Technical Report

CMU/SEI-89-TR-023
ESD-89-TR-031

Hartstone: Synthetic Benchmark Requirements for Hard Real-Time Applications

Nelson Weiderman
June 1989

Technical Report

CMU/SEI-89-TR-023
ESD-89-TR-031
June 1989

Hartstone: Synthetic Benchmark Requirements for Hard Real-Time Applications

Nelson Weiderman

Real-Time Embedded Systems Testbed Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through SAIC/ASSET: 1350 Earl L. Core Road; PO Box 3305; Morgantown, West Virginia 26505 / Phone: (304) 284-9000 / FAX: (304) 284-9001 / World Wide Web: http://www.asset.com/sei.html / e-mail: webmaster@www.asset.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8274 or toll-free in the U.S. — 1-800 225-3842).

Hartstone: Synthetic Benchmark Requirements for Hard Real-Time Applications

Abstract: The purpose of this paper is to define the operational concept for a series of benchmark requirements to be used to test the ability of a system to handle hard real-time applications. Implementations of such benchmarks would be useful in evaluating scheduling algorithms, protocols, and design paradigms, as well as processors, languages, compilers, and operating systems. Several Ada programs are under development to test standard versions of the benchmark requirements and will be released into the public domain.

1. Background and Motivation

Ada was designed to be used for time-critical embedded applications. These applications include systems that are embedded into automobiles, airplanes, ships, and robots as well as systems that monitor and control industrial processes such as steel milling and power generation. These applications typically have time constraints (deadlines) as part of their specified requirements. There has generally been a distinction made between "hard" real-time applications, in which processes must meet specified deadlines in order to satisfy the requirements of the system, and "soft" real-time, where a statistical distribution of response times is acceptable [Liu 73]. There has been significant research in real-time scheduling techniques, real-time languages and operating systems, and real-time databases [IEEE 88]. There has also been a significant amount of speculation as to whether current Ada implementations are capable of handling hard real-time applications.

The currently available evaluation technology for Ada does not adequately address deadline-driven computing. For the most part the existing test suites and checklists address the questions of functionality (does Ada provide a function), capacity (how big can various aspects of programs be), or performance (how fast can certain operations or programs be performed). None adequately addresses the question of whether a set of activities (when there may be rapid switching of attention between those activities) can meet preestablished deadlines. The answer to this question has more to do with the ability to keep accurate time and perform in a predictable, deterministic fashion than it has to do with the throughput of the system. Just as a pilot of an airplane must be chosen more for consistent reliable behavior than for raw throughput, some software systems must be evaluated more for their predictable, deterministic behavior rather than for their throughput.

One recent effort to address the lack of evaluation technology for real-time behavior has been described as a Rhealstone benchmark [Kar 89]. In their paper, Kar and Porter propose to define a metric obtained as a possibly weighted sum of six categories of activity that are deemed to be most crucial to the performance of real-time systems. These categories are: task switching time, preemption time, interrupt latency time, semaphore shuffling time, deadlock breaking time, and datagram throughput time. These concepts are defined in the paper. The idea is to add the reciprocals of the times required for these crucial operations and to use the sum as an indicator of

the number of crucial operations performed per second. With respect to Ada, many of these times are available by running tests from Ada test suites, other tests could be coded easily, and at least one (interrupt latency) may require special hardware such as a logic analyzer. This concept can be an important contribution, but like many fine-grained benchmarks, has limitations. Even under the best of circumstances, trying to predict real-time behavior of a real system based on such information is risky. Not only are such measurements subject to many sources of variation (see [Altman 87], for example), but counting the operations in real applications is difficult; and using them in combination may have unexpected performance implications.

Benchmark programs have been used for years to compare the relative speeds of computers and language implementations. One of the most important benchmarks is the Whetstone benchmark [Curnow 76]. It is a *synthetic* benchmark, which means that rather than performing any useful function, it is constructed based on the instruction frequencies of a representative set of programs. It is meant to be typical of scientific numerical computation and is heavily weighted toward floating point operations. Its importance as a benchmark is rooted in its wide acceptance and use, as well as its "controlled" status. The United Kingdom's National Physical Laboratory maintains the official versions implemented in a number of programming languages. Another synthetic benchmark is called Dhrystone [Weicker 84, Weicker 88]. It is intended to reflect the features of modern programming languages (e.g., record and pointer data types) and is intended to be typical of systems programs. The program is synthesized from static and dynamic statement frequencies gathered from 16 different studies, which analyzed large and small programs from a variety of sources. These synthetic benchmarks tend to be better accepted than individual programs such as sorts (Sieve of Eratosthenes), recursive programs (Ackermann's function), or games (Eight Queens). They also have the advantage that their results can be expressed as a more meaningful metric (e.g., Kilo Whetstone Instructions Per Second) rather than as a simple time for execution.

