

Effects of Compiler and Runtime System Features on Embedded System Designs

William R. Bush

This study identified Ada features important to real-time performance, and investigated various real-time benchmarking methodologies.

The overall effort consisted of three steps:

- (1) we reviewed a number of efforts concerned with Ada features and performance;
- (2) we examined studies that quantified Ada performance requirements; and
- (3) we investigated a number of real-time benchmarking efforts.

Ada is a complex language, intended for the programming of large, complicated systems. It is also, somewhat paradoxically, designed for time-critical software. Thus interrupt handling, context switching, and other substantial language features must execute quickly and predictably.

Both computing hardware and Ada compilers vary substantially with respect to performance. Unavoidably, there are embedded programs with constraints that cannot be met by some platforms.

Thus a method is needed for deciding which platforms are suitable for running a particular embedded application. Furthermore, understanding which embedded system constraints are difficult to satisfy may lead to new insights into embedded system implementation.

1. Technical Objectives

Real-time systems must reliably meet a variety of real-time constraints. Although Ada is intended to be used for such real-time applications, the Ada Language Reference Manual [LRM] deals neither with absolute or relative performance. The Ada Compiler Validation Capability Suite was established to validate the form and meaning of programs written in Ada, not to be used for measurement. Thus the Ada language definition deals with half of the real-time problem; it contains mechanisms to accommodate real-time applications, but leaves performance undefined and unpredictable.

1.1. Objective 1: Identify Critical Areas of Real-Time Performance

List features of Ada that are expected to affect real-time performance. The expectations are based on experience implementing real-time systems in Ada.

1.2. Objective 2: Quantify Requirements of Real-Time Embedded Systems

Collect the performance requirements of real-time embedded systems. These requirements quantify the needed performance of the language features identified in the first objective.

1.3. Objective 3: Survey Attempts at Measuring Real-Time Performance

Survey previous and ongoing attempts at measuring (benchmarking) real-time performance. Many, but not all, of these attempts involve Ada.

2. Technical Background

The Ada standard was developed by language implementors, not real-time system developers. It thus leaves many implementation issues to be determined by implementors, and avoids making commitments on implementation issues. The result is widely varying performance between implementations, and an uncomfortable amount of unpredictability.

The resulting problems are substantial enough to have stimulated the Ada 9X effort (see [PRW]), an attempt to minimally redefine the language, in order to solve these problems (and others that have been identified as experience with the language has grown).

In addition, two user groups, the ACM Ada Special Interest Group's Ada Runtime Environment Working Group (ARTEWG) and Performance Issues Working Group (PIWG), have been monitoring the problems that real-time users have had using Ada, and have made many important contributions (such as [Survey] and [Performance]).

3. Technical Results

We present here the technical results of our study. We discuss the results from each of our technical objectives, and then summarize our conclusions.

3.1. Critical Areas of Real-Time Performance

We now present a number of critical performance areas. Keep in mind that there are three aspects to real-time performance:

- (1) direct feature performance (such as speed or memory size),
- (2) control (accurate real-time clocks, priority scheduling, etc.), which affects overall system performance, and
- (3) predictability, which has a major effect on reliable execution.

The area surveys can be divided into three types, those that enumerate features, those that identify problem areas, and those that indirectly identify areas by specifying how Ada should be used.

3.1.1. The Michigan Benchmarks

Researchers at the University of Michigan [Michigan] developed a set of small benchmarks, which were primarily designed to measure the performance of Ada implementations. The benchmark list enumerates the features thought important to real-time performance. The individual benchmarks are:

- (1) *Subprogram overhead.* Relevant measurements are of simple call and return, simple parameter passing, composite parameter passing, and passing parameters with constraints. In addition, implementation tests include intrapackage call, interpackage call, generic call, inline pragma tests, and tests to determine if parameter passing is done with call by value or call by reference.
- (2) *Dynamic object allocation.* Three types of allocation are tested: when the amount of storage is known at compile time and when it is not, and when the *new* allocator is used.
- (3) *Memory deallocation and garbage collection.* Relevant tests include experiments to determine paging and garbage collection behavior. Implementation tests perform implicit and explicit freeing (unchecked deallocation), to determine if unchecked deallocation is

implemented.

- (4) *Task elaboration, activation, and termination.* These are measured in various ways (with and without the *new* allocator).
- (5) *Task synchronization.* Rendezvous between tasks and procedures are performed, with and without various parameters.
- (6) *Exceptions.* Four types are tested, numeric errors (discovered by hardware), constraint errors (raised by the Ada runtime system), tasking errors, and user-defined exceptions. These types are tested under two conditions: when the exception is handled in the unit in which it was raised, and when the exception is propagated between units.
- (7) *CLOCK function overhead.* The CLOCK function is in the CALENDAR package, and is used for timing.
- (8) *Arithmetic for types TIME and DURATION.* These standard types are used for timing and scheduling, and arithmetic on them is often used in *delay* statements; TIME specifies absolute times, while DURATION specifies intervals.
- (9) *Delay and scheduling.* Minimum delay overhead is measured. In addition, the benchmarks determine whether an implementation uses fixed-interval versus preemptive scheduling.
- (10) *Interrupt response time.* Special hardware is used to measure response time.

