S11-61
136141

SOFTWARE ENGINEERING LABORATORY ADA PERFORMANCE STUDY–RESULTS AND IMPLICATIONS

P. 9

N93-17172

Eric W. Booth
Computer Sciences Corporation
Lanham-Seabrook, Maryland
(301) 794-1277

Michael E. Stark
NASA/Goddard Space Flight Center
Greenbelt, Maryland
(301) 286-5048

SUMMARY

The Ada Language Reference Manual (LRM) (Reference 1) states:

> "Ada was designed with three overriding concerns: program reliability and maintenance, programming as a human activity, and efficiency."

Initial implementations of Ada compilers and development environments tended to favor the first two concerns over the concern for efficiency. Similarly, initial (non-real-time, non-embedded) applications development using Ada as the programming language tended to favor maintainability, readability, and reusability.

As software systems become more sophisticated the need to predict, measure, and control the run time performance of systems in the Flight Dynamics Division (FDD) is a growing concern. The transition to Ada introduces performance issues that were previously nonexistent. More-over, this transition is often accompanied by the transition to object-oriented development (OOD), which has performance implications, independent of the programming language, that must be considered. To better understand the implications of new design and implementation approaches, the Software Engineering Laboratory (SEL) conducted an Ada performance study.

The SEL is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) to investigate the effectiveness of software engineering technologies applied to the development of applications software. The SEL was created in 1977 and has three organizational members: NASA/GSFC, Systems Development Branch; The University of Maryland, Computer Sciences Department; and Computer Sciences Corporation, Systems Development Operation.

The goals of the SEL are (1) to understand the software development process in the GSFC environments; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes the *Ada Performance Study Report* (Reference 2).

This paper describes the background of Ada in the FDD, the objectives and scope of the Ada Performance Study, the measurement approach used, the performance tests performed, the major test results, and the implications for future FDD Ada development efforts.

APPROACH TO MEASUREMENT

To measure the run-time performance of design alternatives and language features, two fundamental approaches were used. The first approach measured the run-time improvement of existing systems after an alternative had been incorporated into a baseline version of the system. The second approach used the ACM SIGAda PIWG test suite and added tests specific to the flight dynamics environment.

Overview

Benchmark programs are commonly used to evaluate the performance of design alternatives and language features. Such benchmark programs include (1) sample applications such as sorting programs or, as in the FDD, simulators, (2) programs to measure the overhead associated with a design alternative or language feature, and (3) synthetic benchmarks designed to measure the time needed to execute a representative mix of statements (e.g., Whetstone, Dhrystone) (Reference 6). The first approach used by this study falls into the first benchmark category, and the second approach falls into the last two.

To measure the overhead of a design alternative or language feature, the dual-loop approach is used to subtract the overhead associated with control statements that aid in performing the measurement. This approach uses a control loop and a test loop; the test loop contains everything contained in the control loop and the alternative being measured. A major factor in designing a dual-loop benchmark is compiler optimization. It is critical that the code generated by the compiler for both loops be identical except for the quantity being measured (Reference 7). In addition, it is necessary to ensure that the statement or sequence of statements being tested does not get optimized away.

Although the dual-loop approach can be used for synthetic benchmarks and applications, this technique is not required if the run time of the program is long in comparison to the system clock resolution (Reference 7). Instead, the CPU time can be sampled at the beginning of the program and again after a number of iterations of the program. The time for the benchmark/application is then (CPU_Stop - CPU_Start)/Number_Iterations. The same measurement can be achieved by submitting the test program to run as a batch job and obtaining the CPU time from the batch log file. This CPU time can then be divided by the number of times the sequence of statements being measured is executed in the main control loop of the test program.

It is important to understand the run-time environment in which the benchmarks are run when interpreting test results. VMS checks the timer queues once per second, which can affect measurement accuracy. Under VMS, the Ada run-time system is bundled with the release of the operating system and installed as a shareable executable image. Consequently, DEC Ada performance is directly dependent on the installed version of VMS. There is also a degree of uncertainty when using CPU timers provided in time-shared systems like VMS. In the presence of other jobs, CPU timers charge ticks to the running process when the wall clock is updated. It is therefore possible for time to be charged to active processes inaccurately because context switches can occur at any time. