Technical Report

CMU/SEI-87-TR-40
ESD-TR-87-203

Ada Performance Benchmarks on the Motorola MC68020: Summary and Results

Version 1.0

Patrick Donohoe

December 1987

Technical Report

CMU/SEI-87-TR-40

ESD-TR-87-203

December 1987

Ada Performance Benchmarks on the Motorola MC68020: Summary and Results

Version 1.0

Patrick Donohoe

Ada Embedded Systems Testbed Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/XRS
Hanscom AFB, MA 01731

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.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Karl H. Shingler
SEI Joint Program Office

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

Ada Performance Benchmarks on the Motorola MC68020: Summary and Results

Abstract: This report documents the results obtained from running the ACM SIGAda Performance Issues Working Group (PIWG) and the University of Michigan Ada performance benchmarks on a Motorola MC68020 microprocessor (MVME133 VMEmodule Monoboard Microcomputer), using the Systems Designers Ada-Plus, the TeleSoft TeleGen2, and the VERDIX VAX/VMS hosted cross-compilers. A brief description of the benchmarks and the test environment is followed by a discussion of some problems encountered and lessons learned. Wherever possible, the output of each benchmark program is also included.

1. Summary

The primary purpose of the Ada Embedded Systems Testbed (AEST) Project at the Software Engineering Institute (SEI) is to develop a solid in-house support base of hardware, software, and personnel to investigate a wide variety of issues related to software development for real-time embedded systems. Two of the most crucial issues to be investigated are the extent and quality of the facilities provided by Ada runtime support environments. The SEI support base will make it possible to assess the readiness of the Ada language and Ada tools to develop embedded systems.

The benchmarking/instrumentation subgroup was formed to:

- Collect and run available Ada benchmark programs from a variety of sources on a variety of targets.
- Identify gaps in the coverage and fill them with new test programs.
- Review the measurement techniques used and provide new ones if necessary.
- Verify software timings by inspection and with specialized test instruments.

This report documents the results of running two suites of Ada performance benchmarks on a Motorola MC68020 microprocessor using the Systems Designers Ada-Plus, the TeleSoft TeleGen2, and the VERDIX VAX/VMS hosted MC68020 cross-compilers. The benchmarks were the ACM SIGAda Performance Issues Working Group (PIWG) Ada benchmarks (excluding the compilation tests) and a subset of the University of Michigan Ada benchmarks. A description of the benchmarks, and the reasons for choosing them, are given in [6]. A summary description of each suite and the execution environment is given below. The problems encountered and the lessons learned from running the benchmarks are summarized. The output of each benchmark program is listed in the appendices, wherever possible, but the caveats discussed in the body of the report must be borne in mind when examining these results.

2. Discussion

2.1. The Performance Issues Working Group (PIWG) Ada Benchmarks

The PIWG benchmarks comprise many different Ada performance tests that were either collected or developed by PIWG under the auspices of the ACM Special Interest Group on Ada (SIGAda). In addition to language feature tests similar to the Michigan benchmarks, the PIWG suite contains composite synthetic benchmarks such as Whetstone [5], [9]; Dhrystone [15]; and a number of tests to measure speed of compilation. PIWG distributes tapes of the benchmarks to interested parties and collects and publishes the results in a newsletter.[1] Workshops and meetings are held during the year to discuss new benchmarks and suggestions for improvements to existing benchmarks.

2.2. The University of Michigan Ada Benchmarks

The University of Michigan benchmarks concentrate on techniques for measuring the performance of individual features of the Ada programming language. The development of the real-time performance measurement techniques and the interpretation of the benchmark results are based on the Ada notion of time. An article by the Michigan team [4] begins by reviewing the Ada concept of time and the measurement techniques used in the benchmarks. The specific features measured are then discussed, followed by a summary of the results obtained and an appraisal of those results. A follow-up letter about the Michigan benchmarks appears in [3].

2.3. Testbed Hardware and Software

The hardware used for benchmarking was DEC MicroVAX II host, running MicroVMS 4.4, linked to two 12.5 MHz Motorola MVME133 single board computers [11], [10] enclosed in a VMEbus chassis. The setup can be summarized as follows:

Host:	DEC MicroVAX II, running MicroVMS 4.4
Compilers:	Systems Designers (SD) Ada-Plus VAX/VMS MC68020, release 2B.01
	TeleSoft TeleGen2 for VAX/VMS embedded MC680X0 targets, release 3.13
	VERDIX Ada Development System (VADS), release 5.41(h)
Targets:	2 Motorola MVME133 VMEmodule 32-Bit Monoboard Microcomputers, each having a 12.5 MHz MC68020 microprocessor (one wait state), a 12.5 MC68881 floating-point co-processor, and 1 megabyte of RAM

Each MVME133 board had a debug serial port and two additional serial I/O ports that were connected to their counterparts on the host. The SD [12], TeleGen2 [13], and VERDIX [14] cross-compilation systems contained tools for compiling, linking, downloading, and executing target programs. Each also had a cross-debugger.

To verify benchmark timing results, one of the MVME133 boards was connected to a Gould K115 logic analyzer [8]. Although designed primarily for testing hardware, the analyzer can be con-

[1] The benchmarks came from the PIWG distribution tape known as TAPE_8_31_86. The name, address, and telephone number of the current chairperson of the PIWG can be found in Ada Letters, a bimonthly publication of SIGAda, the ACM Special Interest Group on Ada.

figured for use with an MC68020 microprocessor so that data, address, and control-line transitions can be captured. The device also provides a symbolic disassembler so that assembly language instructions rather than, say, data addresses, can be monitored. This latter feature, however, proved to be impractical for timing purposes because the disassembler interface removed timing information during data capture. The analyzer is a far from ideal tool for software debugging, but it has been used successfully for benchmark timing, albeit to a limited extent.

2.4. Running the Benchmarks

Command files to compile, link, download, and execute the benchmarks were written. For each cross-compiler, the link phase included commands to define the memory layout, e.g., program placement and stack and heap sizes. Tests were run singly, rather than in groups, because of problems noted below. The Systems Designers cross-compiler was the first one used in the testing, thenTeleGen2, and finally VERDIX.

Based on the earlier VAXELN benchmarking experience [7], it was decided that only selected PIWG and Michigan benchmarks would be run. This decision was reinforced by the discovery of the SD compiler's TARGET_IO package problem, described below. In practice, running selected tests quickly became an attempt to see how many benchmarks would run at all. Eventually, most of the PIWG tests and some of the Michigan tests were run. Fewer problems were encountered with the TeleSoft TeleGen2 compiler.

The SD cross-compiler provided optimization by default, although details are not given in the documentation. TeleGen2 and VERDIX do not optimize by default; optimization must be explicitly requested. The TeleGen2 and VERDIX times listed in this report are for un-optimized runs. The results for each cross-compiler are given in the appendices. A brief comparison of optimized versus un-optimized runs of selected PIWG tests compiled under TeleGen2 is also given in an appendix. All the benchmarks contained code to prevent the language feature of interest from being optimized away. Runtime checks were not suppressed, and, apart from the modifications to the output routines discussed below, the benchmarks' source code was not modified in any way. There was no performance difference evident between the two MVME133 boards.

2.5. Problems Encountered and Lessons Learned

2.5.1. The Dual Loop Problem

One major problem with benchmarking had already been encountered during the MicroVAX II runs of the University of Michigan benchmarks: negative time values were produced for some of the tests. An investigation revealed that the VAXELN paging mechanism lengthened the execution times of loops that spanned a page boundary. (Physical memory on the VAXELN target is divided into 512-byte pages; however, no swapping to disk took place since disk support was not included. The benchmarks were entirely resident in memory.) Thus the control loop of some benchmarks would actually take longer to run than the test loop, and the execution time of the language feature being measured (expressed as the difference of the test and control times) would sometimes be negative. A similar investigation of the MC68020 target, using the SD cross-compiler, revealed that the timing problem was present there also. The reason, in this case, is that the MC68020 memory accesses are by word, whereas the SD compiler placed the

loop statements without regard to word boundaries. Thus, depending on placement in memory, loops would sometimes require fewer memory accesses to execute. A more detailed discussion of the so-called "dual loop problem" may be found in [1]. A complete report on the problems encountered during the AEST benchmarking effort and a discussion of other possible benchmarking pitfalls is contained in [2].