What would be helpful for evaluating systems for hard real-time applications is a representative family of synthetic benchmark requirements. Implementation of these requirements will then help us to determine which component in a system (algorithm, compiler implementation, operating system, hardware component) is the most critical for continued improvement in building real-time systems. For example, such a standard set of requirements could help to identify changes in performance due to hardware, software, or algorithms. If we keep the system constant while varying the algorithm, we have an apples-to-apples comparison of the algorithms. If we hold the algorithm constant while varying the loading of the benchmark, we will be able to observe the stability of the algorithm under load. If we keep the algorithm constant and change the language implementation, we will be able to compare the performance of various compilers for that language. If we change the language or operating system, we can observe the influence of those components. Hence what we wish to define first is the *requirements* for a synthetic application, not the design or the implementation of a solution for those requirements. Furthermore the success of such a family of benchmark requirements will be dependent on the acceptance and use by the community. If the success of Whetstone and Dhrystone are any indication, there is a good chance that such a family would serve a useful purpose to both researchers and practitioners. For Ada users, the language implementation and the scheduling model are the components of greatest interest.

The purpose of this paper is to define the operational concept for a series of benchmark requirements to be used to test the ability of a system to handle hard real-time applications. In the tradition of Whetstone and Dhrystone, we will call these the "Hartstone" benchmarks, where the "Hart" is derived from Hard Real-Time. These will be synthetic benchmarks in the spirit of Whetstone and Dhrystone, and their definition will make it possible to compare a number of hardware/software/algorithm architectures. Several programs written in Ada for a specific machine configurations will be available from the SEI as examples of implementations of the requirements.

2. General Benchmark Requirements

The Hartstone benchmark requirements define several series of tests. The initial requirement of a given series will provide a baseline from which the others are derived. Each test in a series either succeeds (meets its real-time deadlines) or fails (misses one or more deadlines during the test). Among the characteristics of the requirements are the following:

- **Spanning hard real-time problem domain:** First and foremost, the requirements should be representative of the problem domain for which it is designed. Hard real-time applications are characterized by periodic activities, aperiodic processing generated by interrupts or user interaction, synchronization between activities, access to common data, mode changes, and distributed processors. It is required that the Hartstone be as faithful as possible to actual real-world applications. However, some of these characteristics will not be incorporated in the simplest requirements.
- **Increasing complexity:** The requirements will increase in complexity. It is presumed that these will be implemented from simple to complex.
- **Stress testing:** Each individual test series will be structured to have a base requirement and a strategy for modifying that requirement to stress the system to its limit along a number of dimensions. These dimensions would include, for example, the processing load, number of periodic or aperiodic activities, and the frequency of the activities. Each family of tests would have a "figure of merit" that would be an indication of the amount of useful work achieved during the tests.
- **Self-verifying:** Each individual test should verify that the computations are being performed correctly and that the deadlines are being met.
- **Synthetic workload:** A synthetic load guarantees that an equal amount of work is being done on each system running the benchmarks independent of hardware and software. The synthetic load should be representative of work that may be done in real-time systems. The amount of work should be variable to facilitate benchmark programs that more closely match different application environments.
- **Relative figure of merit:** The metric indicating the result of the test series should be a relative rather than absolute figure that clearly distinguishes the productive workload achieved from the overhead of scheduling, task switching, task synchronization, keeping time, and interrupt handling. It is more important to know the maximum utilization achievable before deadlines start to be missed than it is to know the raw throughput of the system. The relative figure can be used to estimate the throughput for larger or smaller systems.