Note that some of these features are common to almost all programming languages, such as subprogram overhead. They are important to, but not characteristic of, real-time systems. Other features, such as rendezvous and delay, are more Ada and real-time specific.

3.1.2. The Ada Performance Issues Working Group Benchmarks

Another benchmark set was developed by the Ada Performance Issues Working Group [PIWG]. This benchmark set consists of feature benchmarks similar to the Michigan set, but also includes larger application benchmarks.

The language features measured are:

- (1) task elaboration, activation, and termination, with one task with no entries, in various contexts;
- (2) elaboration of an array of 1000 integers, in various contexts;
- (3) time to raise and handle an exception, in various contexts (local, nested three and four deep in procedure calls, and propagated through a rendezvous);
- (4) boolean flag setting;
- (5) TEXT_IO (not necessarily relevant to embedded applications);
- (6) boolean operations on arrays, packed in various ways (to test space management and memory optimization);
- (7) UNCHECKED_CONVERSION tests;
- (8) bit field operations on record components;
- (9) loop overhead (simple for, while, and exit), and compiler loop unrolling decisions;
- (10) procedure call and return, in various contexts (inline or not, and with parameters of assorted modes);

- (11) rendezvous entry call and return time, in various contexts (one task and one entry, two tasks and one entry, ten tasks and one entry, one task and ten entries, producer-buffer-consumer tasks); and
- (12) commanded delay versus actual delay.

Several of these tests are designed to investigate compiler optimization effects.

3.1.3. The Naval Avionics Center Benchmarks

The Naval Avionics Center was concerned with the speed of a number of Ada language features in the context of their existing embedded systems [NAC]. Features were divided into three categories, based on how important they were: essential, important, and desirable.

Essential features were:

- (1) subprogram call speed,
- (2) delay statement accuracy,
- (3) rendezvous speed,
- (4) task context switch speed,
- (5) selective wait,
- (6) conditional timed call,
- (7) rendezvous parameter passing speed,
- (8) record component reference speed, and
- (9) the penalty for using packages.

Important features were:

- (1) parameter passing,
- (2) interrupt latency,
- (3) exception handler overhead, and
- (4) the ability to have no implicit dynamic storage.

Desirable features were:

- (1) task declaration/allocation, and
- (2) task termination.

3.1.4. Next Generation Computer Resources

The Operating System Interface Standard Requirements document [NGCR] has an entire section (2.16) devoted to Ada language support interfaces. These interfaces include:

- (1) creating a task,
- (2) aborting a task,
- (3) suspending a task,
- (4) resuming a task,
- (5) terminating a task,
- (6) restarting a task,

- (7) task entry calls (simple, timed, conditional),
- (8) task entry call accepting (simple, simple selective waits, selective waits with delay, selective waits with else, selective waits with terminate),
- (9) accessing task attributes (*),
- (10) monitoring task execution status (*),
- (11) accessing a precise, continuous real-time clock (*),
- (12) accessing a time-of-day clock,
- (13) dynamic task priorities (*),
- (14) scheduling policy selection (*),
- (15) memory allocation and deallocation,
- (16) interrupt binding (of an interrupt to Ada code, at least using task entry) (*),
- (17) enabling and disabling interrupts (*),
- (18) masking and unmasking interrupts (*),
- (19) raising a task exception, and
- (20) LRM Chapter 14 input/output support.

Some of these interfaces can be classified as providing OS support for basic Ada primitives. Others, identified by a star (*), are not required by Ada, but affect real-time performance. Many of these starred items are important because they provide control over system behavior.

In general, this list is suggestive. It highlights areas of potential system overhead. Items on it are not necessarily critical to real-time performance -- they may not occur frequently enough.

3.1.5. The Ada 9X Effort

The goal of the Ada 9X effort "is to revise ANSI/MIL-STD-1815A to reflect current essential requirements with minimum negative impact and maximum positive impact to the Ada community. The Ada 9X process is a revision and not a redesign of the language and should be viewed as a natural part of the language maturation process." (*Ada Letters*, Volume IX, Number 4, p. 9.) One of several Ada 9X concerns is real-time performance. A number of real-time groups have developed Ada Revision Requests (ARRs) dealing with real-time issues.