2.5.2. Accuracy of Results

Another interesting issue is the accuracy of times reported by the PIWG benchmarks. One of the PIWG benchmark support packages, A000032.ADA, contains the body of the ITERATION package. This package is called by a benchmark program to calculate, among other things, the minimum duration for the test loop of a benchmark run. The minimum duration is computed to be the larger of 1 second, 100 times **System.Tick**, and 100 times **Standard.Duration'Small**. The idea appears to be (a) to run the benchmark for enough iterations to overcome the problem of the relatively coarse resolution of the **Calendar.Clock** function, and (b) to minimize the relative error of the timing measurement. There are two observations to be made about this:

- The times reported by the benchmark programs are printed with an accuracy of one-tenth of a microsecond; however, merely running the test for a specific minimum duration does not guarantee this degree of accuracy. If the clock resolution is 10 milliseconds, for example, and the desired accuracy is to within one microsecond, then the test should be run for 10,000 iterations. For Ada language features that execute in tens of microseconds, running for a specific duration may ensure enough iterations for accuracy to within one microsecond; this is not so for language features that take longer.
- Since **System.Tick** is used in the minimum duration calculation, the implicit assumption seems to be that **System.Tick** is equivalent to one tick of **Calendar.Clock**. This is true for SD but not for TeleGen2 or VERDIX. For the TeleGen2 MC68020 cross-compiler, **System.Tick** is 10 milliseconds, but the resolution of **Calendar.Clock**, as determined by the University of Michigan calibration test, is 100 milliseconds. Thus to obtain results accurate to one microsecond when using the TeleGen2 **Calendar.Clock**, tests should iterate 100,000 times. For VERDIX, **System.Tick** is 10 milliseconds and the resolution of **Calendar.Clock** is 61 microseconds, so the accuracy of timing measurements is actually much better than plus or minus one microsecond.

In general, the accuracy of the PIWG and Michigan benchmarks is to within one tick of **Calendar.Clock** divided by the number of iterations of the benchmark (see the "Basic Measurement Accuracy" section of the U. Michigan report). The University of Michigan benchmarks typically run for 10,000 iterations, and so are accurate to better than 1 microsecond for SD Ada-Plus (7.8 millisecond **Calendar.Clock** resolution). For TeleGen2, however, they are accurate to the nearest 10 microseconds. Also, the task creation tests and some of the dynamic storage allocation tests run for fewer iterations, probably because of the amount of storage they use up, so the accuracy is further reduced. The reduction in accuracy is noted in the relevant sections in the TeleGen2 results appendix. For the PIWG tests, which run for varying iteration counts, a table of iteration counts and resultant accuracy is provided in both the SD and TeleGen2 results appendices.

2.5.3. The SD Cross-Compiler

Other problems encountered during this benchmarking effort related to the varying degrees of difficulty experienced in getting the cross-compilers installed and running, and the benchmarks running on the MC68020 targets. The SD cross-compiler had its own special problem: programs compiled using the SD cross-compiler had to use a TARGET_IO package, instead of TEXT_IO, to produce output from the target machine. This meant that all benchmark output statements had to be converted to use the routines provided by TARGET_IO. An additional problem was then discovered: the routine to print floating-point numbers never produced any output. Examination of the source of TARGET_IO (which was fortunately provided with the SD product) revealed that the routine had a null body. Rather than write a floating-point routine, a quick solution was to scale up the timing results in microseconds and use the integer output routine. This meant that times would be to the nearest microsecond instead of to the theoretically-achievable nearest tenth of a microsecond. For the PIWG benchmarks, resolving the shortcomings of TARGET_IO was only a matter of changing a single general-purpose output program. For the Michigan benchmarks, however, it would have meant changing each of the output routines associated with the individual tests. For this reason, not all of the Michigan benchmarks were converted for the SD cross-compiler.

2.5.4. Timing with the Logic Analyzer

Problems arose when attempts were made to verify some of the timing results using the logic analyzer. These included: identifying the beginning and end of the assembly code generated for the language feature being measured; computing the actual start and end addresses of this code by examining load maps, adding appropriate offsets, and allowing for the MC68020 word-bounding; and choosing the correct "window" to capture data. Because of these problems, only the Michigan and PIWG task rendezvous times for the SD cross-compiler have been verified using the logic analyzer; these times are within 5 percent of the times reported by the benchmark runs. More use of the logic analyzer will be made in the next phase of the AEST Project when a PC with a data storage and analysis package will be connected to the analyzer.

2.5.5. Miscellaneous

For all three cross-compilers, most of the tests that failed did so with a STORAGE_ERROR. Changing the stack and heap sizes and re-linking the programs resolved some of these problems, but not all. This proved to be one of the more tedious aspects of the benchmarking effort. Other time-consuming problems were a serial port data overrun problem and a VADS downloading problem. The host and target serial ports could not operate all of the time at 9600 baud; intermittent data overrun messages would be produced during or after benchmarking runs. They were eventually made to work at 1200 baud. The VADS downloading problem - intermittent "TDM is not responding" error messages when attempting to download tests using the debugger - turned out to be a problem with VADS software. It will be corrected in the next release (version 5.5). VADS also suffered from the fact that it didn’t reset the timer (to disable interrupts) on exit from a benchmark program, so that the next benchmark would always fail with a "stopped in init_clock" error message. The workaround was to reset the timer manually before exiting the debugger. Putting this reset command in a debugger command file, after the download and run commands, alleviated some of the tedium of running the VADS tests. For some unknown reason, this also greatly reduced the intermittent TDM errors, allowing (finally) the VADS-compiled benchmarks to be run.

Comparison of the results from the most closely equivalent PIWG and Michigan benchmarks has been hindered by the accuracy problem and the dual loop problem. Even when the correction factors are applied to take care of the former, the precise effects of the dual loop problem on each benchmark program are not known. Thus, the disparity between the times reported for the Michigan task rendezvous test, R_REND, and the PIWG T000001 test has yet to be resolved. For the earlier VAXELN testing, T000001 took about 5 percent longer than R_REND to execute. For the MC68020 testing, R_REND took about 5 percent longer than T000001 when compiled with TeleGen2, but only about 1 percent longer when compiled with SD Ada-Plus. (For VERDIX, the numbers were equal, to the nearest microsecond.) It may be possible to resolve this issue by employing a timing scheme that uses the fast timers on the MVME133 board and eliminates the need for dual looping; a benchmark test harness that does this is currently being developed by the AEST Project.

A question prompted by the preceding discussion is: What exactly is being measured in a task rendezvous? The timing calls to **Calendar.Clock** are in the calling task, so they bracket a "round trip" rendezvous: the clock is read in Task A, Task A calls Task B, they proceed in parallel for the duration of the **accept** statement, and the control eventually returns to Task A where the second clock reading is taken. Depending on whether or not the called task is waiting at an **accept** statement, the "round trip" may involve up to three context switches. Are these context switches part of the rendezvous or not? If a more sophisticated timing scheme can measure the duration of the **accept** statement, is it measuring the "real" rendezvous, and can the result be compared with the PIWG or Michigan measurements? It is clear that more work needs to be done to resolve such issues.

As was the case with the VAXELN benchmarking effort, the major result of the MC68020 effort is not just a list of numbers to be taken at face value; rather, it is an appreciation of the problems and pitfalls facing the would-be benchmarker. It has also shown that it is difficult to generalize about benchmarking results; results must be interpreted within the context of the specific hardware and software environment.

References

[1] Altman, N. A., and Weiderman, N. H.
Timing Variation in Dual Loop Benchmarks.
Technical Report SEI-87-TR-21, Software Engineering Institute, September, 1987.

[2] Altman, N. A.
Factors Causing Unexpected Variations in Ada Benchmarks.
Technical Report SEI-87-TR-22, Software Engineering Institute, September, 1987.

[3] Broido, Michael D.
Response to Clapp et al.: Toward Real-Time Performance Benchmarks for Ada.
Communications of the ACM 30(2):169-171, February, 1987.

[4] Clapp, Russell M., et al.
Toward Real-Time Performance Benchmarks for Ada.
Communications of the ACM 29(8):760-778, August, 1986.

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

[6] Donohoe, P.
A Survey of Real-Time Performance Benchmarks for the Ada Programming Language.
Technical Report SEI-87-TR-28, Software Engineering Institute, October, 1987.

[7] Donohoe, P.
Ada Performance Benchmarks on the MicroVAX II: Summary and Results.
Technical Report SEI-87-TR-27, Software Engineering Institute, September, 1987.

[8] *Gould K115 Logic Analyzer User's Manual.*
Gould, Inc., 1985.

[9] Harbaugh, S. and Forakis, J.
Timing Studies using a Synthetic Whetstone Benchmark.
Ada Letters 4(2):23-34, 1984.

[10] *MC68020 32-Bit Microprocessor User's Manual.*
Second edition, Motorola Inc., 1985.

[11] *MVME133 VMEmodule 32-Bit Monoboard Microcomputer User's Manual.*
First edition, Motorola Inc., 1986.

[12] *Ada-Plus VAX/VMS MC68020, Volume 1 & 2.*
Systems Designers PLC, 1987.