There are several good reasons to choose either Whetstone or Dhrystone as the synthetic load for these experiments:

1. They are well-defined and implemented in several different languages.
2. The raw speed of the hardware and runtime system (in terms of Whetstones or Dhrystones) can be used to derive the overhead of the scheduling algorithm and the scheduling implementation. The ratio of the actual throughput to the raw throughput will indicate how much time is lost due to various overheads and blocking.
3. Each thousand Whetstones or Dhrystones takes on the order of a few milliseconds for most modern machines (usually less than 10 milliseconds). This makes it possible to consider frequencies up to and beyond 100 Hz for a single activity requiring 1000 Whetstones or Dhrystones.
4. The Whetstone and Dhrystone benchmarks are instruction mixes that are accepted in the community as representative of a broad spectrum of computing applications. Whetstones are characteristic of scientific programming, and Dhrystones are characteristic of systems programming.

We have decided to use the Whetstone as the basis for Hartstone because it may be more representative of the numerical and scientific nature of the problem domain. In addition, Brian Wichmann of the National Physical Laboratory has provided us with a revised version of Whetstone, which is self-validating and which will allow us to run 1 Kilo Whetstone Instructions (KWI) at a time [Wichmann 88]. Wichmann's version is written in Pascal and has been translated to Ada.

3. Specific Requirements

The purpose of this section is to outline the requirements for five test series and then define the requirements of one test series in some detail. For the series defined in detail, the goal is to provide a description that is specific enough so that there are no ambiguities about what is to be implemented, but general enough that the implementation can be done in a variety of languages on a variety of systems. For the general descriptions, the goal is to provide the notion of extending the initial requirement so that it is more applicable to real systems. In all cases, the design of the solutions should be completely system dependent. We would like to see the benchmark suite implemented in different languages, and for various operating systems and machines, as well as using various design paradigms.

In the discussions that follow, we refer to an activity as a "task." This is merely a convenient term for talking about a thread of control or process. It is not meant to imply that the Hartstone must be executed in a concurrent language. Nor is it meant to imply that, for comparison purposes, Hartstones should not be implemented using a cyclic executive or any other paradigm of scheduling. Similarly, nothing should be construed to rule out implementation on a distributed system.

We will define the following categories of tests in increasing order of difficulty.

3.1. PH Series: Periodic Tasks, Harmonic Frequencies

The objective of the PH Series is to provide test requirements with a set of tasks that are periodic and harmonic. This means that each task in the system runs at precise regular intervals, and the frequency of each task is an integral multiple of all tasks with lower frequencies. For example, a task set with four tasks running at frequencies of 1Hz, 5Hz, 10Hz, and 20Hz would be a harmonic task set. The significance of harmonic frequencies is that they are found widely in real-time applications and are the easiest to handle using a number of designs and scheduling algorithms.

3.2. PN Series: Periodic Tasks, Non-Harmonic Frequencies

The objective of the PN Series is to provide test requirements with a set of tasks that are periodic, but non-harmonic. This series represents application domains similar to the PH Series, but in which the frequencies are chosen to match the application requirements (natural frequencies of physical phenomena) rather than the implementation requirements (frequencies required by hardware or implementation details). This task set is not particularly amenable to a cyclic executive design and will have low scheduling theoretic utilization (utilization level at which deadlines can be guaranteed) for a rate monotonic scheduler [Liu 73].

3.3. AH Series: PH Series with Aperiodic Processing Added

The objective of the AH Series is to provide test requirements with a background set of periodic and harmonic tasks and a foreground set of interrupt-driven aperiodic tasks. This series represents application domains in which the system responds to external events. For example, combat control systems and user interfaces are often interrupt driven. The aperiodic tasks will be characterized by interarrival times that are taken from statistical distributions and may or may not have deadlines. It is important to minimize the response time while still meeting all the deadlines of the background task set. A number of algorithms have recently been proposed for improving response time in this context [Lehoczky 87]. This task set is amenable to both cyclic executives as well as data driven executives.

3.4. SH Series: PH Series with Synchronization

The objective of the SH Series is to provide test requirements with a set of tasks that require synchronization among the tasks. Synchronization introduces the possibility of task blocking and a rich set of priority inversion possibilities [Cornhill 87]. This task set can be used to test the effectiveness of the various algorithms and protocols that deal with priority inversions. It will also be useful to investigate the efficiency of various synchronization mechanisms.