3.1.5.1. The Ada Run Time Environment Working Group ARR

In 1989 ARTEWG finalized a number of ARR [ARTEWG]. (See also other proposals in *Ada Letters*, Volume IX, Number 5, pp. 31-35.) The ones most relevant to real-time performance were:

- (1) Make the execution of interrupt handlers predictable.
- (2) Allow multiple objects of the same task type to be associated with different interrupting devices of the same kind.
- (3) The programmer should be able to declare alternatives to raising `STORAGE_ERROR`.
- (4) An application should be able to reserve a dynamic amount of storage for recovering from `STORAGE_ERROR`.

- (5) Garbage collection should be required and under user control.
- (6) The LRM should guarantee that memory should never be "lost".
- (7) Preemptive scheduling should be preserved.
- (8) The language should not preclude the explicit modification of task priorities.
- (9) The rate of CLOCK should be changeable.
- (10) The value of CLOCK should be adjustable.
- (11) Absolute as well as relative delays should be supported.
- (12) Asynchronous event handling should be supported.

3.1.5.2. The Ada 9X Project Requirements Workshop

As part of the 9X process, the Ada 9X Project Requirements Workshop was held in 1989 to give various parties (in DoD, industry, and academia) a chance to present revision requirements [PRW]. There were five working groups, one of which was Real-Time Embedded Systems. Its recommendations were:

- (1) Allow objects of a task type to wait on different interrupts.
- (2) Provide a mechanism for backloging interrupts.
- (3) Provide a reliable capability for suppressing checks (DIVISION_CHECK, OVERFLOW_CHECK, ELABORATION_CHECK).
- (4) Provide a means for determining the cause of a STORAGE_ERROR (such as allocator failure or stack overflow).
- (5) Support efficient bit manipulation (shift, rotate, test, find first one).
- (6) Allow fixed point model numbers to be specified exactly, for all rational numbers.
- (7) Support the sharing of memory with an external agent by providing a mechanism to express the need for every read and write to occur (as opposed to optimizing by keeping values in registers).
- (8) Provide an integer type that does not cause NUMERIC_ERROR or CONSTRAINT_ERROR, but instead wraps around.
- (9) Support explicit call by value and call by reference (for predictability).

3.1.5.3. The Third International Workshop on Real-Time Ada Issues

The 1989 Workshop on Real-Time Ada Issues, sponsored by SEI and ONR, specifically considered solutions to perceived problems with Ada, and produced a number of recommended language changes [RTAI].

- (1) Support asynchronous transfer of control (through a modification to the select construct)
- (2) Allow task parameters (to simplify and speed up initialization).
- (3) Support asynchronous task communication by providing messages and queues.
- (4) A secondary Ada standard should be developed, which specifies a set of high level tasking optimizations (for uniformity). Three examples are (pp. 41-46): usage restrictions leading to efficiency, fast interrupt entries, and monitors.

- (5) Allow complete storage recovery on task termination.
- (6) Allow user-defined task priorities to be changed at runtime.
- (7) A secondary Ada standard should be developed, which provides access to low level tasking primitives (such as defined in the ARTEWG Catalog of Interface Features and Options). At a minimum, the interface should provide (p. 102): the ability of a task to determine its identity, and others with which it rendezvous; the ability to invoke the scheduler; the ability to suspend and resume a task; the ability to determine if a task is suspended; the ability to enable and disable machine interrupts; and the ability to enable and disable preemption.
- (8) Provide bounds on the delay statement, so that execution will resume within a bounded length of time.
- (9) Provide two clocks, a time of day clock and a continuous fixed rate clock.
- (10) Provide a timeout on the execution of a sequence of statements (through a modified select statement).
- (11) Require validated compiler vendors to supply performance data on, for example, (pp. 127-128): the delay statement, tasking, interrupts, and clocks.
- (12) The semantics of timed entry calls and delay alternatives in selective waits should be the same as ordinary delay statements.
- (13) Provide absolute as well as relative delays (with a delay until statement).
- (14) Allow the enforcement of a CPU time budget on tasks (through a modified select statement).
- (15) Support substantial pre-elaboration, so that programs may begin execution faster.
- (16) Allow specifying the location and placement of various entities in memory.
- (17) Supply a standard, optional package of exclusion primitives.
- (18) Require heap storage to be allocated in a predictable manner such that the heap space requirements of a program can be reliably predicted.

In addition, general proposals were made regarding fault tolerance and Ada for distributed systems.

3.1.6. NUSC Systems: CCS MK 2

Information on the CCS MK 2 system comes from the Software Standards and Procedures Manual Style Guide [CCS]. It is a normative document, showing how Ada should be used (see pages I-33-37, I-48-49, and I-24). The reasons for particular style guidelines vary; some are given for software engineering reasons, while others are based on characteristics, and current limitations, of Ada.