[13] *The TeleSoft Second Generation Ada Development System for VAX/VMS to Embedded MC680X0 Targets.*
TeleSoft, 1987.

[14] *VAX/VMS-Motorola 68000 Family Processors, Version 5.40,*
Verdix Corporation, 1987.

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

Appendix A: PIWG Benchmark Results, SD Cross-Compiler

These results are for benchmarks that were compiled with optimization enabled (the default for SD Ada-Plus). The SD documentation does not specify what kinds of optimization are performed. The PIWG G tests (Text_IO tests) and the Z tests (compilation tests) were not run.

Most of the PIWG tests that failed did so with a STORAGE_ERROR. The SD cross-compiler reported such exceptions as "unhandled exception #4." (#1 is CONSTRAINT_ERROR, #2 is NUMERIC_ERROR, #3 is PROGRAM_ERROR, and #5 is TASKING_ERROR.) It is believed that the solution to the STORAGE_ERROR problem lies in simply finding the right settings for the storage parameters (e.g., stack size and heap size) of the target environment definition file.

The output of each PIWG benchmark program contains a terse description of the feature being measured. For any further details, the user must inspect the benchmark code. The reported "wall time" is based on calls to the **Calendar.Clock** function. The reported "CPU time" is based on calls to the PIWG function CPU_TIME_CLOCK. This function is intended to provide an interface to host-dependent CPU time measurement functions on multi-user systems where calls to **Calendar.Clock** might return misleading results. For the SD MC68020 tests, the basic version of CPU_TIME_CLOCK, which simply calls **Calendar.Clock**, was used.

Because of the issue of the accuracy of PIWG results (see Section 2.5), the table below is provided. Note that the actual iterations of the benchmarks are 100 times greater than the reported iteration counts. The reported counts are only for the main loop enclosing the control and test loops; these latter loops alway iterate 100 times. The accuracy delta is computed by dividing the resolution of the **Calendar.Clock** function (7812 microseconds) by the actual number of iterations. The accuracy of results is to within one microsecond or better when the actual iteration count equals or exceeds 7812.

Reported Iteration Count	Actual Iterations	Accuracy Delta in Microseconds
1	100	78.120
2	200	39.060
4	400	19.530
8	800	9.765
16	1600	4.882
32	3200	2.441
64	6400	1.220
128	12800	0.610

In general, for the PIWG tests compiled with SD Ada-Plus, the accuracy delta ranges from about 1 percent to 4 percent of the reported execution time. The exceptions are the loop overhead tests: for these the accuracy delta is much greater than the reported test times, and so the times must be rejected as unreliable.

A.0.1. Composite Benchmarks

A.0.1.1. The Dhrystone Benchmark

This test failed with an "unhandled exception #2," i.e., a NUMERIC_ERROR. The problem was traced to the generated assembly language where it was found that a 16-bit value was being picked up as a 32-bit value.

A.0.1.2. The Whetstone Benchmark

Two versions of the Whetstone benchmark [5] are provided. One uses the math library supplied by the vendor; the other has the math functions coded within the benchmark program so that the test can be run even when a math library is not supplied. SD Ada-Plus does not include a math library, so the second version of the Whetstone benchmark was used. "KWIPS" means Kilo Whetstones Per Second.

```
ADA Whetstone benchmark
A000093 using standard internal math routines

Average time per cycle :  4510 milliseconds
Average Whetstone rating :         222 KWIPS
```

A.0.1.3. The Hennessy Benchmark

This is a collection of benchmarks that are relatively short in terms of program size and execution time. Named after the person who gathered the tests, it includes such well-known programming problems as the Towers of Hanoi, Eight Queens, Quicksort, Bubble Sort, Fast Fourier Transform, and Ackermann's Function. The Hennessy benchmark, known as PIWG A000094, failed with a STORAGE_ERROR. Initial attempts to resolve the problem by modifying the stack and heap sizes were unsuccessful.

A.0.2. Task Creation

```
Test name:   C000001                        Class name:  Tasking
CPU time:     3691  microseconds
Wall time:    3691  microseconds         Iteration count:   8
Test description:
 Task create and terminate measurement
 with one task, no entries, when task is in a procedure
 using a task type in a package, no select statement, no loop



Test name:   C000002                        Class name:  Tasking
CPU time:     2753  microseconds
Wall time:    2753  microseconds         Iteration count:   4
Test description:
 Task create and terminate time measurement
 with one task, no entries when task is in a procedure,
 task defined and used in procedure, no select statement, no loop



Test name:   C000003                        Class name:  Tasking
CPU time:     2773  microseconds
Wall time:    2773  microseconds         Iteration count:   4
Test description:
 Task create and terminate time measurement
 task is in declare block of main procedure
 one task, no entries, task is in the loop
```

A.0.3. Dynamic Storage Allocation

```
Test name:   D000001                        Class name:  Allocation
CPU time:        9  microseconds
Wall time:       9  microseconds         Iteration count:   128
Test description:
 Dynamic array allocation, use and deallocation time measurement
 dynamic array elaboration, 1000 integers in a procedure
 get space and free it in the procedure on each call



Test name:   D000002                        Class name:  Allocation
CPU time:     15703  microseconds
Wall time:    15703  microseconds         Iteration count:   1
Test description:
 Dynamic array elaboration and initialization time measurement
 allocation, initialization, use and deallocation
 1000 integers initialized by  others=>1
```

The tests D000003 and D000004 failed with a STORAGE_ERROR; neither produced any output. The test description each would have produced is shown below.

```
Test name:   D000003                        Class name:  Allocation
Test description:
 Dynamic record allocation and deallocation time measurement
 elaborating, allocating and deallocating
 record containing a dynamic array of 1000 integers



Test name:   D000004                        Class name:  Allocation
Test description:
 Dynamic record allocation and deallocation time measurement
 elaborating, initializing by ( DYNAMIC_SIZE,(others=>1))
 record containing a dynamic array of 1000 integers
```

A.0.4. Exception Handling

There is no E000003 test in the PIWG 8/31/86 suite.

```
Test name:   E000001                        Class name:  Exception
CPU time:      32  microseconds
Wall time:     32  microseconds         Iteration count:  256
Test description:
 Time to raise and handle an exception
 exception defined and handled locally



Test name:   E000002                        Class name:  Exception
CPU time:      48  microseconds
Wall time:     48  microseconds         Iteration count:  128
Test description:
 Exception raise and handle timing measurement
 when exception is in a procedure in a package



Test name:   E000004                        Class name:  Procedure
CPU time:      68  microseconds
Wall time:     68  microseconds         Iteration count:   64
Test description:
 Exception raise and handle timing measurement
 when exception is in a package, four deep
```

A.0.5. Coding Style

```
Test name:   F000001                        Class name:  Style
CPU time:        3  microseconds
Wall time:       3  microseconds         Iteration count:   512
Test description:
 Time to set a boolean flag using a logical equation
 a local and a global integer are compared
 compare this test with F000002


Test name:   F000002                        Class name:  Style
CPU time:        4  microseconds
Wall time:       4  microseconds         Iteration count:   512
Test description:
 Time to set a boolean flag using an "if" test
 a local and a global integer are compared
 compare this test with F000001
```

A.0.6. Loop Overhead

The times reported here are unreliable because the number of actual iterations of the tests (400) means the times are accurate to plus or minus 19.53 microseconds.

```
Test name:   L000001                        Class name:  Iteration
CPU time:        6  microseconds
Wall time:       6  microseconds         Iteration count:   4
Test description:
Simple "for" loop time
for I in 1 .. 100 loop
time reported is for once through loop


Test name:   L000002                        Class name:  Iteration
CPU time:        4  microseconds
Wall time:       4  microseconds         Iteration count:   4
Test description:
Simple "while" loop time
while I <= 100 loop
time reported is for once through loop


Test name:   L000003                        Class name:  Iteration
CPU time:        4  microseconds
Wall time:       4  microseconds         Iteration count:   4
Test description:
Simple "exit" loop time
 loop I:=I+1; exit when I>100; end loop;
time reported is for once through loop
```