3.5. SA Series: PH Series with Aperiodic Processing and Synchronization

The objective of the SA Series is to provide test requirements with a set of tasks that combine all the characteristics of the previous series of tests. This series is the most complex and demanding for a system to handle. It will contain periodic and aperiodic tasks as well as synchronization between tasks. This series of tests is complex enough that it can be modified to prototype many real systems. It will provide a good systems test of many of the features required of real-time systems.

4. Detailed Requirements: PH Series

The objective of the PH Series is to provide simple test requirements with a load of tasks that are purely periodic and harmonic. This series might represent a program that monitors several banks of sensors at different rates and displays the results with no user intervention or interrupt requirements. It is particularly amenable to a cyclic executive design and will have the highest theoretical utilization (100%) for a rate monotonic scheduler.

The baseline system consists of five periodic tasks. Each task frequency is a multiple of every higher task frequency. The tasks each execute a number of Whetstone units specified in the table below. The total Whetstone load for each task as well as for the entire task set is expressed in Kilo Whetstone Instructions Per Second (KWIPS). The baseline test of the series has a workload of 80 KWIPS allocated as follows:

Task No.	Frequency	Whetstones	Total Whetstones
1	1 Hz	16 KWI/cycle	16 KWIPS
2	2 Hz	8 KWI/cycle	16 KWIPS
3	4 Hz	4 KWI/cycle	16 KWIPS
4	8 Hz	2 KWI/cycle	16 KWIPS
5	16 Hz	1 KWI/cycle	16 KWIPS
Total			80 KWIPS

All tasks should be scheduled to start at the same time. (This is the worst case phasing.) It is up to the scheduler to determine which of the tasks actually runs first. The periodic tasks are defined such that the deadline for each period is defined as the start of the next period. If task i is first requested to start (dispatched) at time t_0, then its first deadline (and second request) is at time t_0+t_i, where t_i is the period (reciprocal of the frequency) for task i. Subsequent deadlines (and requests) are at t_0+2*t_i, t_0+3*t_i, and so forth. These times can be computed in any units which are of sufficient accuracy.

Successful tests should have three trials each of, say, 30 seconds. If a deadline is missed, the fact should be recorded, and the next cycle for the task may be skipped. The number of activations of each task should have the same relative accuracy as the type duration and be verified by the software. The Whetstone calculations should be the self-verifying version specified by Wichmann [Wichmann 88].

The test results for the baseline configuration might appear as follows:

```
Raw compute speed in Kilo Whetstones Per Second:  1086.98

Test duration (seconds):  30.0

Task   Period in    No. Times    Missed     Cumulative
 No.      Secs      Activated   Deadlines      Late
  1      1.0000        30           0         0.0000
  2      0.5000        60           0         0.0000
  3      0.2500       120           0         0.0000
  4      0.1250       240           0         0.0000
  5      0.0625       480           0         0.0000

 Total workload achieved (KWIPS):                    80.00
 Total workload achieved as percent of requested:  100.00
 Total workload achieved as percent of raw speed:    7.36
```

The raw compute speed is based on the time required to execute 1000 KWIPS without any task interactions. In the table, the period is derived as the reciprocal of the frequency, the activations and the number of missed deadlines of each task are counted by the test program, and the cumulative amount that the deadlines for each task are missed is accumulated. The total workload achieved is computed from the activations and the workload per activation. The achieved workload is then compared to the requested workload and the maximum achievable workload to give the utilization. It should be noted that the word "utilization" is defined differently than in scheduling theory. Here it refers to "useful work," whereas in scheduling theory the utilization includes the system overheads and gives an indication of the idle time associated with a scheduling algorithm.

One way to stress-test this baseline configuration is to increase the frequency of Task 5 by 8 Hz for each new test. This will allow a gradual increase in frequency of a single task while keeping the task set harmonic. This progression rapidly increases context switching rather than adding significantly to the computing load. This would mean, for example, that the tenth test in the series would have Task 5 running at 88 Hz with the entire task set executing 152 KWI. After increasing the frequency of the highest frequency task a number of times, the task set becomes:

```
Task No.    Frequency     Whetstones   Total Whetstones
--------    ---------     ----------   ----------------
   1           1 Hz      16 KWI/cycle       16 KWIPS
   2           2 Hz       8 KWI/cycle       16 KWIPS
   3           4 Hz       4 KWI/cycle       16 KWIPS
   4           8 Hz       2 KWI/cycle       16 KWIPS
   5          56 Hz       1 KWI/cycle       56 KWIPS
 Total                                     120 KWIPS
```

