Real-Time Schedulability Analysis for Ada

Frank Mueller

Humboldt University Berlin, Institut f. Informatik, Unter den Linden 6, 10099 Berlin, Germany, mueller@informatik.hu-berlin.de

Abstract. This work describes the design and implementation of a tool for schedulability analysis of Ada tasks. Based on the results of schedulability theory, a component-based design through the object-oriented features of Ada 95 provides portability, composability and extensibility together with simplicity for this tool. It is part of a larger timing analysis framework that works in conjunction with the Gnat compiler. This framework provides the means to make schedulability analysis an integral part of the software development cycle of real-time applications.

1 Introduction

The language Ada was designed as a general-purpose language but it distinguishes itself from other general-purpose languages by its facilities for real-time and embedded systems. Ada provides support for real-time systems on the language level through tasks, synchronization, time facilities and the real-time annex [8]. Ada implementations often cater to real-time and embedded systems by providing cross compilation for embedded architectures and tools for program analysis of real-time applications. In the following, a system adhering to the real-time annex in terms of scheduling and locking policies as well as the Ravenscar Ada 95 subset [3] are assumed.

This work introduces a tool for real-time schedulability analysis. It calculates if a task set will meet its deadlines for a given scheduling paradigm. The tool is designed for extensibility in terms of new protocols as well as additions to existing protocols. Features of protocols can be combined to model a compound scheduling model by using object-like components as building blocks for such models. This is demonstrated by the addition of priority ceiling and inheritance protocols to arbitrary base protocols. The tool is implemented in Ada and supports rate-monotone analysis (RMA), response-time analysis (RTA) and earliest deadline first (EDF) analysis [10, 2]. It covers analysis with and without resource allocations, the latter optionally including protocols to avoid priority inversion. The tool is applicable to schedulability analysis of Ada tasks but is not necessarily limited to Ada. It is designed for integral compilation and schedulability analysis in the software development cycle of real-time systems.

The work is aimed at real-time scheduling in a uniprocessor environment where schedulability theory has supplied firm results that can be applied during the software development cycle of real-time applications. The target systems are hard real-time systems with firm deadlines or soft real-time systems with subsets of hard tasks, the latter possibly due to economic reasons rather than the risks of a system on its environment, as given in traditional hard real-time systems. The application area are systems with preemptive and strict priority scheduling and given task parameters, such as periods, worst-case execution times and resource requirements, just to name a few.

The paper is structures as follows: Section 2 describes the fundamental design issues and the overall methodology and motivation for this work. Section 3 explains aspects of the implementation and specifics related to Ada. Section 4 discusses the applicability of this work to Ada tasks. Section 5 contrasts this work with previous work. Section 6 concludes this paper.

2 Methodology and Design

The design of the tool for schedulability analysis was motivated by the following aims: simplicity, detailed analysis, portability, composability, extensibility and its ability to be integrated into a larger analysis environment. These issues were addressed as follows:

Simplicity: The result of the analysis either confirms the schedulability of a task set or rejects it, *i.e.*, the user receives a yes/no answer. In some cases, schedulability analysis may remain inconclusive but in this case other schedulability methods are suggested to yield conclusive results.

Detailed analysis: Optionally, the internal results of analyzing the task set are reported. This allows a user to asses how close to exceeding its schedulability bounds a task set is. It provides the means to report which task caused a task set not to be schedulable. The results may also be helpful for tuning purposes.

Portability: The analysis is performed on an set of tasks with given parameters that abstract from the implementation language, operating system and architecture.

Composability: Analysis methods are encapsulated into different components that can be combined to form a model for a scheduling framework with certain protocols, *e.g.*, for response-time analysis with resources and the priority inheritance protocol.

Extensibility: The interfaces of the analysis components serve as a contract for the addition of new protocols or methods for schedulability analysis. They can then be combined with existing methods and protocols.

Integration: The tool accepts the specification of the task set as a human readable input and also produces simple output, which is suitable for an integration into a larger analysis framework for automatically generated input and output potentially forwarded to a graphical user interface.