A.0.7. Procedure Calls

There is no P000008 or P000009 test in the PIWG 8/31/86 suite.

```
Test name:   P000001                        Class name:  Procedure
CPU time:        23  microseconds
Wall time:       23  microseconds        Iteration count:   256
Test description:
 Procedure call and return time (may be zero if automatic inlining)
 procedure is local
 no parameters


Test name:   P000002                        Class name:  Procedure
CPU time:       25  microseconds
Wall time:      25  microseconds         Iteration count:   256
Test description:
 Procedure call and return time
 Procedure is local, no parameters
 when procedure is not inlinable


Test name:   P000003                        Class name:  Procedure
CPU time:       18  microseconds
Wall time:      18  microseconds         Iteration count:   256
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 compare to P000002


Test name:   P000004                        Class name:  Procedure
CPU time:        16  microseconds
Wall time:       16  microseconds          Iteration count:   256
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 pragma INLINE used
 compare to P000001


Test name:   P000005                        Class name:  Procedure
CPU time:       20  microseconds
Wall time:      20  microseconds         Iteration count:   256
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 one parameter, in INTEGER
```

```
Test name:   P000006                        Class name:  Procedure
CPU time:       19  microseconds
Wall time:      19  microseconds         Iteration count:   256
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 one parameter, out INTEGER

Test name:   P000007                        Class name:  Procedure
CPU time:       23  microseconds
Wall time:      23  microseconds         Iteration count:   256
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 one parameter, in out INTEGER

Test name:   P000010                        Class name:  Procedure
CPU time:       40  microseconds
Wall time:      40  microseconds         Iteration count:   128
Test description:
 Procedure call and return time measurement
 compare to P000005
 10 parameters, in INTEGER

Test name:   P000011                        Class name:  Procedure
CPU time:      71  microseconds
Wall time:     71  microseconds          Iteration count:   128
Test description:
 Procedure call and return time measurement
 compare to P000005, P000010
 20 parameters, in INTEGER

Test name:   P000012                        Class name:  Procedure
CPU time:       49  microseconds
Wall time:      49  microseconds         Iteration count:   128
Test description:
 Procedure call and return time measurement
 compare with P000010 (discrete vs composite parameters)
 10 parameters, in MY_RECORD  a three component record

Test name:   P000013                        Class name:  Procedure
CPU time:       90  microseconds
Wall time:      90  microseconds         Iteration count:   64
Test description:
 Procedure call and return time measurement
 twenty composite "in" parameters
 package body is compiled after the spec is used
```

A.0.8. Task Rendezvous

The T000004 test produced no output except an "unhandled exception #4" message (STORAGE_ERROR). Test T000006 produced an "unhandled exception #5" message (TASKING_ERROR). The test descriptions that each test would have produced are included below.

```
Test name:   T000001                        Class name:  Tasking
CPU time:     480  microseconds
Wall time:    480  microseconds         Iteration count:  32
Test description:
 Minimum rendezvous, entry call and return time
 one task, one entry, task inside procedure
 no select



Test name:   T000002                        Class name:  Tasking
CPU time:     473  microseconds
Wall time:    473  microseconds         Iteration count:  32
Test description:
 Task entry call and return time measured
 one task active, one entry in task, task in a package
 no select statement



Test name:   T000003                        Class name:  Tasking
CPU time:     490  microseconds
Wall time:    490  microseconds         Iteration count:  16
Test description:
 Task entry call and return time measured
 two tasks active, one entry per task, tasks in a package
 no select statement



Test name:   T000004                        Class name:  Tasking
Test description:
 Task entry call and return time measured
 one tasks active, two entries, tasks in a package
 using select statement

 STORAGE_ERROR raised, no output produced
```

```
Test name:   T000005                        Class name:  Tasking
CPU time:     460  microseconds
Wall time:    460  microseconds         Iteration count:   4
Test description:
 Task entry call and return time measured
 ten tasks active, one entry per task, tasks in a package
 no select statement


Test name:   T000006                        Class name:  TASKING
Test description:
 Task entry call and return time measurement
 one task with ten entries, task in a package
 one select statement, compare to T000005

 TASKING_ERROR raised, no output produced



Test name:   T000007                        Class name:  Tasking
CPU time:     480  microseconds
Wall time:    480  microseconds         Iteration count:  32
Test description:
 Minimum rendezvous, entry call and return time
 one task, one entry
 no select
```

Appendix B: Selected U. Michigan Results, SD Cross-Compiler

In the results presented below, certain lines of output have been omitted for the sake of brevity. Many of the Michigan tests print out lines of "raw data," and the command files sometimes run a particular test many times; these are the lines that have been omitted. Also, some of the headings have been split over two lines to make them fit this document.

These tests were compiled with the compiler's default optimizations enabled. The SD documentation does not specify what these optimizations are. Except where otherwise stated, the accuracy of these results is better than one microsecond because the Michigan tests typically iterate 10,000 times. Times are reported in integral numbers because of the TARGET_IO problems discussed earlier in the report.

B.0.1. Clock Calibration and Overhead

For the SD Ada-Plus system, **System.Tick** and **Standard.Duration'Small** are both 7812 microseconds (1/128 seconds). The second-differencing clock calibration test gave the resolution of **Calendar.Clock** as 7812 microseconds also. The clock overhead test yielded:

```
Clock function calling overhead :   100  microseconds
```

B.0.2. Delay Statement Tests

For the delay tests, the actual delay is always the desired delay plus 7812 microseconds.

B.0.3. Task Rendezvous

For this test, a procedure calls the single entry point of a task; no parameters are passed, and the called task executes a simple **accept** statement. According to the Michigan report, it is assumed that such a rendezvous will involve at least two context switches.

```
Rendezvous time : No parameters passed
Number of iterations = 10000

_____________________________________________________________________

Task rendezvous time :    478 microseconds
_____________________________________________________________________
```

B.0.4. Task Creation

These tests measure the composite time taken to elaborate a task's specification, activate the task, and terminate the task. The coarse resolution of the clocks available at the time the tests were developed did not allow for measurement of the individual components of the test. Also, because these tests are run for 100 iterations, the times are accurate to 78.12 microseconds, or 0.078 milliseconds.

The third task creation benchmark — **new** Object, Task Type — failed with a STORAGE_ERROR.

```
Task elaborate, activate, and terminate time:
Declared object, no type
Number of iterations = 100

Task elaborate, activate, terminate time: 1 milliseconds
_____________________________________________________________________


Task elaborate, activate, and terminate time:
Declared object, task type
Number of iterations =  100

Task elaborate, activate, terminate time: 1 milliseconds
_____________________________________________________________________
```

B.0.5. Exception Handling

The times reported here are accurate to plus or minus 7.812 microseconds.

```
Number of iterations =  1000

              Exception Handler Tests
              =======================

      Exception raised and handled in a block

      15 uSEC.    User defined, not raised
      54 uSEC.    User defined
      62 uSEC.    Constraint error, implicitly raised
      54 uSEC.    Constraint error, explicitly raised
      62 uSEC.    Numeric error, implicitly raised
      54 uSEC.    Numeric error, explicitly raised
      54 uSEC.    Tasking error, explicitly raised

      Exception raised in a procedure and handled in the
      calling unit

      23 uSEC.    User defined, not raised
      62 uSEC.    User defined
      70 uSEC.    Constraint error, implicitly raised
      70 uSEC.    Constraint error, explicitly raised
      70 uSEC.    Numeric error, implicitly raised
      62 uSEC.    Numeric error, explicitly raised
      70 uSEC.    Tasking error, explicitly raised
```

B.0.6. Dynamic Storage Allocation

The times reported here are accurate to plus or minus 7.812 microseconds.

```
Number of iterations =  1000


                Dynamic Allocation with NEW Allocator
```

Time (microsec.)	# Declared	Type Declared	Size of Object
93	1	Integer array	1
101	1	Integer array	10
101	1	Integer array	100
93	1	Integer array	1000
93	1	2-D Dynamically bounded array	10

Appendix C: PIWG Benchmark Results, TeleSoft Cross-Compiler

These results are for benchmarks that were compiled without optimization enabled (the default for TeleGen2). All of the PIWG tests, with the exception of the task creation tests, ran without problems. The G tests (Text_IO tests) and the Z tests (compilation tests) were not run.

The output of each PIWG benchmark program contains a terse description of the feature being measured. For any further details, the user will have to inspect the benchmark code. The reported "wall time" is based on calls to the **Calendar.Clock** function. The reported "CPU time" is based on calls to the PIWG function CPU_TIME_CLOCK. This function is intended to provide an interface to host-dependent CPU time measurement functions on multi-user systems where calls to **Calendar.Clock** might return misleading results. For the SD MC68020 tests, the basic version of CPU_TIME_CLOCK, which simply calls **Calendar.Clock**, was used.

Because of the issue of the accuracy of PIWG results (see Section 2.5), the table below is provided. Note that the actual iterations of the benchmarks are 100 times greater than the reported iteration counts. The reported counts are only for the main loop enclosing the control and test loops; these latter loops alway iterate 100 times. The accuracy delta is computed by dividing the resolution of the **Calendar.Clock** function (100 milliseconds) by the actual number of iterations. The accuracy of results is to within one microsecond or better when the actual iteration count equals or exceeds 100,000.