The test result may then show that deadlines are missed at this level:

```
Raw compute speed in Kilo Whetstones Per Second:  1086.98

Test duration (seconds):  30.0

Task    Period in     No. Times     Missed      Cumulative
 No.       Secs       Activated    Deadlines       Late
  1      1.0000          30            0          0.0000
  2      0.5000          60            0          0.0000
  3      0.2500         120            0          0.0000
  4      0.1250         240            0          0.0000
  5      0.0192        1578          102          0.0353

 Total workload achieved (KWIPS):                    116.60
 Total workload achieved as percent of requested:     97.17
 Total workload achieved as percent of raw speed:     10.73
```

The baseline task set for the PH series can actually be the basis for a number of experiments testing the sensitivity of the runtime system to a number of different changes. The sequence of tests can then be the basis for a single metric that is indicative of how the system handles periodic harmonic task sets. We define four experiments as follows:

Experiment 1: As shown above, the frequency of the highest frequency task is increased by 8Hz until a deadline is missed. This sequence increases the total workload by 8 KWIPS at a time and tests the system's ability to handle a fine granularity of time and to rapidly switch between processes.

Experiment 2: Starting with the baseline task set, all the frequencies are scaled by 1.1, then 1.2, then 1.3, and so on until a deadline is missed. This sequence increases the total workload by 8 KWIPS at a time and tests the system's ability to handle an increasing but balanced workload.

Experiment 3: Starting with the baseline task set, the workload of each task is increased by 1, then 2, then 3 Kilo-Whetstones per period, and so on until a deadline is missed. This sequence increases the total workload by 5 KWIPS at a time without significantly increasing the system overhead.

Experiment 4: Starting with the baseline task set, new 4 Hz tasks, each with a workload of 2 Kilo-Whetstones per period, are added until a deadline is missed. This sequence increases the total workload by 8 KWIPS at a time and tests the system's ability to handle a large number of tasks.

Each of the above experiments could lead to utilization as a metric, but a more balanced metric would be the average utilization derived from the four experiments. We define the Hartstone metric for test series PH as:

$$U_{PH} = 1/4 * (U_1 + U_2 + U_3 + U_4)$$

where U_i is the utilization of each individual experiment. That is, U_i is the actual achieved workload for experiment *i* divided by observed maximum workload for a single thread.

5. Four Ada Designs

In this section we describe four distinct Ada software architectures for implementing the requirements of the PH Series of tests. These four models are generated by the method of achieving a periodic process (either using the Ada delay statement or using interrupts generated by a hardware timer) and by the scheduling method (either a priority-based, preemptive, non-deterministic (ND) discipline using Ada's runtime system for scheduling or a non-preemptive, deterministic cyclic executive (CE) that is handcrafted based on the periods of the individual tasks). We describe these as the Delay/ND Model, the Interrupt/ND Model, the Delay/CE Model, and the Interrupt/CE Model. These are, by no means, the only Ada models, but are a good starting point for experimentation. We describe each of these briefly, but in enough detail that it should be possible to carry out an implementation with a modicum of Ada design experience. References to more detailed information are given.

5.1. Delay/ND Model

In the delay model each of the periodic tasks is implemented as an Ada task. To ensure that all tasks start at the same time, the main program performs a rendezvous with all the periodic tasks and passes a parameter giving the same starting time and test duration to each of the tasks. Each task is identically structured as a loop. In the loop the task delays until the starting time for its next period. Then it executes its workload. Finally it reads the clock to determine if it has met its deadline. If it meets the deadline, it computes the delay for the next starting time. If it fails to meet its deadline, it computes the amount by which the deadline was missed and sheds load skipping to the next period that it has not missed. For optimal scheduling, the tasks are assigned priorities according to the rate monotonic discipline, that is, the highest frequency tasks have the highest priorities. The advantages of delay models are that they are much more portable than interrupt models and are, in general, easier to implement. Only delay statements and clock reads are used to implement the periodicity. One disadvantage of this model can be that it is entirely dependent on the resolution of the clock and the resolution of the delay statement. The minimum delay possible, as well as the error of the actual delay compared to the requested delay, will strongly influence the results. Another disadvantage of this model is that it has a subtle failure mode. Because there is no "delay-until-absolute-time" function in Ada, this function must be approximated by a clock read and an interval delay. If the task is interrupted (by the delay expiration of another task, for example) between the clock read and the initiation of the delay, then the delay may be too long by the amount of time consumed by the interrupt. This may cause premature termination of the experiment.