Schedulability analysis is based on the specification of a task set and a scheduling method. This specification contains two parts. A first section describes general settings while a second section specifies the parameters of each task.

General settings: This information applies to the entire task specification.

- The method for the schedulability analysis can be selected. Current methods include RMA, EDF and RTA.
- The time unit can be set. This unit applies for all further numerical specifications of times.
- The resource protocol can be selected among priority ceiling and priority inheritance. Alternatively, no resource protocol can be selected, which implies that priority inversion may occur and will be accounted for during schedulability analysis.

Task Parameters: For each task, a number of parameters are expected.

- The task number simply serves as a naming reference to relate the output of the schedulability analysis to a specific task.
- The period describes the rate of reinvocation of the task.
- The deadline specifies the latest point in time that a task's execution must have completed. The deadline is always interpreted relative to the previous invocation time.
- The worst-case execution time (WCET) represents the maximum duration that the task may need to execute. It must account for the duration from an invocation to the completion of a task along the longest possible execution path. The computation time may be specified as a singleton or as a list of n elements, where each element specifies the worst-case duration for a specific phase of the task's execution. In each phase, a task may hold different resources.
- The resource list is a list of n elements, *i.e.*, it has the same length as the WCET list. Each element corresponds to a distinct phase and specifies a set of resources that the task holds during this phase. The empty set indicates a phase without resources.

Based on this description of a task set and the analysis method, the tool performs the requested schedulability analysis, reports the results and, in case of a negative outcome, may suggest alternative analysis methods for the same task set.

The following symbols are used for schedulability analysis:

D_i - Deadline of task i;
T_i - Period of task i;
C_i - worst-case execution (or computation) time of task i;
P_i - Priority of task i;
R_i - Response time of task i;
B_i - Blocking time (or factor) of task i.

Depending on the chosen method, schedulability analysis is based on the following theoretical results:

RMA: Rate-monotone analysis is based on a rate-monotone priority assignment, *i.e.*, shorter periods imply higher priority (higher urgency). Schedulability of the task set is determined by checking on the system utilization calculated as the sum of each task's contribution to the utilization [10]:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1).$$

In addition, the critical zone is constructed, *i.e.*, a scheduling plan is given for the instance where all tasks are released at time 0. The critical zone reaches from time 0 to either the point in time where all tasks have completed execution at least once or where a task has missed its deadline. In the presence of resources, the blocking time is included in the schedulability test and the test is applied for each task separately [5]:

$$\forall i, 1 \leq i \leq n, \ \sum_{k=1}^{i} \frac{C_k}{T_k} + \frac{B_i}{T_i} \leq i(2^{\frac{1}{i}} - 1).$$

If the main inequality holds, the task set is schedulable. When system utilization exceeds 100%, the task set is not schedulable. However, if utilization exceeds the right-hand term of the inequation but is less or equal 100%, the test remains inconclusive. Other analysis methods, such as RTA, need to be applied.

EDF: For earliest-deadline-first scheduling, the following schedulability test is performed [10]:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1.$$

This test allows a maximum utilization of 100%. In the presence of resources, the blocking time is included in the schedulability test and the test is applied for each task separately [13]: For priority inheritance

$$\forall i, 1 \leq i \leq n, \ \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq 1$$

and for priority ceiling

$$\sum_{i=1}^{n} \frac{C_i + B_i}{T_i} \leq 1.$$

RTA: Priorities are assigned in a deadline-monotone fashion. The schedulability test uses a fixed-point algorithm based on the following formulas [2]:

$$R_i^0 = C_i$$

$$R_i^{k+1} = C_i + \sum_{j=1}^{i-1} \lceil \frac{R_i^k}{T_j} \rceil C_j.$$

If R_i^k equals R_i^{k+1}, then the fixed point is reached and the response time of task i has been determined. If the response time does not exceed the task's deadline, then the task is schedulable. In the presence of resource constraints the formula is augmented by a blocking factor [2, 5]:

$$R_i^{k+1} = C_i + B_i + \sum_{j=1}^{i-1} \lceil \frac{R_i^k}{T_j} \rceil C_j.$$

RTA provides conclusive results to determine if a task set if schedulable where RMA may be inconclusive or if priorities are not assigned in a rate-monotone fashion.