Abstracting from the guidelines, we can make some observations. The sentences in parentheses are summaries of the text of the guidelines. The plain sentences are the implications that can be drawn from the summaries.

- (1) Nested tasks are *not* common. (Tasks should not be nested.)
- (2) Access types of task types are *not* common. (Access types of task types should be avoided.)
- (3) Conditional entry calls are *not* common. (Conditional entry calls should be avoided (because of busy waiting).)

- (4) Inaccuracy with delay times is a problem. (The delay time in a delay statement is only a lower bound on the delay, and cannot be relied on as precise.)
- (5) Only a small number of priority levels need be supported. (Only a small number of priority levels should be used.)
- (6) Aborting tasks will not occur frequently. (Aborting tasks should be avoided.)
- (7) Most tasks will have an associated exception handler. (A task should generally have a block statement with an exception handler coded within its main loop, to handle local exceptions without termination.)
- (8) Tasks that call other tasks will have `TASKING_ERROR` alternatives in their exception handlers. (Tasks that make entry calls on other tasks should include a `TASKING_ERROR` alternative in their exception handlers.)
- (9) Interrupt handlers and interfaces (to hardware, or foreign code or data) will be implemented to some extent outside of Ada. (Representation clauses and implementation dependent features should only be used for: interrupt handlers, interfaces (to hardware, or foreign code or data), or efficiency (when absolutely necessary).)
- (10) All subprograms in which an exception can be raised will have an "others" alternative. (An "others" alternative is a minimum requirement for all subprograms in which any exception can be raised.) However, a following guideline expressed concern for the efficiency of this guideline.

In general, items on this list are not necessarily important to real-time performance. They do, however, suggest possible critical items, as well as simplifications.

3.1.7. NUSC Systems: AN/BSY-2

We had hoped that the BSY-2 assessment team study [BSY-2], which reviewed the development of the system, would provide insight into critical Ada issues. Unfortunately, the review determined that the software developer was unfamiliar with Ada, and this inexperience led to three project risk items. These risk items were: insufficient guidance (in the "Standards and Procedures Manual") on the use of tasking and rendezvous, insufficient guidance on memory management, and insufficient prototyping to determine the performance of key Ada features (p. 42, p. A-11). The benchmarking information produced by the third item would have been useful to this survey.

3.1.8. NUSC Systems: ALSN/Phalanx

We had hoped to gain some insights from this system, but we were unable to obtain information on it.

3.2. Quantified Real-Time Requirements

Quantified real-time requirements are hard to find. This seems to be due either to the failure to articulate them during project definition (see [BSY-2]), or to the failure to make them available outside the project. Two notable exceptions are the ARTEWG Survey and the NAC Requirements.

3.2.1. The Ada Runtime Environment Working Group

The Ada Runtime Environment Working Group undertook a user survey to collect and categorize the Ada runtime requirements of mission critical software from an applications point of view [Survey]. The survey questions were varied, covering such areas as the knowledge of the

respondent, the type and cost of the application, and reliability requirements. Responses for 35 projects were received.

A number of questions are directly relevant to this study. Response counts are supplied in parentheses. RTSE stands for Run Time Support Environment.

- (1) The application will receive inputs, other than messages: asynchronously (19), periodically (19), synchronously (17).
- (2) The inputs can include spurious signals (13).
- (3) The application will receive messages: asynchronously (17), periodically (14), synchronously (10).
- (4) The messages can contain illegal information (17).
- (5) The application will produce outputs, other than messages: asynchronously (13), periodically (13), synchronously (16).
- (6) The application will output messages: asynchronously (17), periodically (13), synchronously (11).
- (7) The application is: numeric intensive (14), moderate (8), not numerically intensive (11).
- (8) The application handles the following types: integer (27), fixed point (16), floating point (23), complex (4), character data (24).
- (9) How much memory will be available for the application (bytes)? (answers included 8K, 64K, 128K, 256K, 512K, 1M, 4M, 8M, 10M, 32M, and enough)
- (10) The hardware generates fault interrupts for some Ada exceptions (7).
- (11) What percentage of the code must execute without interruption? (1-10%, 10; 10-20%, 6; 20-30% 3; > 30%, 7)
- (12) The application can afford a context switch for critical sections (7).
- (13) The application will disable runtime checks (5 yes, 17 only if necessary to meet performance goals).
- (14) How will procedure parameters be passed? (global structures, 7; parameters only, 6; mixed parameters and global structures, 17; mailboxes, 4)
- (15) The application will have tasks (29).
- (16) Tasks will be synchronized via rendezvous (14)?
- (17) How will tasks communicate? (rendezvous, 10; global structures, 15; mailboxes, 6)
- (18) How many priority levels are needed? (answers included 0 (5 responses), 2, 3, 4, 5, 6, 10, 15, and 256)
- (19) The number of tasks will be determined at runtime (12).
- (20) What is the maximum number of active tasks? (answers included 2, 4, 5, 6, 10, 15, 20, 25, 30, 30+, and 200+)
- (21) What is the maximum level of task nesting? (answers ranged from 0 to 5)
- (22) Tasks will be created dynamically (3).
- (23) What tasking model do you need? (cyclic exec, 9; foreground-background, 1; communicating sequential processes, 2; monitors, 1)