5.2. Interrupt/ND Model

In the interrupt model each of the periodic tasks is also implemented as an Ada task. The difference is that instead of using the delay statement for periodicity, a stream of interrupts is generated from a hardware clock at the frequency of the highest frequency task. The interrupt service routine acts as a dispatcher for the periodic tasks. At each interrupt, the highest fre-

quency task is dispatched using a rendezvous. A table is used to keep track of when the lower frequency tasks are dispatched. In the worst case, all the tasks would require dispatching on the same clock tick. One implementation-dependent version of the dispatching model is described in two SEI technical reports [Borger 87a, Borger 87b]. This solution does not use delay statements, but *does* use a rendezvous for each execution period of each task. It is determined that deadlines are missed when the conditional rendezvous fails. Priorities can be assigned in the same manner as the delay model for optimal rate monotonic scheduling. The advantage of this solution is that the interrupt stream is very regular. The disadvantage is that this solution is non-portable because the characteristics of the hardware clock and the manner in which interrupts are handled are not uniform from one implementation to the next. To adapt this version to a new compilation system, it is necessary to have a detailed understanding of the target timer device and possibly interrupt controllers, as well as how the language implementation handles interrupts, tasks, interrupt vectors, and access to low-level devices. This model is a particularly good test of the maturity and efficiency of runtime implementations.

5.3. Delay/CE and Interrupt/CE Models

The use of cyclic executive models in Ada are described in [Baker 88]. This paper provides a good exposition of the cyclic executive model, evaluates Ada as a programming language for constructing real-time software within this model, and presents a delay-based solution and a number of interrupt-based solutions to the more serious Ada problems. The problems for the two basic models are quite similar to the problems described above for the ND models. The difficulty and inflexibility of a handcrafted deterministic schedule can be easily illustrated by using a task set consisting of the first three tasks of the example given above. A cyclic executive would have a major cycle of one second (corresponding to the time period in which all activities must be completed) and a frame size of 0.25 seconds (corresponding to the frequency of the 4 Hz task). Then the developer of the schedule is faced with the same choices as the designer of real systems. Because it is often difficult to determine the computation time (including overhead) of each task, the designer is faced with the "cut and try" approach of fitting the tasks into frames, monitoring performance, and then modifying the schedule. Given the scenario of the first three tasks, the first possible schedules to try might be:

	Frame 1	Frame 2	Frame 3	Frame 4
Schedule 1:	**T3 T2 T1**	**T3**	**T3**	**T3**
Schedule 2:	**T3 T2**	**T3 T1**	**T3**	**T3**
Schedule 3:	**T3 T2**	**T3 T1A**	**T3 T1B**	**T3**
Schedule 4:	**T3 T2A**	**T3 T2B T1A**	**T3 T1B**	**T3 T1C**

Here the division of Task T1 into two components is represented by T1A and T1B. It can be easily seen that even for this simple example, there are numerous combinations of schedules. For the five-task baseline there are even more possibilities, so that conducting the experiments defined above may be a somewhat arduous task. It is for these reasons that the SEI has focused its scheduling efforts on priority-based preemptive scheduling [Sha 88].

6. Preliminary Results and Conclusions

Both the Delay/ND and the Interrupt/ND models have been coded in Ada; a limited amount of experimentation has been carried out; and results have been obtained for several Ada compilers. The utilization levels at which deadlines start to be missed in Experiment 1 have been rather low compared to expectations, on the order of 10% to 30% in most cases. Several termination modes have been observed. The first is represented by the example given above. Deadlines begin to be missed by the highest frequency task because the frequencies have been driven to the level that the resolution of time and the resolution of the delay statement become a problem. Another termination mode, caused by increasing workloads is predictable by theory and causes the lowest frequency task to miss its deadlines. A more troubling termination mode for real-time embedded systems is when the deadlines are missed for tasks that do not have the lowest frequency (and consequently the lowest priority). For host-based systems that run with an operating system such as UNIX, it can be expected that unpredictable events or daemons might cause deadlines to be missed. In bare machine Ada implementations, intervention by the Ada runtime system must be short and predictable if deadlines are to be met, so that when higher frequency tasks miss their deadlines before or along with lower frequency tasks, it is a warning sign that the Ada runtime system may not be suitable. Finally, one compiler did not even support preemptive scheduling at delay termination and could not meet any deadlines.

Over all, the Ada implementations of the Hartstone requirements tended to be very stressful of Ada runtime systems, and appear to be rigorous tests of tasking behavior so important to real-time performance. Several compiler bugs have been uncovered. The Hartstone programs exercise the important characteristics of runtime behavior, including task switching, interrupt handling, rendezvous, scheduling behavior, and determinism of the runtime system. Our experience has been that the Hartstone programs bring to light unexpected behavior or performance problems that should be investigated with fine-grained benchmark programs. To correctly interpret the results, the user may need detailed knowledge of and execution times for clock operations, task rendezvous and context switching, timer interrupt latency, interactions with the floating point coprocessor (if any), and the procedure calling overhead. In addition to providing insight into the real-time performance characteristics of an Ada runtime system, the Hartstone programs also provide canonical examples of real-time processing that can be useful for modeling and experimentation.

References

[Altman 87] N. Altman.
Factors Causing Unexpected Variation in Ada Benchmarks.
Technical Report CMU/SEI-87-TR-22, DTIC: ADA187231, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, October, 1987.

[Baker 88] Baker, T.P. and Shaw, A.
The Cyclic Executive Model and Ada.
In *Proceeding of Real-Time Systems Symposium, Huntsville, AL*, pages 120-129. IEEE, December, 1988.

[Borger 87a] Borger, M.W.
VAXELN Experimentation: Programming a Real-Time Clock and Interrupt Handling Using VAXELN Ada 1.1.
Technical Report CMU/SEI-87-TR-29, DTIC: ADA188100, Software Engineering Institute, October, 1987.

[Borger 87b] Borger, M.W.
VAXELN Experimentation: Programming a Real-Time Periodic Task Dispatcher Using VAXELN Ada 1.1.
Technical Report CMU/SEI-87-TR-32, DTIC: ADA200612, Software Engineering Institute, October, 1987.

[Cornhill 87] Cornhill, D. and Sha, L.
Priority Inversion in Ada.
Ada Letters 7(7):30-32, November, 1987.

[Curnow 76] Curnow, H.J. and Wichmann, B.A.
A Synthetic Benchmark.
Computer Journal 19(1):43-49, January, 1976.

[IEEE 88] .
The Fifth Workshop on Real-Time Software and Operating Systems.
In . IEEE Computer Society and USENIX Association, Washington, D.C., May 12-13, 1988.

[Kar 89] Kar, R.P. and Porter, K.
Rhealstone -- A Real-Time Benchmarking Proposal.
Dr. Dobbs Journal 14(2):14-24, February, 1989.

[Lehoczky 87] Lehoczky, J.P., Sha, L., and Strosnider, J.
Enhancing Aperiodic Responsiveness in Hard Real-Time Environment.
In *Proceeding of Real-Time Systems Symposium*, pages 261-270. IEEE, December, 1987.

[Liu 73] Liu, C.L. and Layland, J.W.
Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment.
Journal of the Association of Computing Machinery 20(1):46-61, January, 1973.

[Sha 88] Sha, L. and Goodenough, J.B.
Real-Time Scheduling Theory and Ada.
Technical Report CMU/SEI-89-TR-14, Software Engineering Institute, November, 1988.

[Weicker 84] Weicker, R.P.
Dhrystone: A Synthetic Systems Programming Benchmark.
Communications of the ACM 27(10):1013-1030, October, 1984.

[Weicker 88] Weicker, R.P.
Dhrystone Benchmark: Rationale for Version 2 and Measurement Rules.
SigPLAN Notices 23(8):49-62, August, 1988.

[Wichmann 88] Wichmann, B.A.
Validation Code for the Whetstone Benchmark.
Technical Report DTIC 107/88, National Physical Laboratory, Teddington, Middlesex, UK, March, 1988.

Table of Contents