processor. This scheme can be supported by a streamlined run-time system, and applications built following this approach can apply timing analysis techniques on each processor separately.

2 Task Scheduling

We propose a simple extension to the single-processor fixed-priority preemptive scheduling algorithm supported by the Ravenscar profile, where tasks are statically allocated to processors and task migration among processors is not allowed. Each processor implements a preemptive fixed-priority scheduling policy with separate and disjoint ready queues. A task is only on the ready queues of one processor, and the processor to which a task belongs is defined statically. Whenever a task running on a processor reaches a task dispatching point, it goes back to the ready queues of the same processor.

Tasks are statically allocated to processors using a new pragma (*Affinity* was proposed by Wellings and Burns [10]). If the pragma *Affinity* is not specified, the task is allocated to a default processor.

The underlying idea is that each processor executes a statically defined set of tasks, as it would be the case for Ravenscar on a single processor.

There is a single run-time system, where the only per-processor information would be the ready queues. Some run-time data is common and shared among tasks in different processors (such as the tasks waiting in the delay queue).

When internal data in the run-time system can be modified by tasks executing in different processors, we need to add inter-processor locking mechanisms (such as spinlocks), to guarantee mutual exclusion. The standard single-processor solution of disabling interrupts to guarantee that the task is not preempted before the access has been completed is not sufficient for multi processors.

Finally, something that must be taken into account is that the execution of a task (or an interrupt handler) in a given processor may modify another processor's ready queues (and may also force the preemption of the running task). These operations on a different processors can be implemented triggering an special interrupt in the target processor, which is the one performing the actual changes in the ready queues.

3 Task Synchronization

The restricted library-level protected objects defined by the Ravenscar profile are used for inter- and intra-processor communication and synchronization. The same restrictions that exist in the Ravenscar profile for single processor apply to the case of a multi processor (a maximum of one protected entry per protected object with a simple boolean barrier using ceiling locking access).

One big advantage of single-processor Ravenscar is the simple and very efficient synchronization mechanism required for protected objects, where entering/exiting to/from the protected object can simply be done by just increasing/decreasing task's priorities [9].

In order to simplify timing analysis, and to allow for an efficient implementation when possible, protected objects can be allocated to processors using the pragma *Affinity*. These protected objects, bound statically to processors, would use the optimized single-processor implementation, and can only be used by tasks allocated to the same processor. *Tasking_Error* is raised if a task tries to use a protected object allocated to another processor than the one the task is bound to.

Protected objects for inter-processor communication are those for which the pragma *Affinity* is not specified. In this case, the compiler would select the underlying multi-processor synchronization mechanism (probably using spinlocks). When a task waiting on an entry queue is awaken by another tasks executing in a different processor than the waiting task, we need to use the inter-processor interrupt facility to modify the ready queues, as described in section 2.

One of the advantages of using affinities for protected objects is to potentially allow for a more efficient implementation of the protocols to lock and release protected objects. But the most important advantage is when doing timing analysis of the application, because it guarantees that, when using local protected objects, there is no possible interference coming from tasks executing on other processors, as well as no interference being caused on other processors.

4 Interrupt Handling

The only mechanism to specify interrupt handlers is to attach a protected procedure, the same way as it is done in single-processor Ravenscar. The difference in a multi-processor system is that the affinity of the interrupt is set. The affinity for the interrupts is that of the protected object that contains the protected handler.

The effects of the interrupt handler will be those of the protected operations, and therefore the same considerations for intra- and inter- processor synchronization (as described in section 3) apply.

5 Timing Services

A single clock and a single delay queue are used for all the processors, providing a common reference for all the tasks in the system, and thus avoiding time drifting problems.

Hardware timers are executed in the context of a given processor, and when a timer expires, it has an effect on the ready queues of potentially any processor, not only the one where the timer interrupt is handled. The timer handler may awake tasks waiting on a delay statement, or execute the protected actions associated to either timing events or execution time timers. In any of these cases, the mechanisms to exert the required actions in the different processors are the same as those described in previous sections 2, 3, and 4.