Reported Iteration Count	Actual Iterations	Accuracy Delta in Microseconds
1	100	1000.000
2	200	500.000
4	400	250.000
8	800	125.000
16	1600	62.500
32	3200	31.250
64	6400	15.625
128	12800	7.812
256	25600	3.906
512	51200	1.953
1024	102400	0.976

Compared with the execution times of the language feature tests, TeleGen2's 100-millisecond Calendar.Clock is extremely coarse. Also, even when a test iterates enough times to be accurate to within a microsecond or two, the accuracy delta may still be a significant percentage of the execution time (see P000002 and P000003 results, for example).

C.0.1. Composite Benchmarks

C.0.1.1. The Dhrystone Benchmark

```
0.70996 is time in milliseconds for one Dhrystone
```

C.0.1.2. The Whetstone Benchmark

Two versions of the Whetstone benchmark [5] are provided. One uses the math library supplied by the vendor; the other has the math functions coded within the benchmark program so that the test can be run even when a math library is not supplied. TeleGen2 does not include a math library, so the second version of the Whetstone benchmark was used. "KWIPS" means Kilo Whetstones Per Second.

```
ADA Whetstone benchmark
A000093 using standard internal math routines

Average time per cycle :  4240.02 milliseconds
 Average Whetstone rating :         236 KWIPS
```

C.0.1.3. The Hennessy Benchmark

This is a collection of benchmarks that are relatively short in terms of program size and execution time. Named after the person who gathered the tests, it includes such well-known programming problems as the Towers of Hanoi, Eight Queens, etc. Execution times are reported in seconds.

```
A000094

   Perm   Towers   Queens   Intmm      Mm   Puzzle
   2.10     2.30     1.10    1.80   3.90     6.50

   Quick  Bubble   Tree    FFT    Ack
    0.80    1.70   1.10   8.60  27.10
```

C.0.2. Task Creation

The three task creation benchmarks failed with a STORAGE_ERROR. It is believed that the problem can be solved simply by changing the stack or heap size specifications in the linker options file. Another possibility is moving the debugger/receiver into PROM so that it leaves the maximum amount of RAM available for downloaded programs. Time did not permit the exploration of these possibilities.

C.0.3. Dynamic Storage Allocation

```
Test name:   D000001                        Class name:  Allocation
CPU time:       11.7  microseconds
Wall time:      11.7  microseconds         Iteration count:   256
Test description:
 Dynamic array allocation, use and deallocation time measurement
 dynamic array elaboration, 1000 integers in a procedure
 get space and free it in the procedure on each call


Test name:   D000002                        Class name:  Allocation
CPU time:     9499.5  microseconds
Wall time:    9499.5  microseconds         Iteration count:   2
Test description:
 Dynamic array elaboration and initialization time measurement
 allocation, initialization, use and deallocation
 1000 integers initialized by others=>1


Test name:   D000003                        Class name:  Allocation
CPU time:     1375.1  microseconds
Wall time:    1375.1  microseconds         Iteration count:   8
Test description:
 Dynamic record allocation and deallocation time measurement
 elaborating, allocating and deallocating
 record containing a dynamic array of 1000 integers


Test name:   D000004                        Class name:  Allocation
CPU time:    12993.3  microseconds
Wall time:   12993.3  microseconds         Iteration count:   1
Test description:
 Dynamic record allocation and deallocation time measurement
 elaborating, initializing by (DYNAMIC_SIZE,(others=>1))
 record containing a dynamic array of 1000 integers
```

C.0.4. Exception Handling

There is no E000003 test in the PIWG 8/31/86 suite.

```
Test name:   E000001                        Class name:  Exception
CPU time:       31.3  microseconds
Wall time:      31.3  microseconds         Iteration count:   256
Test description:
 time to raise and handle an exception
 exception defined locally and handled locally



Test name:   E000002                        Class name:  Exception
CPU time:      109.4  microseconds
Wall time:     109.4  microseconds         Iteration count:   128
Test description:
 Exception raise and handle timing measurement
 when exception is in a procedure in a package



Test name:   E000004                        Class name:  Procedure
CPU time:      374.8  microseconds
Wall time:     374.8  microseconds         Iteration count:    32
Test description:
 Exception raise and handle timing measurement
 when exception is in a package, four deep
```

C.0.5. Coding Style

```
Test name:   F000001                        Class name:  Style
CPU time:        3.9  microseconds
Wall time:       3.9  microseconds         Iteration count:   512
Test description:
 Time to set a boolean flag using a logical equation
 a local and a global integer are compared
 compare this test with F000002


Test name:   F000002                        Class name:  Style
CPU time:        0.0  microseconds
Wall time:       0.0  microseconds         Iteration count:   512
Test description:
 Time to set a boolean flag using an "if" test
 a local and a global integer are compared
 compare this test with F000001
```

C.0.6. Loop Overhead

The times reported here are unreliable because the number of actual iterations of the tests (800) means the times are accurate to plus or minus 125 microseconds.

```
Test name:   L000001                        Class name:  Iteration
CPU time:        1.3  microseconds
Wall time:       1.3  microseconds         Iteration count:   8
Test description:
Simple "for" loop time
for I in 1 .. 100 loop
time reported is for once through loop


Test name:   L000002                        Class name:  Iteration
CPU time:        0.0  microseconds
Wall time:       0.0  microseconds         Iteration count:   8
Test description:
Simple "while" loop time
while I <= 100 loop
time reported is for once through loop


Test name:   L000003                        Class name:  Iteration
CPU time:        1.3  microseconds
Wall time:       1.3  microseconds         Iteration count:   8
Test description:
Simple "exit" loop time
loop I:=I+1; exit when I>100; end loop;
time reported is for once through loop
```

C.0.7. Procedure Calls

There is no P000008 or P000009 test in the PIWG 8/31/86 suite.

```
Test name:   P000001                          Class name:  Procedure
CPU time:       11.7  microseconds
Wall time:      11.7  microseconds         Iteration count:   512
Test description:
 Procedure call and return time (may be zero if automatic inlining)
 procedure is local
 no parameters


Test name:   P000002                          Class name:  Procedure
CPU time:        7.8  microseconds
Wall time:       7.8  microseconds         Iteration count:   512
Test description:
 procedure call and return time
 procedure is local, no parameters
 when procedure is not inlinable


Test name:   P000003                          Class name:  Procedure
CPU time:        7.8  microseconds
Wall time:       7.8  microseconds         Iteration count:   512
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 compare to P000002


Test name:   P000004                          Class name:  Procedure
CPU time:        9.8  microseconds
Wall time:       9.8  microseconds         Iteration count:   512
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 pragma INLINE used
 Compare to P000001


Test name:   P000005                          Class name:  Procedure
CPU time:        9.7  microseconds
Wall time:       9.7  microseconds         Iteration count:   512
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 one parameter, in INTEGER
```

```
Test name:   P000006                        Class name:  Procedure
CPU time:        7.8  microseconds
Wall time:       7.8  microseconds         Iteration count:   512
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 one parameter, out INTEGER

Test name:   P000007                        Class name:  Procedure
CPU time:        9.8  microseconds
Wall time:       9.8  microseconds         Iteration count:   512
Test description:
 Procedure call and return time measurement
 procedure is in a separately compiled package
 one parameter, in out INTEGER

Test name:   P000010                        Class name:  Procedure
CPU time:       15.6  microseconds
Wall time:      15.6  microseconds         Iteration count:   256
Test description:
 Procedure call and return time measurement
 compare to P000005
 10 parameters, in INTEGER

Test name:   P000011                        Class name:  Procedure
CPU time:       31.3  microseconds
Wall time:      31.3  microseconds         Iteration count:   128
Test description:
 Procedure call and return time measurement
 compare to P000005, P000010
 20 parameters, in INTEGER

Test name:   P000012                        Class name:  Procedure
CPU time:       31.3  microseconds
Wall time:      31.3  microseconds         Iteration count:   256
Test description:
 Procedure call and return time measurement
 compare with P000010 (discrete vs composite parameters)
 10 parameters, in MY_RECORD a three component record

Test name:   P000013                        Class name:  Procedure
CPU time:       54.7  microseconds
Wall time:      54.7  microseconds         Iteration count:   128
Test description:
 Procedure call and return time measurement
 20 composite "in" parameters
 package body is compiled after the spec is used
```