Blocking factors are optional and will be omitted in the absence of resources. Priority assignment occurs automatically for the above methods unless specified otherwise. In other words, a user does not have to be concerned about priorities as long as the traditional methods are adequate to describe a real-time application. Interrupts may be modeled by specifically assigning high priorities to the task that represents an interrupt handler. This priority assignment may violate the rate-monotone or deadline-monotone assignment. This does not pose any problems for RTA. For RMA, on the other hand, it is treated separately by an extension of the above formulas along the lines of the rate-monotone handbook [9].

Resources are taken into account by determining a blocking time for tasks covering delays incurred by lower priority tasks engaged in critical sections that cause a higher priority task to suspend until resources become available. Blocking times are based on the duration $D_{j,k}$ of "the longest critical section of task j among those guarded by semaphore S_k" [5]. These durations can be derived from the worst-case execution time of the critical sections protected by the same semaphore. Let us assume there are m semaphores. Each semaphore has a ceiling priority $C(S_k)$. The blocking times are calculated as follows (depending on the priority protocol) [5]:

Priority Ceiling Protocol: The ceiling $C(S_k)$ of a semaphore is the maximum of the priorities of tasks using this semaphore. Then, the blocking time of task i is given by:

$$B_i = \max_{j,k}[D_{j,k} : T_j < T_i \wedge C(S_k) \geq T_i].$$

Priority Inheritance Protocol: The ceiling $C(S_k)$ of a semaphore is the maximum of the priorities of tasks using this semaphore or of any outer ceiling $C(S_l)$ for a semaphore S_l that is requested before S_k is requested (S_k nested within S_l), which ever one is larger. Then, the blocking time of task i is given by:

$$B_i = min(B_i^l, B_i^s)$$

where

$$B_i^l = \sum_{j=i+1}^{n} \max_k [D_{j,k} : C(S_k) \geq P_i]$$

and

$$B_i^s = \sum_{k=1}^{m} \max_{j>1} [D_{j,k} : C(S_k) \geq P_i].$$

The term B_i^l determines the blocking for *all tasks* with lower priority using common semaphores. The term B_i^s represents the blocking for *all semaphores* that are being used by lower priority tasks. Hence, the former may overestimate the actual blocking by taking blocking on the same semaphore by multiple tasks into account while the latter may result in overestimation due to blocking by the same tasks on distinct and non-nested semaphores. The minimum of these two values, therefore, represents a safe estimate for the blocking factor of a task i. For nested critical sections, we require that the durations $D_{j,k}$ of outer semaphore include the durations due to inner semaphores in a transitive fashion. An example will be given later on.

No Protocol: In the absence of such a protocol, priority inversion may cause a lower priority task to block a higher one for the duration of its entire execution. These long delays are prone to cause schedulability problems where one of the above protocols still allows a task set to be schedulable. Nonetheless, in the absence of the above protocols within a scheduling framework, priority inversion can be included in the calculation of the blocking times. This allows schedulability analysis to be performed as usual [9].

3 Implementation

The methods for schedulability analysis are applied to a task set by determining the priority of a task and calculating the utilization or response time, depending on the analysis method. The priorities are based on rate-monotone assignments for RMA and arbitrary priority assignments for RTA, *e.g.*, deadline-monotone assignments, *i.e.* tasks with smaller values statically receive higher urgency (priority). For EDF, the task closest to its deadline during runtime (dynamically) receives the highest priority. The calculations for schedulability may involve fixed-point algorithms that either iterate until they reach a stable state or they determine that a deadline is violated and terminate prematurely. The blocking times are also calculated by a fixed-point algorithm and are subject to the same constraints. In addition, an event-based simulation of a scheduling scenario can be constructed for a task set and a set of release times. This discrete event modeling provides an efficient way to simulate scheduling actions, which are then recorded for later replay by the user. Figure 1 depicts the Ada code for the calculation of critical zones. For a rate-monotone priority assignment, the critical

```
begin -> procedure Compute_Crit_Time_Zone.
    loop
        Switch(Current_Pos).Whenn := Time_Ptr;
        Switch(Current_Pos).Which := Current_Task;
        if Task_Data(Current).Suspended and
                Task_Data(Current).To_Run /=
                Task_Data(Current).Wcet_Sum then
            Switch(Current_Pos).What := Continued; -> task not completed
        else
            Switch(Current_Pos).What := Started;
        end if;
        Current_Pos := Current_Pos + 1;
        Next := Get_Next; -> get next task to be running.
        if Exists_Suspending_Task then
            -> other task which preempts CURRENT?
            Passed_Time := Task_Data(Next).Start - Time_Ptr;
            Task_Data(Current).To_Run := Task_Data(Current).To_Run -
                                         Passed_Time;
            Task_Data(Current).Suspended := True;
            Time_Ptr := Time_Ptr + Passed_Time;
            Switch(Current_Pos).What := Suspended;
            Exists_Suspending_Task := False;
        else
            Time_Ptr := Time_Ptr + Task_Data(Current).To_Run;
            Task_Data(Current).To_Run := Task_Data(Current).Wcet_Sum;
            Task_Data(Current).Suspended := False;
            Task_Data(Current).Times := Task_Data(Current).Times + 1.0;
            Task_Data(Current).Start := Task_Data(Current).Times *
                                        Task_Data(Current).Period;
            Switch(Current_Pos).What := Finished;
        end if;
        Switch(Current_Pos).Whenn := Time_Ptr;
        Switch(Current_Pos).Which := Current;
        Current_Pos := Current_Pos + 1;

        Correct_Starts;
        Check_Deadline_Miss(Switch, Current_Pos);

        exit when Current = N and not Task_Data(N).Suspended;
        Current := Next;
    end loop;
    if Common_Spec.Info_Output_Mode = Full or
        Common_Spec.Info_Output_Mode = Zone then
        Put_Crit_Time_Zone (Switch, Current_Pos);
    end if;
end Compute_Crit_Time_Zone;
```

Fig. 1. Calculating the Critical Zone

zone is complete when the task with the longest period (implying the lowest priority) has completed execution. If a deadline is missed before this point, then the task set is said to not be schedulable for this given scheduler.

The task set is supplied as the standard input for the analysis tool while results are reported as standard output. Figure 2 depicts an example for an RMA task set without resource as an input, where task 1 has a period and WCET of 3 and 1 and task 2 of 4 and 2, respectively. RMA remains inconclusive for this task set. The critical zone consists of a single execution of each task in this case without a missed deadline. A sequence of executions may be generated from this output by a graphical user interface, such as depicted in Figure 3. It turns out that this task set is schedulable under RTA.

Input	Output
# data file for the anatool. info output mode: full method: RMA unit: ms task: 1 period: 3 computation time: 1 task: 2 period: 4 computation time: 2	Analysis method: RMA Unit: ms task number, now: 1; in input: 1 period: 3.000 computation times: 1.000 sum of computation times: 1.000 priority: 2 task number, now: 2; in input: 2 period: 4.000 computation times: 2.000 sum of computation times: 2.000 priority: 1 at 0.000 task 1 STARTED at 1.000 task 1 FINISHED at 1.000 task 2 STARTED at 3.000 task 2 FINISHED Critical time zone ranges from 0.000 to 3.000. Utilization of 0.833 exceeds the bound of 0.828. All time data in ms. This set of tasks MAY NOT BE schedulable. RMA remains inconclusive, try RTA.