- (24) The number of some data objects will be determined at runtime (11).
- (25) The size of some data objects will be determined at runtime (7).
- (26) The RTSE must provide: task timeout (12), I/O timeout (18), rendezvous timeout (8), preemptive scheduling (17), guarantee that a task begins execution before a specific time (16), guarantee that some tasks execute with a given frequency (23), detect and report that no task is scheduled (7), control access to shared data (18), dynamically assign task priorities (12), alert the application that memory is nearly exhausted (4), provide memory status on request (7), perform garbage collection (11).
- (27) When should the RTSE perform garbage collection? (whenever necessary, 6; periodically, 1; only when permitted, 11; never, 5)
- (28) The RTSE must provide analytically predictable execution times (21).
- (29) The application will service interrupts (28).
- (30) The application will control devices directly (21).
- (31) What interrupt latency is acceptable? (answers include 0 (9 responses), .01 msec, .1 msec, 1 msec, and 5.2 msec)
- (32) The application will make use of Ada exceptions (11).
- (33) Recursion: will need (3), will use if needed (13), will forbid (13).
- (34) Constraint checks: will use during debugging (7), will use during operation (5), will use during operation as performance permits (19).
- (35) Unchecked data type conversion: will need (11), will use if needed (15), will forbid (3).
- (36) Unchecked deallocation: will need (3), will use if needed (17), will forbid (6).
- (37) Access types: will need (8), will use if needed (14), will forbid (3).
- (38) Inline assembly language: will need (11), will use if needed (10), will forbid (5).

Interesting facts: only 6 projects will use an existing RTSE; 23 respondents were very concerned about performance, 7 were somewhat concerned; 12 respondents were very concerned about memory usage, 14 were somewhat concerned.

3.2.2. Naval Avionics Center Requirements

The Naval Avionics Center requirements document includes a table for the required performance of various Ada features (p. 39). It includes:

- (1) subprogram entry/exit, 5 usec;
- (2) parameter passing, 2 usec + 1 usec/parameter;
- (3) task declaration/allocation, 1000 usec;
- (4) normal task termination, 500 usec;
- (5) delay, 200 usec;
- (6) simple rendezvous, 200 usec;
- (7) task context switch, 70 usec;
- (8) interrupt latency, 200 usec;

- (9) selective wait, 20 usec;
- (10) conditional or timed call, 20 usec;
- (11) parameter passing during rendezvous, 20 usec + 10 usec/parameter;

3.2.3. Next Generation Computer Resources

The Operating System Interface Standard Requirements document contains a "Metric" heading for each subsection, including all the required Ada language support interfaces. In Version 2.0 of the document there are no metrics supplied. When metrics become available they will provide requirements for all the features in the interface.

3.2.4. The AN/BSY-2 System

Useful quantified requirements could be developed from the AN/BSY-2 system, when the recommended prototyping benchmarks are completed.

3.3. Measuring Real-Time Performance

Measuring system performance accurately is always difficult, and real-time systems add additional problems. Several well thought out attempts have been made at measuring real-time performance, with varying goals and varying sets of benchmark programs. We first present various general categories of measurement, and then discuss individual measurement efforts. Much of the following general discussion is based on, or inspired by, material in [C&M].

We have identified three reasons for measurement. First, there is implementation improvement. Such measurements are mainly for the benefit of implementors, to help them improve their systems. Second, there is comparative performance evaluation. These measurements are primarily for the benefit of users (or, more cynically, marketing departments). Third, there is required performance. Such measurements are also for the benefit of users, but the ultimate value of the measurements is not a set of numbers, but a boolean result, pass or fail. The motivation for making measurements obviously affects the measurement approach taken.

There appear to be four types of benchmarks: feature, composite, synthetic application, and full application. Feature benchmarks test individual language features. Composite benchmarks are composed of feature benchmarks, but are aimed at measuring total performance. Synthetic applications also measure total performance, but are not constructed out of discrete feature benchmarks. Full application benchmarks are just that, complete applications.

Combining reasons and types, consider some existing Ada benchmarks. The Michigan set is a feature type, aimed at improvement and comparison. The PIWG set is a multiple feature/synthetic/full type, also aimed at improvement and comparison. The NAC set is a composite type, specifying requirements. The Hartstone set is synthetic, for comparison. The CFA non-Ada set is also synthetic and for comparison.

Real-time benchmarks measure three things: space, time, and control. Real-time space measurements are conventional, measuring stack, heap, and code. Real-time time measurements monitor two quantities, execution speed and latency. Execution speed is conventional. Latency is the time between events, and its important characteristic is predictability. Control, particularly with respect to time, is a special concern of real-time systems, and affects predictability. Control over scheduling and delays is particularly important.

Given these quantities, measuring them raises a number of issues.

The first is timing. Both accuracy (the range of error) and precision (significant digits of measurement) must be quantified. In addition, translation anomalies, such as optimizations and code placement effects, must be accounted for. The second is repeatability, which may involve external timing factors. The third is feature isolation. Individual features are affected by compiler optimizations and by distributed overhead, a compilation problem which involves generating code to support unused features (see [Optimizing]). The fourth is language isolation (important mainly for improvement-oriented feature benchmarks). Performance is not only affected by the language system (compiler, library, and runtime system), but also by the larger host context (operating system and hardware platform).

3.3.1. The Michigan Benchmarks

The Michigan benchmark effort [Michigan] developed techniques for isolating and measuring specific language features, while avoiding optimizations and operating system interference. It uses a dual loop approach, with one null loop and one loop containing the feature under test. The loops are while loops with isolated iteration variable code, in order to defeat loop optimization.

The individual benchmarks are enumerated above.

3.3.2. The PIWG Benchmarks

Like the Michigan set, the PIWG set uses various strategies for producing meaningful results on widely differing architectures [PIWG]. It also uses a dual loop approach.

The PIWG set contains feature tests, synthetic applications, and a full application. The feature benchmarks are enumerated above. The synthetic applications consist of Whetstone (mathematical processing), Dhrystone (system programming based on a statistical study of feature use), and the Hennessy set (Towers of Hanoi, 8 Queens, quicksort, etc.). The full application is a path tracker using a covariance matrix, originally coded in JOVIAL.

3.3.3. The NAC Benchmarks

The individual features tested, and their required performance, are listed above.

In addition, the benchmark specifies an overall cyclic executive based on existing operational flight programs (OFPs). The parameters of this synthetic OFP are: the length of a cycle, the number and types of tasks, the number and types of interrupts per cycle, the number and types of rendezvous per cycle, the number and types of subprogram calls per cycle, and the hardware resources (processors and buses) required. The particular OFP requirements set out in the NAC document include 5 interrupt handlers, 19 application functions, and 5 server tasks. The main loop is executed every 25 milliseconds. Alternate cycles are heavily and lightly loaded. A heavily loaded cycle will keep each processor busy roughly 75% of the time.

3.3.4. Hartstone Benchmarks

The Hartstone benchmark set is similar to the NAC effort, constructed around a hard real-time cyclic executive [Hartstone]. The Hartstone set differs in its synthetic load and in its executive's structure. The synthetic load is simply Dhrystone or Whetstone, rather than a selected set of features. The executive is somewhat more complicated, testing performance under five different regimes: 1) periodic tasks at harmonic frequencies, 2) periodic tasks at non-harmonic frequencies, 3) as 1) with aperiodic processing added, 4) as 1) with synchronization, and 5) a

combination of 3) and 4).

As with the NAC benchmark, a success/failure result is produced (deadlines are met or missed).

3.3.5. Military Computer Family Benchmarks

One non-Ada benchmark set comes from the military computer family evaluation [CFA]. It defines a set of benchmarks aimed at determining real-time performance, and consists of several small, kernel type programs. The benchmarks are:

- (1) an I/O kernel, handling four devices with distinct priorities;
- (2) an I/O kernel, handling four devices with FIFO processing using one queue;
- (3) an I/O device handler doing block transfers;
- (4) a large fast Fourier transform (using a large amount of memory);
- (5) character searches (finding a first occurrence in a large string);
- (6) bit tests, sets, and resets;
- (7) a Runge-Kutta integration (third order);
- (8) doubly linked list insertions;
- (9) an application of quicksort (using a large vector of fixed length strings);
- (10) ASCII to floating point conversions;
- (11) a boolean matrix transpose (using a tightly packed bit matrix); and
- (12) a virtual memory space exchange.

This is a varied collection of programs covering a range of features. The benchmarks were evaluated in terms of three metrics, program size in bytes, the number of bytes transferred between processor and memory, and the number of internal register transfers. The latter numbers were determined by constructing hardware simulators for the processors.

3.3.6. SPECmarks

The SPECmark set is another non-Ada benchmark set that is, in addition, not real-time, but it is illustrative in being a set of full applications, and is achieving wide acceptance on the part of both computer vendors and users.