C.0.8. Task Rendezvous

```
Test name:   T000001                        Class name:  Tasking
CPU time:      437.3  microseconds
Wall time:     437.3  microseconds         Iteration count:  32
Test description:
 Minimum rendezvous, entry call and return time
 one task, one entry, task inside procedure
 no select



Test name:   T000002                        Class name:  Tasking
CPU time:      437.3  microseconds
Wall time:     437.3  microseconds         Iteration count:  32
Test description:
 Task entry call and return time measured
 one task active, one entry in task, task in a package
 no select statement



Test name:   T000003                        Class name:  Tasking
CPU time:      468.9  microseconds
Wall time:     468.9  microseconds         Iteration count:  16
Test description:
 Task entry call and return time measured
 two tasks active, one entry per task, tasks in a package
 no select statement



Test name:   T000004                        Class name:  Tasking
CPU time:     1874.4  microseconds
Wall time:    1874.4  microseconds         Iteration count:   4
Test description:
 Task entry call and return time measured
 one task active, two entries, tasks in a package
 using select statement



Test name:   T000005                        Class name:  Tasking
CPU time:      450.0  microseconds
Wall time:     450.0  microseconds         Iteration count:   4
Test description:
 Task entry call and return time measured
 10 tasks active, one entry per task, tasks in a package
 no select statement
```

```
Test name:   T000006                        Class name:  TASKING
CPU time:     2299.3  microseconds
Wall time:    2299.3  microseconds         Iteration count:   1
Test description:
 Task entry call and return time measurement
 one task with ten entries, task in a package
 one select statement
 compare to T000005


Test name:   T000007                        Class name:  Tasking
CPU time:      468.9  microseconds
Wall time:     468.9  microseconds         Iteration count:  32
Test description:
 Minimum rendezvous, entry call and return time
 one task, one entry
 no select
```

Appendix D: Selected U. Michigan Results, TeleSoft Cross-Compiler

These results are for benchmarks that were compiled without optimization enabled (the default for TeleGen2).

In the results presented below, certain lines of output have been omitted for the sake of brevity. Many of the Michigan tests print out lines of "raw data," and the command files sometimes run a particular test many times; these are the lines that have been omitted. Also, some of the headings have been split over two lines to make them fit this document.

These tests were run close to the deadline for this report, so there was no time to re-run tests that caused problems. It is likely that for tests that failed with a STORAGE_ERROR, the solution is simply to increase the stack or heap size and re-run them. The subprogram overhead tests were not run. Of the dynamic storage allocation tests, only the array allocation tests were run. Except where otherwise stated, the accuracy of these tests is plus or minus 10 microseconds (100 millisecond clock resolution divided by 10,000 iterations).

D.0.1. Clock Calibration and Overhead

For the TeleGen2 cross-compiler, **System.Tick** is 10 milliseconds, and **Standard.Duration'Small** is 60 microseconds. The clock calibration test reported the resolution of **Calendar.Clock** as 0.10009 seconds, or approximately 100 milliseconds. The clock overhead test yielded:

```
Clock function calling overhead :   349.95  microseconds
```

D.0.2. Delay Statement Tests

The delay statement tests failed with a STORAGE_ERROR.

D.0.3. Task Rendezvous

For this test, a procedure calls the single entry point of a task; no parameters are passed, and the called task executes a simple **accept** statement. According to the Michigan report, it is assumed that such a rendezvous will involve at least two context switches.

```
Rendezvous time : No parameters passed
Number of iterations =  10000

_____________________________________________________________________

Task rendezvous time :     480.0 microseconds
_____________________________________________________________________
```

D.0.4. Task Creation

These tests measure the composite time taken to elaborate a task's specification, activate the task, and terminate the task. The coarse resolution of the clocks available at the time the tests were developed did not allow for measurement of the individual components of the test. Also, because these tests are run for only 100 iterations, the times are accurate to plus or minus 1000 microseconds, or 1 millisecond.

The task creation test below that failed with a TASKING_ERROR was supposed to allocate a task object using the **new** allocator.

```
Task elaborate, activate, and terminate time:
Declared object, no type
Number of iterations =  100

Task elaborate, activate, terminate time:     2.0 milliseconds
_____________________________________________________________________


Task elaborate, activate, and terminate time:
Declared object, task type
Number of iterations =  100

Task elaborate, activate, terminate time:     1.0 milliseconds
_____________________________________________________________________


>>> Unhandled exception: TASKING_ERROR
```

D.0.5. Exception Handling

The times reported here are accurate to plus or minus 100 microseconds.

```
Number of iterations =  1000

             Exception Handler Tests
             =======================

      Exception raised and handled in a block

         0.0 uSEC.    User defined, not raised
         0.0 uSEC.    User defined
       100.1 uSEC.    Constraint error, implicitly raised
         0.0 uSEC.    Constraint error, explicitly raised
        99.1 uSEC.    Numeric error, implicitly raised
         0.0 uSEC.    Numeric error, explicitly raised
         0.0 uSEC.    Tasking error, explicitly raised

      Exception raised in a procedure and handled in
      the calling unit

         0.0 uSEC.    User defined, not raised
       200.2 uSEC.    User defined
       299.3 uSEC.    Constraint error, implicitly raised
       200.2 uSEC.    Constraint error, explicitly raised
       299.3 uSEC.    Numeric error, implicitly raised
       200.2 uSEC.    Numeric error, explicitly raised
       299.3 uSEC.    Tasking error, explicitly raised
```

D.0.6. Time and Duration Math

```
Number of iterations =  10000


         TIME and DURATION Math
         ======================
   uSEC.   Operation

   50.05   Time := Var_time + var_duration
   59.96   Time := Var_time + const_duration
   79.98   Time := Var_duration + var_time
   89.89   Time := Const_duration + var_time
   89.89   Time := Var_time - var_duration
   80.08   Time := Var_time - const_duration
   29.93   Duration := Var_time - var_time
   10.01   Duration := var_duration + var_duration
    9.91   Duration := Var_duration + const_duration
    9.91   Duration := Const_duration + var_duration
    0.00   Duration := Const_duration + const_duration
   10.01   Duration := Var_duration - var_duration
   10.01   Duration := Var_duration - const_duration
   10.01   Duration := Const_duration - var_duration
   -0.01   Duration := Const_duration - const_duration
```

D.0.7. Dynamic Storage Allocation

Because of time constraints, the first set of dynamic storage allocation tests (dynamic allocation in a declarative region) was not run. Of the second set, only the array allocation tests were run; results are listed below. The times reported are accurate to plus or minus 100 microseconds.

```
Number of iterations =  1000
```

Dynamic Allocation with NEW Allocator

Time (microsec.)	# Declared	Type Declared	Size of Object
290.0	1	Integer array	1
290.0	1	Integer array	10
290.0	1	Integer array	100
290.0	1	Integer array	1000
310.0	1	1-D Dynamically bounded array	1
310.0	1	1-D Dynamically bounded array	10
340.0	1	2-D Dynamically bounded array	1
340.0	1	2-D Dynamically bounded array	100
390.0	1	3-D Dynamically bounded array	1
390.0	1	3-D Dynamically bounded array	1000

D.0.8. Memory Management

No timing results are produced by these tests; they are used to determine whether or not garbage collection takes place. They attempt to allocate up to ten million integers by successively allocating 1000-integer arrays using the **new** allocator. Only the last test explicitly attempted to free any allocated storage (using UNCHECKED_DEALLOCATION). The tests were designed either to report how much storage they allocated before the expected STORAGE_ERROR exception occurred, or a message saying they had succeeded. Running the tests confirmed that garbage collection did not occur; reclamation of storage is only done when explicitly requested. The output of the three tests is shown below.

```
Memsize:  127 arrays of 1_000 integers allocated

>>> Unhandled exception: STORAGE_ERROR  (allocator failure)


Implicit deallocation:  127 arrays of 1_000 integers allocated

>>> Unhandled exception: STORAGE_ERROR  (allocator failure)


Storage is reclaimed by calling UNCHECKED_DEALLOCATION,
or the memory space is larger than 10_000_000 INTEGER units
```

An additional test included with the memory management tests uses a first differencing scheme to determine the scheduling discipline of the target operating system. This test was not run because it was already known that TeleGen2 is a pre-emptive system.

Appendix E: PIWG Benchmark Results, VERDIX Cross-compiler

These results are for benchmarks which were compiled without optimization enabled (the default for VADS). All of the PIWG tests, with the exception of the task creation tests, ran without problems. The G tests (Text_IO tests) and the Z tests (compilation tests) were not run.

The output of each PIWG benchmark program contains a terse description of the feature being measured. For any further details, the user will have to inspect the benchmark code. The reported "Wall Time" is based on calls to the **Calendar.Clock** function. The reported "CPU Time" is based on calls to the PIWG function CPU_TIME_CLOCK. This function is intended to provide an interface to host-dependent CPU time measurement functions on multi-user systems where calls to **Calendar.Clock** might return misleading results. For the VADS MC68020 tests, the basic version of CPU_TIME_CLOCK, which simply calls **Calendar.Clock**, was used.