Fig. 2. Sample Execution for RMA

Figure 4 depicts the schedulability information for a task set with resources under RTA, where priorities were assigned in a deadline-monotone fashion. The

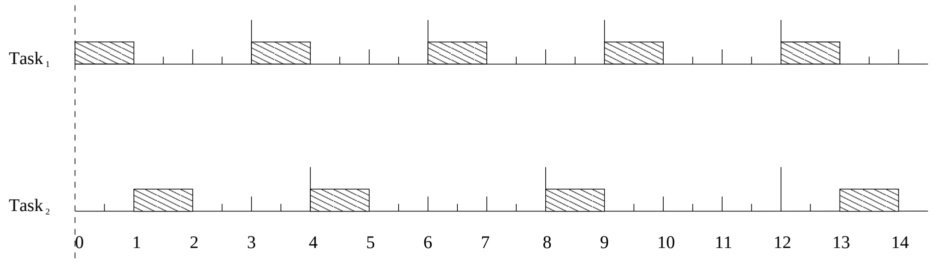

Fig. 3. Sample Schedule for RMA with Critical Time Zone

table represents the longest critical sections for pairs of tasks and resources, *i.e.*, the components $D_{j,k}$ introduced in the context of the priority ceiling and inheritance protocols, and is used to determine the blocking times (called blocking factors in the Figure). For example, the blocking time B_i^l is determined from the maximum value of each row for lower priority tasks. B_i^s is the maximum from each column of a semaphore that task i uses. The blocking times (factors) and task parameters are then reported for each task (only depicted in part in the Figure). This task set is schedulable since the response time of a task does not exceed its deadline. The verbose information is optional and can be suppressed by omitting the full mode as an input/output mode.

4 Schedulability Analysis for Ada

The schedulability analysis is based on the specification of parameters for tasks. These tasks have to represent sequences of code without task suspension other than suspensions caused by preemption or due to resource requirements. In other words, Ada tasks may not always correspond to scheduling tasks (simply called tasks in the following) but they may often consist of a sequence of phases. Ada tasks may be subject to suspension due to synchronization, delays or due to I/O requirements. Thus, the code of an Ada task has to be partitioned into a sequence of phases with consecutive start times and deadlines. Voluntary delays and I/O delays may be accounted for by offsets of the start time relative to the previous deadline in a sequence of phases. Ada tasks must be statically created rather than dynamically and execute an infinite loop whose body represents the sequence of phases. The worst-case execution time for each phase has to be specified as part of the input for the schedulability analyzer.

Synchronization requires further work. Ada 95 protected objects (excluding protected entries) can simply be treated as resources within the existing scheduling frameworks. They represent the common case of asynchronous communication already accounted for in the models. Such restrictions on tasking and synchronization are consistent with the requirements imposed by the Ravenscar subset of Ada 95 [3]. The Ravenscar profile also allows restricted use of protected

Input	Output
# data file for the anatool. method: RTA	Analysis method: RTA
unit: ms	Unit: ms
resource access protocol: inheritance	Resource access protocol: INHERITANCE
info output mode: full	
	R1[4] — R2[4] — R3[3] —
task: 1	————————————-
period: 40	1— 1.000 2.000 0.000
deadline: 40	2— 0.000 9.000 3.000
computation time: 1.0, 1.0, 2.0, 2.0	3— 8.000 7.000 0.000
resource: {0}, {1}, {2}, {0}	4— 6.000 5.000 4.000
task: 2	blocking factor task 1: 17.000
period: 80	blocking factor task 2: 14.000
deadline: 80	blocking factor task 3: 6.000
computation time: 1.0, 9.0, 3.0, 3.0	blocking factor task 4: 0.000
resource: {0}, {2}, {3}, {0}	
	task number: 1
task: 3	period: 40.000
period: 100	deadline: 40.000
deadline: 100	computation times: 1.000, 1.000, 2.000, 2.000
computation time: 1, 8, 7, 3	sum of computation times: 6.000 to 3.000.
resource: {0}, {1}, {2}, {0}	resources: {0}, {1}, {2}, {0}
	priority: 4
task: 4	[...]
period: 120	response time task 1: 23.000
deadline: 120	response time task 2: 36.000
computation time: 4, 6, 5, 4, 2	response time task 3: 53.000
resource: {0}, {1}, {2}, {3}, {0}	response time task 4: 68.000
	All time data in ms.
	This set of tasks IS schedulable with RTA.

Fig. 4. Sample Execution for RTA

entries for simple task synchronization, which may be represented by partial ordering of task phases as discussed in the next section.

Rendezvous, on the other hand, represent synchronous communication. Either the caller or the receiver (acceptor) may be stalled on a rendezvous, depending on the dynamic schedule. This scenario cannot simply be modeled as resources. A restricted solution was suggested by handling rendezvous as synchronous sessions [4]. This model requires that different rendezvous may only occur in a certain order that allows each rendezvous in sequence to be assigned priorities that are strictly monotonically decreasing.

We are currently working on integrating the schedulability analysis into our existing timing analysis framework. We use a set of tools to derive the WCET of arbitrary code sections from the source program, as depicted in Figure 5.

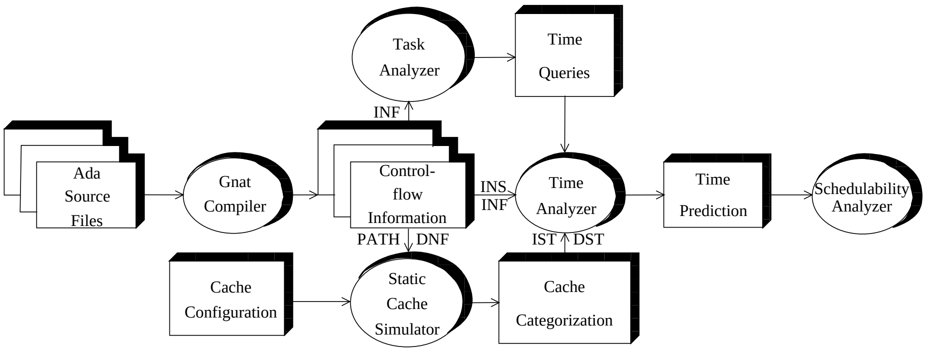

Fig. 5. Toolset for Timing Analysis

The information required for WCET prediction is extracted by a modified version of the Gnat compiler [16] and then fed to a cache analyzer [12, 17], a pipeline simulator and a path analysis tool [6]. The missing link is a task analyzer whose objective is to determine the phases for each Ada task. Phases are separated by synchronization points, which are well-defined in the Ada 95 language [8]. The timing analysis tool is then requested to report the WCET of each phase. This information, together with the specification of the task set, provides the input to the schedulability analysis. Overall, the framework allows software development of real-time systems with an integrated timing analysis. Compilation of software components may be accompanied by an automatic check to determine if timing requirements can still be met. If timing violations are detected, proper steps can be taken during software development, ranging from reverting changes that caused deadline overruns over activation of different scheduling methods to a partial redesign of the culprit task or, in the worst case, even a complete redesign based on insight knowledge about the reasons that caused problems.

In summary, current work includes the task analyzer and extensions to the schedulability tool to handle resources for EDF and the validation of methods for non-monotone priority assignments (*e.g*, due to interrupts) for the analysis methods.

5 Related Work

Stoyenko describes a schedulability analyzer [14] specifically targeted for the Real-Time Euclid environment but not applicable to more general schedulability problems. Caisarts [7] is an extensible, rule-based tool for schedulability analysis. It supports a window-based user interface and a large number of task parameters. Its strength is to give advice in situations where schedulability problems arise. Scheduler 1-2-3 [15] allows schedulability analysis for RMA. Perts [11] includes schedulability analysis for RMA and RTA with resources and is part of a larger framework for software development with a set of scheduling algorithms whose code can be selected from a library. Stress [1] provides schedulability analysis for a number of predefined frameworks including RMA and RTA but also allows user-defined scheduler support through a programming-language based specification, which is also used for simulations of new protocols. Our approach differs from the previous work by its component-based approach based on the object-oriented programming paradigm. In this sense, it is probably closest to the goals of Perts and Stress. The main difference to prior work is that our work aims at an integration into a larger toolset for software development of real-time systems in the context of the Gnat compiler, which may be unprecedented. Furthermore, our work builds on the results of the rate-monotone handbook [9], Buttazzo's compilation of schedulability results [5] and the approach of deadline scheduling [13]. It goes beyond the previous work in that we account for priority inversion in the absence of a priority protocol, that we support nested resources and that we allow ad-hoc priority assignments to tasks to model interrupts.

6 Conclusions

The paper described the design and implementation of a tool for schedulability analysis of Ada tasks. A the component-based design through the object-oriented features of Ada 95 provides portability, composability and extensibility together with simplicity for this tool. It is based on the results of schedulability theory, including RMA, EDF and RTA with and without resource constraints, optionally with priority ceiling and inheritance protocols. Priorities are automatically assigned to tasks according to the analysis method. In addition, the framework supports user-assigned priorities during the analysis that go beyond the automatic assignments. The schedulability tool is part of a larger timing analysis framework that works in conjunction with the Gnat compiler. This framework provides the means to make schedulability analysis an integral part of the software development cycle of real-time applications.

Acknowledgement

Oliver Bühn implemented the schedulability analysis tool in Ada 95.

References

1. N. Audsley, A. Burns, M. Richardson, and A. Wellings. STRESS: A simulator for hard real-time systems. *Software Practice & Experience*, 24(6), June 1994.
2. N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *IEEE Workshop on Real-Time Operating Systems and Software*, May 1991.
3. A. Burns, B. Dobbing, and G. Romanski. The ravenscar tasking profile for high integrity real-time programs. In *Ada Europe*, pages 263–275, 1998.
4. A. Burns and A. Wellings. Synchronous sessions and fixed priority scheduling. *Journal of Systems Architecture*, 44:107–118, 1997.
5. Giorgio C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer, 1997.
6. C. A. Healy, R. D. Arnold, F. Mueller, D. Whalley, and M. G. Harmon. Bounding pipeline and instruction cache performance. *IEEE Transactions on Computers*, 48(1):53–70, January 1999.
7. M. Humphrey and J. A. Stankovic. CAISARTS: A tool for real-time scheduling assistance. In *IEEE Real-Time Technology and Applications Symposium*, June 1996.
8. ISO/IEC. *ISO/IEC 8652L 1995 (E), Information Technology – Programming Languages – Ada*, February 1995.
9. M. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Harbour. *A Practitioner's Handbook for Real-Time Analysis: Giude to Rate Monotonic Analysis for Real-Time Systems*. Kluwer, 1993.
10. C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. of the Association for Computing Machinery*, 20(1):46–61, January 1973.
11. J. W. S. Liu, J. L. Redondo, Z. Deng, T. S. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, and W. K. Shih. PERTS: A prototyping environment for real-time systems. In Susan Davidson and Insup Lee, editors, *Proceedings of the Real-Time Systems Symposium*, pages 184–188, Raleigh-Durham, NC, December 1993. IEEE Computer Society Press.
12. F. Mueller. Timing analysis for instruction caches. *Real-Time Systems*, 18(2/3):209–239, May 2000.
13. J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo. *Deadline Scheduling for Real-Time Systems*. Kluwer, 1998.
14. A. Stoyenko. A schulability analyzer for real-time euclid. In *IEEE Real-Time Systems Symposium*, pages 218–227, December 1987.
15. Hideyuki Tokuda. Arts real-time scheduler analyzer/debugger. *IEEE Software*, 5(3):29, May 1990. [Abstract].
16. New York University. The Gnu NYU Ada Translator (GNAT). Available by anonymous FTP from cs.nyu.edu.
17. R. T. White, F. Mueller, C. Healy, D. Whalley, and M. G. Harmon. Timing analysis for data and wrap-around fill caches. *Real-Time Systems*, 17(2/3):209–233, November 1999.