6 System Modelling

The selected partitioned approach has the advantages of being simple to implement, and of allowing the use well-known optimal single-processor priority-assignment schemes and timing analysis techniques.

Additionally, the partitioned approach simplifies development, testing, verification, validation, and eventually certification, by the physical separation between tasks executing in different processors. This concept is the major strength of the Integrated Modular Avionics architecture and the ARINC 653 [2] standard, that enables independently-produced applications to execute together on the same hardware.

The problem of assigning tasks to processors is removed from the system scheduler, and needs to be solved off-line in this partitioned scheme. An optimal assignment of tasks to processors is a NP-hard bin-packing problem, but there exist heuristic algorithms [8], such as Rate-Monotonic-Next-Fit (RMNF), Rate-Monotonic-First-Fit (RMFF), and Rate-Monotonic-Best-Fit (RMBF).

7 Schedulability Analysis

Once the system has been partitioned, the schedulability analysis can be performed simply using RMA on each processor.

What needs to be taken into account is the inter-processor interference motivated by the potentially simultaneous access to shared hardware and data (such as the timer and delay queue), and by the explicit inter-processor synchronization using protected objects.

The major drawback of the partitioned scheduling is that the maximum worst-case utilization is a third the capacity of the platform [1], but the advantage is its simple implementation and timing analysis.

8 Conclusions

This paper contains a very schematic description of some ideas for a simple and natural extension to the Ravenscar model to address multi-processor systems. The idea behind it is to take an Ada application (the whole partition in the Ada RM sense), and to statically allocate each task and protected object (and therefore interrupt) to a processor. Any given processor will then have a set of tasks, protected objects and interrupts to handle, that can be modelled and analyzed as a separate single-processor Ravenscar system.

A new pragma *Affinity* is proposed again (it was already introduced in [10] for unrestricted Ada) to support task partitioning, a concept already supported by most operating systems. It is proposed to extend this notion to protected objects (and interrupts, as already supported by some operating systems) to allow for an easier way to model and analyze intra-processor task synchronization. Inter-processor task synchronization is made explicit by using protected objects which are not allocated to specific processors.

This paper describes a first set of ideas about Ravenscar for multi-processors, but there are several aspects to be refined, both for the model itself and for the implementation. In terms of the multi-processor model, there is an asymmetry between the meaning of a task without an specific affinity (allocated to a default predetermined processor) and a protected object without specified affinity (usable by any processors).

With respect to the implementation, handling the access to shared resources (hardware and data) in a simple and efficient way is the main issue. Inter-processor interrupt facilities are the mechanisms typically used to enforce scheduling and dispatching operations on different processors.

The proposed partitioned seems to provide a simple and analyzable model which can be supported by a streamlined run-time system.

References

[1] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *RTSS'01: Proceedings of the 22nd IEEE Real-Time Systems Symposium*. IEEE Computer Society, 2001.

[2] ARINC. *ARINC Specification 653, Avionics Application Software Standard Interface*. Aeronautical Radio, Inc, 2005.

[3] T. P. Baker. An analysis of fixed-priority schedulability on a multiprocessor. *Real-Time Systems*, 32(1–2):49–71, 2006.

[4] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D.A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1994.

[5] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and Sanjoy Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004.

[6] M. Joseph and P. Pandya. Finding response times in real-time systems. *BCS Computer Journal*, 29(5):390–395, 1986.

[7] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

[8] Y. Oh and H. Son. Tight performance bounds of heuristics for a real-time scheduling problem. Technical report, Department of Computer Science, University of Virginia, 1993.

[9] José F. Ruiz. GNAT Pro for on-board mission-critical space applications. In *Ada-Europe 2005, 10th Ada-Europe International Conference on Reliable Software Technologies*, volume 3555 of *Lecture Notes in Computer Science*, pages 248–259. Springer, 2005.

[10] A. J. Wellings and A. Burns. Beyond Ada 2005: allocating tasks to processors in SMP systems. *Ada Letters*, XXVII(2):75–81, 2007.