Precise Response Time Analysis for Ravenscar Kernels *

Juan Zamorano Juan Antonio de la Puente

Department of Telematics Engineering
Technical University of Madrid, Spain
E-mail: <jzamora@fi.upm.es>, <jpuente@dit.upm.es>

Abstract

The Ravenscar Profile defines a subset of the Ada95 tasking constructs which can be implemented using a small, reliable kernel. One of the benefits of this approach is to improve the timing analysis because non-deterministic and non-analyzable features are excluded. But to perform a precise schedulability analysis of a Ravenscar compliant application, the kernel overheads must be taken into account. Therefore, a set of metrics and documentation requirements has to be defined in order to model the kernel influence in response time analysis. In this paper the main components of the response time of periodic and sporadic tasks running on a Ravenscar kernel are identified, and the set of kernel metrics which are required to perform a precise timing analysis is identified.

1 Introduction

The Ravenscar Profile is a subset of Ada tasking which was defined at the 8th International Real-Time Ada Workshop (IRTAW 8) [4] with the aim of enabling the use of Ada tasking in high-integrity real-time systems [11]. Minor modifications to the profile were done at IRTAW 9 [3] and IRTAW 10 [14]. An full updated description can be found in [5].

A key feature of the Ravenscar profile is that it can be implemented by a small, reliable kernel which makes it possible to build concurrent systems with predictable timing behaviour. Two such kernels have been built up to now, Aonix' Raven [9] and UPM's Open Ravenscar Kernel (ORK) [7], demonstrating the feasibility of the concept.

In order to exploit the kernel determinism for an accurate time analysis of the applications built upon it, a set of metrics that characterize the overhead introduced by the execution is required. The Ada Language Reference Manual (ALRM) [2] Annex D specifies such a set but, as some studies have shown (see e.g. [13]), it is not sufficient for modern temporal analysis methods [6].

In the rest of this paper, the requirements for a set of metrics that enable precise timing analysis are explored, and an initial set of metrics is defined. The scope of the paper is limited to metrics related to periodic and sporadic tasks, protected procedures as interrupt handlers, Ada.Real_Time.Clock, and absolute delays, as these metrics make up a complete set which can be used in most real-time applications.

2 Modelling periodic tasks

In order to perform a precise response time analysis, a number of significant events must be taken into account. Figure 1 illustrates such events in the execution of a periodic task. The events define a set of time intervals that have to be included in the response time equations:

Clock inaccuracy: Although absolute time values are used in the delay until statement, the actual alarm setting can differ in a significant amount of time. Of course, a first cause of inaccuracy may be come from a coarse clock tick. But even for fine-grained clocks inaccuracies may arise from the fact that that hardware timers are usually programmed in a relative way [15]. If this is the case, the absolute time parameter of the delay until statement is translated into a relative delay. This may result in a delayed activation (nT' in figure 1). The effect of the delayed activation can be modelled as a delay jitter, which in the worst case equals the difference between the upper and lower bound of the time calculation error.

In some architectures, this kind of calculation usually involves complex operations, such as divide and multiply, which may require a variable number of processor cycles to execute [10, 12]. However, computers with large hardware timers (64 bits long and above) can always use absolute time, as shown in [15], which

*This work has been funded by ESA/ESTEC contract no. No.13863/99/NL/MV and by CICYT project TIC99-1043.

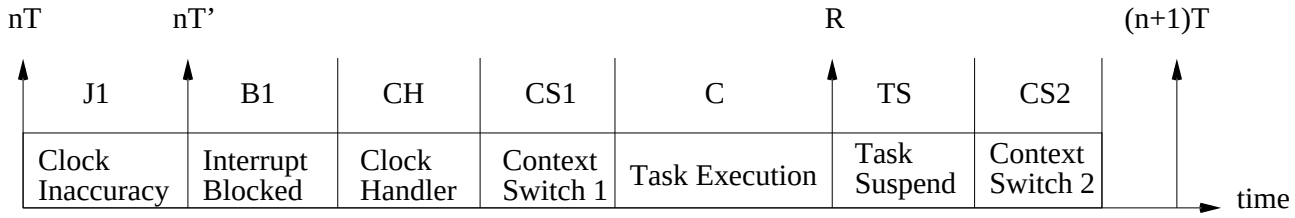

Figure 1. Events in a periodic task execution cycle.

means that clock inaccuracies can be avoided. Of course, if this is not the case, its upper bound value shall be documented.

Interrupt blocked: The acknowledgment of clock interrupts can be temporarily disabled by the kernel when performing critical sections such as context switches. The maximum elapsed time that the kernel executes with the acknowledgment of interrupts disabled is taken into account in the response time equations by a blocking factor.

The maximum value of the elapsed time for interrupt blocking shall be documented.

Clock handler: The clock handler executes when the clock interrupt is acknowledged by the processor. Modeling the clock handler depends on the approach used to support the delays and Ada.Real_Time.Clock. A Ravenscar-compliant kernel should use an interval timer for absolute delays as this approach enables accurate implementations of the clock tick and low handler overhead [15]. The modelling of the clock handler is dealt with in section 4.

Context switch 1: The clock handler execution moves the periodic task from the delay queue to the ready queue. Assuming that the task is the highest priority ready task, a context switch is performed in order to start task execution[1].

The maximum value of the context switch time shall be documented.

Task execution: The code of the task is executed. The task may be preempted a number of times by higher priority tasks or events[2]. Of course, the worst case execution time (WCET) of the code must be computed in order to perform the response time analysis. The significant response time (R in figure 1) is the time at which the code of the task finishes its execution.

[1]Notice that if task is not the first one in the ready queue, the time it takes until it is dispatched for execution is counted as an interference in the response time equation [6].

[2]This is also counted as interference time.

Ideally, the development system computes the value of the task WCET, or otherwise it can be measured in a number of ways.

Task suspension: After the execution of its code the task gets blocked until the next activation time. The operations involved in this phase are the computation of the next activation time and a delay until statement:

```
Next_Time:= Next_Time + Period;
delay until Next_Time;
```

The computation of the next activation time involves a sum of a Time and a Time_Span object. The execution of a delay until statement includes moving the task from the ready queue and inserting it into the delay queue. This operation takes a different amount of time depending on the position where the task must be inserted in the delay queue. The worst case insertion time depends on the maximum queue length, which is equal to the number of periodic tasks in the application.

The overhead of the delay until statement shall be documented.

Context Switch 2: A context switch must be performed in order to transfer the execution to the highest priority ready task (it may be an idle task).

Assuming that all of the above times are significant, the response time equation for a periodic task is as follows:

$$\begin{aligned} R_i = \quad & CS^1 + C_i + B_i \\ & + \sum_{j \in hp(i)} \left\lceil \frac{R_i + J^1}{T_j} \right\rceil (CS^1 + TS + CS^2 + C_j) \\ R_i = \quad & R_i + J^1 \end{aligned} \tag{1}$$

where J^1 is the release jitter due to clock inaccuracy, CS^1 and CS^2 the costs of both context switches, and TS is the cost of the task suspension. B^1 has to be taken into account when the term B_i is computed. The Ravenscar profile specifies the use Ceiling_Locking as the Locking_Policy, and therefore the blocking periods are not cumulative.

3 Modelling sporadic tasks

In the framework of the Ravenscar profile, sporadic tasks are implemented by using a protected object with one entry. A sporadic tasks gets initially blocked by making an entry call on an entry with a closed barrier, and can then be activated by another task or an interrupt which calls a protected procedure that opens the barrier. Figure 2 illustrates the events in the execution of an interrupt activated sporadic task.

The ALRM allows the task that opens the barrier to complete the execution of the protected entry in behalf of the waiting task. This is called the "proxy model", which is used, among others, by the GNAT compiler [1] that is integrated with the ORK kernel [8]. Under this model, after the protected procedure handler is executed, the barrier is evaluated, the sporadic task is moved from the entry queue to the ready queue, and the code of the protected entry is also executed in the context of the interrupt handler. Figure 2 assumes this scenario.

The events in the execution of a sporadic task have the same meaning for response time analysis that the corresponding ones in the execution of a periodic task, although the task suspension mechanisms are different in both cases.

Task suspend: As for periodic tasks, the task gets blocked until the next activation. The sporadic task makes a call to a protected entry. Then the kernel has to move the task from ready queue to the queue associated to the entry of the corresponding protected object. However, the dynamic semantic of the Ravenscar profile does not allow to make an entry call on an entry that already has a queued call (i.e. the maximum queue length is one). This simplification maintains the worst case execution time of this operation low.