The full applications in the set are: a GNU C compilation, an espresso PLA generation, a spice 2g6 run, a doduc run, an execution of the NASA Ames kernel (vectorizable floating point), a LISP interpreter session, an eqntott run, an execution of matrix300 (matrix calculations), an fpppp (quantum chemistry) run, and a run of tomcatv (vectorizable floating point).

3.4. Summary

We first review concerns over specific Ada features. We then consider, from survey data, what issues a measurement effort should address. Finally, we present a number of possible measurement projects.

3.4.1. Feature Review

There are a number of Ada features that appear repeatedly on the above lists, as important features to be measured, supported, and/or modified. Generalized, and ordered by the number of

times they appear on the lists, they are:

- (1) Delay accuracy ([PIWG], [NAC], [ARTEWG], [RTAI], [CCS], [NGCR]). This refers to the difference between requested delay and actual delay, and is both a performance concern and a predictability concern. Actual delays should be close to requested delays, and the variance, if any, should be predictable.
- (2) Allocation, deallocation, and garbage collection ([Michigan], [ARTEWG], [PRW], [RTAI], [NGCR]). Concerns here are with overhead, both in time and space, predictability (particularly with respect to garbage collection), and control (also important with garbage collection).
- (3) Task synchronization overhead ([Michigan], [PIWG], [NAC], [NGCR]). This specifically refers to the performance of the rendezvous mechanism, in its various forms (with and without parameters, etc.). There is an implicit issue here that was of explicit concern to NAC, context switch overhead.
- (4) Preemptive scheduling and dynamic task priorities ([Michigan], [ARTEWG], [RTAI], [NGCR]). These are both control issues, not performance issues. Neither feature is required by the LRM, but both are considered important by users.
- (5) Exception overhead ([Michigan], [PIWG], [NAC]). The performance of this feature is important when it's executed, when it's used but not executed, and when it's not used (it should not contribute much, if anything, to distributed overhead).
- (6) Subprogram overhead ([Michigan], [PIWG], [NAC]). This refers to the mechanism's performance in various forms (with and without parameters, etc.).
- (7) Task elaboration, activation, and termination overhead ([Michigan], [PIWG], [NGCR]). This is more important to applications that dynamically create and terminate tasks; NAC was less concerned about the performance of these features because their tasks are statically allocated and are all created during initialization.
- (8) Interrupt response time ([Michigan], [NAC], [ARTEWG]). This feature is hard to measure, because it is substantially dependent on hardware.
- (9) Better clocks ([ARTEWG], [RTAI], [NGCR]). These are 9X requested features, not in the current LRM. In particular, the requests are for CLOCK to be set and the rate of CLOCK to be changed, and for two explicit clocks, a time of day clock and a continuous fixed rate clock, with the delay statement able to use both.
- (10) Control over interrupts ([ARTEWG], [PRW], [NGCR]). These are 9X requests. There is a desire for predictability (using specific priorities and specific paradigms), and a mechanism for backlogging interrupts.

The above list of features is suggestive. Any measurement strategy should, at a minimum, address these features. Additional features important to individual user groups should also be considered.

In addition, the result of the Ada 9X effort will certainly affect the feature list (the last two items above, for example, will be irrelevant if not incorporated into the 9X standard). The 9X process must be tracked and the outcome folded into any future feature list.

Of particular interest in this (9X) regard are three suggestions from [RTAI]: a secondary standard for tasking optimizations (probably involving a uniform set of tasking use conventions [ARTEWG]); a secondary standard for low level tasking primitives; and the requirement for performance data from validated vendors (measuring, for example, delay, tasking, interrupts, and clocks).

Any measurement effort should include the two secondary standards, and should use at least some of the methodology and feature tests of the required vendor measurement suite.

3.4.2. Survey Review

A number of useful observations can be drawn from the ARTEWG user survey ([Survey]) about the overall nature of applications. These observations will be useful in the construction of composite or synthetic benchmarks.

- (1) The basic structure and behavior of applications is mixed, involving synchronous, asynchronous, and periodic behavior. Cyclic executives are common, but not pervasive.
- (2) The basic data types used by applications are mixed between numeric and nonnumeric, with roughly equal uses of integer, floating point, and character.
- (3) Interrupts and interrupt handling are important.
- (4) Preemptive scheduling is important.
- (5) Execution frequency guarantees are important.
- (6) Predictable execution times are important.
- (7) Applications make moderate use of global structures.
- (8) Applications make moderate use of dynamic allocation and garbage collection.
- (9) Applications make moderate use of exceptions.

Note that for these observations the importance of the various features was determined by the number of applications (percentage of total respondents) using them.

3.4.3. Measurement Strategies

We now review the three aspects of real-time performance, and how they can be measured.

First, and most obviously, is overhead. This is usually expressed in a speed metric. This quantity is relatively easy to measure.