Since the resolution of **Calendar.Clock** under VADS is 61 microseconds, there is no accuracy problem; an accuracy of plus or minus 1 microsecond can be achieved with 61 iterations of a test. PIWG benchmarks run for at least 100 actual iterations (reported by the program as an iteration count of 1). Most of the PIWG benchmarks compiled under VADS ran for many more than 100 iterations.

E.0.1. Composite Benchmarks

E.0.1.1. The Dhrystone Benchmark

```
0.84936 is time in milliseconds for one Dhrystone
```

E.0.1.2. The Whetstone Benchmark

Two versions of the Whetstone benchmark [5] are provided. One uses the math library supplied by the vendor, the other has the math functions coded within the benchmark program so that the test can be run even when a math library is not supplied. VADS does not include a math library, so the second version of the Whetstone benchmark was used. "KWIPS" means Kilo Whetstones Per Second.

```
ADA Whetstone benchmark
A000093 using standard internal math routines

Average time per cycle :  6945.84 milliseconds
 Average Whetstone rating :         144 KWIPS
```

E.0.1.3. The Hennessy Benchmark

This is a collection of benchmarks that are relatively short in terms of program size and execution time. Named after the person who gathered the tests, it includes such well-known programming problems as the Eight Queens problem, the Towers of Hanoi, Quicksort, Bubble Sort, Fast Fourier Transform, and Ackermann's Function. Execution times are reported in seconds.

```
A000094

   Perm   Towers   Queens   Intmm      Mm   Puzzle
   3.20     3.85     2.70    3.18    6.81    14.25

   Quick  Bubble   Tree     FFT     Ack
    2.86    5.98   1.98   16.94   36.58
```

E.0.2. Task Creation

The three task creation benchmarks failed with a STORAGE_ERROR. It is believed that the problem can be solved simply by changing the memory layout specifications in the linker directives file.

E.0.3. Dynamic Storage Allocation

```
Test Name:   D000001                        Class Name:  Allocation
CPU Time:       34.6  microseconds
Wall Time:      34.6  microseconds.         Iteration Count:   128
Test Description:
 Dynamic array allocation, use and deallocation time measurement
 Dynamic array elaboration , 1000 integers in a procedure
 get space and free it in the procedure on each call



Test Name:   D000002                        Class Name:  Allocation
CPU Time:     3844.9  microseconds
Wall Time:    3844.9  microseconds.         Iteration Count:   4
Test Description:
 Dynamic array elaboration and initialization time measurement
 allocation, initialization, use and deallocation
 1000 integers initialized by  others=>1



Test Name:   D000003                        Class Name:  Allocation
CPU Time:     9144.8  microseconds
Wall Time:    9144.5  microseconds.         Iteration Count:   2
Test Description:
 Dynamic record allocation and deallocation time measurement
 elaborating, allocating and deallocating
 record containing a dynamic array of 1000 integers



Test Name:   D000004                        Class Name:  Allocation
CPU Time:    12719.7  microseconds
Wall Time:   12730.1  microseconds.         Iteration Count:   1
Test Description:
 Dynamic record allocation and deallocation time measurement
 elaborating, initializing by ( DYNAMIC_SIZE,(others=>1))
 record containing a dynamic array of 1000 integers
```

E.0.4. Exception Handling

There is no E000003 test in the PIWG 8/31/86 suite.

```
Test Name:   E000001                        Class Name:  Exception
CPU Time:     7035.2  microseconds
Wall Time:    7034.9  microseconds.         Iteration Count:   2
Test Description:
 Time to raise and handle an exception
 Exception defined locally and handled locally



Test Name:   E000002                        Class Name:  Exception
CPU Time:    12559.2  microseconds
Wall Time:   12560.4  microseconds.         Iteration Count:   1
Test Description:
 Exception raise and handle timing measurement
 when exception is in a procedure in a package



Test Name:   E000004                        Class Name:  Procedure
CPU Time:    29559.9  microseconds
Wall Time:   29550.1  microseconds.         Iteration Count:   1
Test Description:
 Exception raise and handle timing measurement
 when exception is in a package, 4 deep
```

E.0.5. Coding Style

```
Test Name:   F000001                        Class Name:  Style
CPU Time:        3.8  microseconds
Wall Time:       3.8  microseconds.         Iteration Count:   512
Test Description:
 Time to set a boolean flag using a logical equation
 a local and a global integer are compared
 compare this test with F000002


Test Name:   F000002                        Class Name:  Style
CPU Time:        3.8  microseconds
Wall Time:       3.8  microseconds.         Iteration Count:   512
Test Description:
 Time to set a boolean flag using an 'if' test
 a local and a global integer are compared
 compare this test with F000001
```

E.0.6. Loop Overhead

```
Test Name:   L000001                        Class Name:  Iteration
CPU Time:        4.3  microseconds
Wall Time:       4.3  microseconds.         Iteration Count:   8
Test Description:
Simple "for" loop time
for I in 1 .. 100 loop
time reported is for once through loop


Test Name:   L000002                        Class Name:  Iteration
CPU Time:        5.9  microseconds
Wall Time:       5.9  microseconds.         Iteration Count:   4
Test Description:
Simple "while" loop time
while I <= 100 loop
time reported is for once through loop


Test Name:   L000003                        Class Name:  Iteration
CPU Time:        6.4  microseconds
Wall Time:       6.4  microseconds.         Iteration Count:   4
Test Description:
Simple "exit" loop time
 loop I:=I+1; exit when I>100; end loop;
time reported is for once through loop
```

E.0.7. Procedure Calls

There is no P000008 or P000009 test in the PIWG 8/31/86 suite.

```
Test Name:   P000001                        Class Name:  Procedure
CPU Time:        7.2  microseconds
Wall Time:       7.2  microseconds.         Iteration Count:   512
Test Description:
 Procedure call and return time (may be zero if automatic inlining)
 procedure is local
 no parameters


Test Name:   P000002                        Class Name:  Procedure
CPU Time:       10.4  microseconds
Wall Time:      10.4  microseconds.         Iteration Count:   512
Test Description:
 Procedure call and return time
 Procedure is local, no parameters
 when procedure is not inlinable


Test Name:   P000003                        Class Name:  Procedure
CPU Time:        9.5  microseconds
Wall Time:       9.5  microseconds.         Iteration Count:   512
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 Compare to P000002


Test Name:   P000004                        Class Name:  Procedure
CPU Time:        0.1  microseconds
Wall Time:       0.1  microseconds.         Iteration Count:   512
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 pragma INLINE used. Compare to P000001


Test Name:   P000005                        Class Name:  Procedure
CPU Time:       11.0  microseconds
Wall Time:      11.0  microseconds.         Iteration Count:   512
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, in INTEGER
```

```
Test Name:   P000006                        Class Name:  Procedure
CPU Time:       12.6  microseconds
Wall Time:      12.6  microseconds.         Iteration Count:   512
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, out INTEGER

Test Name:   P000007                        Class Name:  Procedure
CPU Time:       13.1  microseconds
Wall Time:      13.2  microseconds.         Iteration Count:   512
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, in out INTEGER

Test Name:   P000010                        Class Name:  Procedure
CPU Time:       26.7  microseconds
Wall Time:      26.7  microseconds.         Iteration Count:   256
Test Description:
 Procedure call and return time measurement
 Compare to P000005
 10 parameters, in INTEGER

Test Name:   P000011                        Class Name:  Procedure
CPU Time:       46.4  microseconds
Wall Time:      46.5  microseconds.         Iteration Count:   128
Test Description:
 Procedure call and return time measurement
 Compare to P000005, P000010
 20 parameters, in INTEGER

Test Name:   P000012                        Class Name:  Procedure
CPU Time:      358.4  microseconds
Wall Time:     358.1  microseconds.         Iteration Count:   32
Test Description:
 Procedure call and return time measurement
 Compare with P000010 ( discrete vs composite parameters )
 10 paramaters, in MY_RECORD  a three component record

Test Name:   P000013                        Class Name:  Procedure
CPU Time:      707.5  microseconds
Wall Time:     707.4  microseconds.         Iteration Count:   16
Test Description:
 Procedure call and return time measurement
 twenty composite 'in' parameters
 The package body is compiled after the spec is used
```