As a result, the scheduling equation for response time is:

$$R_i = CS^1 + C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (CS^1 + TS + CS^2 + C_j) \quad (2)$$

where TS is now the cost of suspending the sporadic task. Also, B^1 has to be taken into account when the term B_i is calculated.

If the clock inaccuracy is not significant equations (1) and (2) are identical.

4 Modelling the clock interrupts

In order to support Ada.Real_Time.Clock and periodic tasks the kernel relies on hardware timers that generate interrupts. As shown in [15], a Ravenscar-compliant kernel should use an interval timer which interrupts only when a periodic process has to be released. It has been argued than a periodic timer interrupting on every clock tick is much easier to model for response time analysis purposes. However, if the set of periodic tasks require an accurate timing, using a periodic interrupter will lead to high overheads. Therefore, ORK uses an interval timer to support absolute delays.

On the other hand, the kernel must maintain Ada.Real_Time.Clock. This means that the kernel has to program the interval timer for its maximum interval when there are no closer periodic activation alarms. As a result, two kinds of clock interrupts must be taken into account: periodic interrupts and demanded interrupts.

The maximum number of periodic interrupts that a process may suffer is the ceiling of the rate between its response time and the maximum period of the interval timer. Therefore, the following term must be added to the previous equations (1) and (2):

$$\left\lceil \frac{R_i}{T_{periodic}} \right\rceil CH_{periodic} \quad (3)$$

where $T_{periodic}$ is the periodic interrupt period and $CH_{periodic}$ is the cost of handling the periodic interrupt.

The maximum number of demanded interrupts which a process may suffer is equal to the number of lower priority periodic tasks plus the ceiling of the rate between its response time and the period for every higher priority periodic task. Therefore, the following term must also be added to the previous equations (1) and (2):

$$\sum_{j \in hp_{periodic}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times CH_{demanded} + \sum_{k \in lp_{periodic}(i)} CH_{demanded} \quad (4)$$

where $CH_{demanded}$ is the cost of handling the demanded interrupt.

Of course, this is quite pessimistic because if there are enough demanded interrupts, the periodic ones may be not needed. A less pessimistic approach tries to identify the number of periodic interrupts needed.

However, some kernels, such as ORK for ERC32 [7], use two timers because it is very difficult to keep a monotonic clock without drifts with just an interval timer [15]. Then, a periodic timer is used to maintain Ada.Real_Time.Clock, and an interval timer is used to release periodic tasks when needed. In these cases, the modelling of the clock is exactly done with the above equations.

5 Modelling other interrupts

When there are sporadic tasks which are released by interrupts, additional priority inversion can occur. This priority inversion is due to the fact that the interrupt handlers

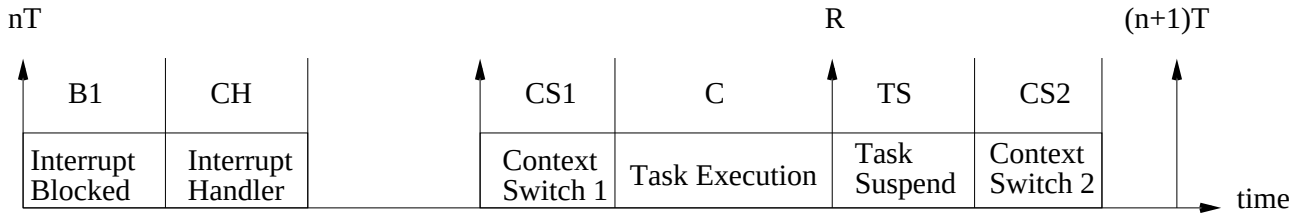

Figure 2. Events in a sporadic task execution cycle.

must be executed at least at the corresponding hardware priority level. Otherwise, the program is erroneous as the ALRM Annex C estates.

The allowed hardware priority levels are specified by the System.Interrupt_Priority range. Ordinary tasks execute at priority levelness within the System.Priority range. Therefore, if a sporadic task has a low priority, a priority inversion may occur. This kind priority inversion can be modelled as an interference from the interrupt handlers.

Therefore, the following term must also be added to the previous equations (1) and (2):

$$\sum_{l \in hp_{interrupt}(i)} \left\lceil \frac{R_i}{T_l} \right\rceil \times CH_l \qquad (5)$$

where CH_l is the cost of handling the interrupt, $hp_{interrupt}$ is the set of sporadic tasks which are released by higher priority interrupts, and T_l is the minimum interarrival time of the corresponding sporadic task.

It must be noticed that some interrupts could have a lower priority than the task being analyzed, as the base priority of task τ_i may be in the Interrupt_Priority range. In this case, the hardware interrupt is really masked and will be not acknowledged during the execution of the task.

The cost of handling the interrupt includes the execution of the preamble, the code of the protected procedure handler and the epilogue. Moreover, additional operations are performed with the proxy model, such as evaluating the barrier, moving the sporadic task from the entry queue to the ready queue, and executing the code of the protected entry. However, the Ravenscar profile only allows at most one entry with a simple boolean barrier. Therefore, the additional operations are simplified by the Ravenscar profile restrictions.

6 Conclusions

Response time analysis models have been adapted in order to allow for precise temporal analysis of Ravenscar-compliant real-time applications. The analysis includes a list of metrics for the run-time system operations that should be documented for all Ravenscar-compliant Ada development systems. We are currently working on providing the full set of metrics for the GNAT/ORK compilation system, including the versions for ERC-32 and PC architectures.

7 Acknowledgments

We gratefully thank the rest of the ORK development team: José Ruiz, Rodrigo García, Ramón Fernández, Alejandro Alosno, and Pedro Palomo. Jorge Amador and Tullio Vardanega, from the European Space Agency, have also contributed with their helpful suggestions.

References

[1] Ada Core Technologies. *GNAT Reference Manual. Version 3.13*, March 2000.

[2] *Ada 95 Reference Manual: Language and Standard Libraries. International Standard ANSI/ISO/IEC-8652:1995*, 1995. Available from Springer-Verlag, LNCS no. 1246.

[3] L. Asplund, B. Johnson, and K. Lundqvist. Session summary: The Ravenscar profile and implementation issues. *Ada Letters*, XIX(25):12–14, 1999. Proceedings of the 9th International Real-Time Ada Workshop.

[4] T. Baker and T. Vardanega. Session summary: Tasking profiles. *Ada Letters*, XVII(5):5–7, 1997. Proceedings of the 8th International Ada Real-Time Workshop.

[5] A. Burns. The Ravenscar profile. Technical report, University of York, 2000. Available at `www.cs.york.ac.uk/~burns/ravenscar.ps`.

[6] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, 3 edition, 2001.

[7] J. A. de la Puente, J. F. Ruiz, and J. Zamorano. An open Ravenscar real-time kernel for GNAT. In H. B. Keller and E. Ploedereder, editors, *Reliable Software Technologies — Ada-Europe 2000*, number 1845 in LNCS, pages 5–15. Springer-Verlag, 2000.

[8] J. A. de la Puente, J. F. Ruiz, J. Zamorano, R. García, and R. Fernández-Marina. ORK: An open source real-time kernel for on-board software systems. In *DASIA 2000 - Data Systems in Aerospace*, Montreal, Canada, May 2000.

[9] B. Dobbing and G. Romanski. The Ravenscar profile: Experience report. *Ada Letters*, XIX(2):28–32, 1999. Proceedings of the 9th International Real-Time Ada Workshop.

[10] INTEL. *i486 User's Manual*. Intel, 1989.

[11] ISO/IEC/JTC1/SC22/WG9. *Guide for the use of the Ada Programming Language in High Integrity Systems*, 2000. ISO/IEC TR 15942:2000.

[12] TEMIC. *SPARC V7 Instruction Set Manual*, 1996.

[13] T. Vardanega. Development of on-board embedded real-time systems: An engineering approach. Technical Report ESA STR-260, European Space Agency, 1999.

[14] A. Wellings. 10th International Real-Time Ada Workshop — Session summary: Status and future of the Ravenscar profile. *Ada Letters*, XXI(1), March 2001.

[15] J. Zamorano, J. F. Ruiz, and J. A. de la Puente. Implementing Ada.Real_Time.Clock and absolute delays in real-time kernels. In A. Strohmeier and D. Craeynest, editors, *Reliable Software Technologies — Ada-Europe 2001*, number 2043 in LNCS, pages 317–327. Springer-Verlag, 2001.