Second is control. The desire for control is due both to the nature of real-time applications and Ada's complexity and relatively abstract definition. Control is hard to measure mechanically. It is usually the presence or absence of features that, at least with the current LRM, are implementation dependent. True evaluation of control may inevitably involve human review of features.

Third is predictability. Ultimately this often reduces to the lack of variance in performance measurements. Absent control issues, predictability can be measured through properly constructed overhead tests.

Reviewing previous approaches to measurements and benchmarking, we can envision a number of possible new measurement systems.

First, there is the generalization of the NAC and Hartstone approach. The NAC benchmark is a composite benchmark for a particular application, based on a cyclic executive. Hartstone is synthetic benchmark with a number of different configurations, based on a cyclic executive. The obvious generalization is to combine the two. The result would be a composite benchmark based on a cyclic executive, with a number of different configurations, and with the individual features selected and parameterized, so that they could be tailored to a specific application. Thus NAC would be subsumed by this new benchmark. Hartstone would not be, because it is based on the synthetic Dhrystone and Whetstone.

Second, there is a need for 9X feature benchmarks. Such a benchmark suite would be a natural extension of the Michigan and PIWG sets. In fact, we believe that any new language development effort should be accompanied by a measurement effort, as part of the proof of concepts. Hardware and applications are developed this way, with both fault/verification and performance test suites.

Third, not all real-time systems are built around cyclic executives. Thus a synthetic benchmark, similar to the *stones or Hennessy, is conceivable. It would exercise some combination of the above identified features. It could be constructed like the Hennessy set, and be composed of a number of small artificial applications.

Fourth, and most ambitious, a benchmark strategy could be based on survey questions (such as those in [Survey]). A catalog of features and overall test structures (such as cyclic executive variants) could be collected and parameterized (the PIWG and NAC sets would make a reasonable start). Users, by answering questions about their applications, would cause the system to construct a benchmark set tailored to their needs. Users could select the level of feature detail they wanted to deal with, specifying either broad requirements, or selecting precise information about specific features. Such a tool, if extant, would clearly have been valuable to the BSY-2 project.

References

[LRM]

Reference Manual for the Ada Programming Language, DOD-STD-1815A, 1982.

[Performance]

Ada Performance Issues, Ada Performance Issues Working Group; Ada Letters, Volume X, Number 3, Winter 1990.

[Michigan]

'Toward Real-Time Performance Benchmarks for Ada'; Russell M. Clapp, Louis Duchesneau, Richard A. Volz, Trevor N. Mudge, Timothy Schultze; *Communications of the ACM*, August 1986; pp. 760-778.

[NAC]

'Ada Run-Time Environment Requirements and Scenario for Airborne Real-Time Applications'; *Naval Avionics Center TR-2422*, Indianapolis, Indiana; October 1987.

[PIWG]

'PIWG Measurement Methodology'; Daniel Roy; *Ada LETTERS*, Volume X, Number 3, Winter 1990; pp. 72-90.

[NGCR]

Operating System Interface Standard Requirements; Version 2.0, 21 December 1989.

[ARTEWG]

'Summary of ARTEWG Meeting'; Alan Burns; *Ada LETTERS*, Volume IX, Number 5, July/August 1989; pp. 35-36.

[PRW]

Ada 9X Project Report; Ada 9X Project Requirements Workshop; June 1989.

[RTAI]

The Third International Workshop on Real-Time Ada Issues; *Ada LETTERS*, Volume X, Number 4, Spring 1990.

[CCS]

CCS MK 2 Software Standards and Procedures Manual; Appendix 1: Ada Style Guide; NAVSEA 0967-LP-022-0540.

[BSY-2]

Assessment of the Development Program for the AN/BSY-2 Submarine Combat System; J.N. Donis, J.P. Penell; IDA Paper P-2355.

[Survey]

First Annual Survey of Mission Critical Application Requirements for Ada Runtime Environments; Ada Runtime Environment Working Group; 1 December 1987.

[C&M]

Chapters 1, 2, 3, and 4; Russell M. Clapp, Trevor Mudge; *Ada LETTERS*, Volume X, Number 3, Winter 1990; pp. 10-32.

[Hartstone]

'Hartstone: Synthetic Benchmark Requirements for Hard Real-Time Applications'; Nelson Weiderman; *Ada LETTERS*, Volume X, Number 3, Winter 1990; pp. 126-136.

[CFA]

'Measurement and Evaluation of Alternative Computer Architectures'; Samuel H. Fuller, William E. Burr; *Computer*, October 1977; pp. 24-35.

[Optimizing]

'Issues in Optimizing Ada Code'; David Rosenfeld, Mike Ryer; *Ada LETTERS*, Volume X, Number 3, Winter 1990; pp. 60-71.

Copyright (c) W.R. Bush, 1990