E.0.8. Task Rendezvous

```
Test Name:   T000001                        Class Name:  Tasking
CPU Time:      328.7  microseconds
Wall Time:     329.0  microseconds.         Iteration Count:  32
Test Description:
 Minimum rendezvous, entry call and return time
 1 task 1 entry , task inside procedure
 no select



Test Name:   T000002                        Class Name:  Tasking
CPU Time:      328.4  microseconds
Wall Time:     328.7  microseconds.         Iteration Count:  32
Test Description:
 Task entry call and return time measured
 One task active, one entry in task, task in a package
 no select statement



Test Name:   T000003                        Class Name:  Tasking
CPU Time:      340.9  microseconds
Wall Time:     340.9  microseconds.         Iteration Count:  16
Test Description:
 Task entry call and return time measured
 Two tasks active, one entry per task, tasks in a package
 no select statement



Test Name:   T000004                        Class Name:  Tasking
CPU Time:      432.2  microseconds
Wall Time:     331.8  microseconds.         Iteration Count:  16
Test Description:
 Task entry call and return time measured
 One tasks active, two entries, tasks in a package
 using select statement



Test Name:   T000005                        Class Name:  Tasking
CPU Time:      328.4  microseconds
Wall Time:     328.7  microseconds.         Iteration Count:   4
Test Description:
 Task entry call and return time measured
 Ten tasks active, one entry per task, tasks in a package
 no select statement
```

```
Test Name:   T000006                        Class Name:  TASKING
CPU Time:      642.4  microseconds
Wall Time:     642.0  microseconds.         Iteration Count:   2
Test Description:
 Task entry call and return time measurement
 One task with ten entries , task in a package
 one select statement, compare to T000005


Test Name:   T000007                        Class Name:  Tasking
CPU Time:      328.7  microseconds
Wall Time:     329.0  microseconds.         Iteration Count:  32
Test Description:
 Minimum rendezvous, entry call and return time
 1 task 1 entry
 no select
```

Appendix F: Selected U. Michigan Results, VERDIX Cross-compiler

These results are for benchmarks which were compiled without optimization enabled (the default for VADS).

In the results presented below, certain lines of output have been omitted for the sake of brevity. Many of the Michigan tests print out lines of "raw data", and the command files sometimes run a particular test many times; these are the lines that have been omitted. Also, some of the headings have been split over two lines to make them fit this document.

Like the TeleGen2 tests, these tests were run close to the deadline for this report, so there was no time to re-run tests which caused problems. It is likely that for tests which failed with a STORAGE_ERROR, the solution is simply to increase the stack or heap size and re-run them. The subprogram overhead tests were not run. Of the dynamic storage allocation tests, only the array allocation tests were run.

F.0.1. Clock Calibration and Overhead

For the VADS cross-compiler, **System.Tick** is 10 milliseconds and **Standard.Duration'Small** is 61 microseconds. The clock calibration test reported the resolution of **Calendar.Clock** as 61 microseconds. The clock overhead test yielded:

```
Clock function calling overhead :  1269.69  microseconds
```

F.0.2. Task Rendezvous

For this test, a procedure calls the single entry point of a task; no parameters are passed, and the called task executes a simple **accept** statement. According to the Michigan report, it is assumed that such a rendezvous will involve at least two context switches.

```
Rendezvous Time : No Parameters Passed
Number of Iterations = 10000

Task Rendezvous Time :     328.9 microseconds.
```

F.0.3. Task Creation

These three tests measure the composite time taken to elaborate a task's specification, activate the task, and terminate the task. The coarse resolution of the clocks available at the time the tests were developed did not allow for measurement of the individual components of the test. The first two tests failed with a STORAGE_ERROR. The test producing the result below used the **new** allocator to create a task object

```
Task Elaborate, Activate, and Terminate Time:
NEW Object, Task Type
Number of Iterations = 100

Task elaborate, activate, terminate time:     1.6 milliseconds.
```

F.0.4. Exception Handling

```
Number of Iterations = 1000

              Exception Handler Tests
              =======================

      Exception raised and handled in a block

         0.0 uSEC.    User Defined, Not Raised
      5541.9 uSEC.    User Defined
     11173.9 uSEC.    Constraint Error, Implicitly Raised
     11287.9 uSEC.    Constraint Error, Explicitly Raised
     11223.9 uSEC.    Numeric Error, Implicitly Raised
     11462.9 uSEC.    Numeric Error, Explicitly Raised
     11523.9 uSEC.    Tasking Error, Explicitly Raised

      Exception raised in a procedure and handled in
      the calling unit

         2.9 uSEC.    User Defined, Not Raised
     10505.0 uSEC.    User Defined
     16007.0 uSEC.    Constraint Error, Implicitly Raised
     16192.0 uSEC.    Constraint Error, Explicitly Raised
     15993.0 uSEC.    Numeric Error, Implicitly Raised
     16179.0 uSEC.    Numeric Error, Explicitly Raised
     16176.0 uSEC.    Tasking Error, Explicitly Raised
```

F.0.5. Dynamic Storage Allocation

Because of time constraints, the first set of dynamic storage allocation tests (dynamic allocation in a declarative region) was not run. Of the second set, only the array allocation tests were run; results are listed below.

```
Number of Iterations = 1000

                Dynamic Allocation with NEW allocator
```

Time (microsec.)	# Declared	Type Declared	Size of Object
88.0	1	Integer Array	1
88.0	1	Integer Array	10
88.0	1	Integer Array	100
88.0	1	Integer Array	1000
135.9	1	1-D Dynamically Bounded Array	1
135.9	1	1-D Dynamically Bounded Array	10
155.0	1	2-D Dynamically Bounded Array	1
155.0	1	2-D Dynamically Bounded Array	100
177.0	1	3-D Dynamically Bounded Array	1
177.0	1	3-D Dynamically Bounded Array	1000

F.0.6. Memory Management

There are no timing results produced by these tests; they are used to determine whether or not garbage collection takes place. They attempt to allocate up to ten million integers, by successively allocating 1000-integer arrays using the **new** allocator. Only the last test explicitly attempted to free any allocated storage (using UNCHECKED_DEALLOCATION). The tests were designed either to report how much storage they allocated before the expected STORAGE_ERROR exception occurred, or a message saying they had succeeded. Running the tests confirmed that garbage collection did not occur; reclamation of storage is only done when explicitly requested. The (edited) output of the three tests is shown below.

```
Memsize:  31 arrays of 1_000 integers allocated

MAIN PROGRAM ABANDONED -- EXCEPTION "storage_error" RAISED


Implicit deallocation:  31 arrays of 1_000 integers allocated

MAIN PROGRAM ABANDONED -- EXCEPTION "storage_error" RAISED


Storage is reclaimed by calling UNCHECKED_DEALLOCATION,
or the memory space is larger than 10_000_000 INTEGER units
```

An additional test included with the memory management tests uses a first differencing scheme to determine the scheduling discipline of the target operating system. This test was not run because it was already known that VADS is a pre-emptive system. (The VADS implementation also gives a user the option of specifying time-slicing when configuring the run-time system.)

Appendix G: A Note on Optimization

For comparison of optimized versus unoptimized runs of the benchmarks, a number of the PIWG tests and the University of Michigan task rendezvous test were compiled using TeleGen2 both with and without optimization enabled. The level of optimization was that provided by the /OPTIMIZE qualifier [13].[2] In addition, the tests were made to run for 100,000 iterations or more to obtain an accuracy of at least plus or minus one microsecond using TeleGen2's 100-millisecond **Calendar.Clock**. The results are summarized below. All times are given in microseconds.

U. Michigan task rendezvous:

	R_REND:	Un-optimized:	485	Optimized:	485

PIWG task rendezvous:

	T000001:	Un-optimized:	458	Optimized:	457
	T000007:	Un-optimized:	459	Optimized:	456

PIWG exception handling:

	E000001:	Un-optimized:	26	Optimized:	25
	E000002:	Un-optimized:	100	Optimized:	102
	E000004:	Un-optimized:	358	Optimized:	358

PIWG subprogram overhead:

	P000013:	Un-optimized:	49	Optimized:	45

Thus, for this small sample, optimization appears to make little difference to benchmarks measuring individual language features. This is not really surprising, since the tests measure only one language feature, and they are designed to prevent optimizing compilers from removing the feature of interest from the benchmark. The "best" reduction in execution time for the above sample is 8 percent for P000013; the "worst" is the apparent gain of 2 percent for E000004.

A quick attempt was made to compare optimized and un-optimized runs of a composite benchmark, Dhrystone. When modified to run for 100,000 iterations (originally 10,000), both the optimized and un-optimized versions crashed with a STORAGE_ERROR. Running optimized and un-optimized versions of the original program produced the result below. Times are in milliseconds, not microseconds, and the results are accurate to plus or minus 0.01 milliseconds.

PIWG Dhrystone benchmark:

	A000091:	Un-optimized:	0.71	Optimized:	0.61

[2] The manual does not give much detail about the kinds of optimization provided by the /OPTIMIZE qualifier, apart from saying that it allows subprograms to be (a) called from parallel tasks, (b) called recursively, (c) expanded inline if the INLINE pragma is given, and (d) expanded inline automatically, whether or not an INLINE pragma is given. The global optimizer optimizes across-compiler collections of units.

Table of Contents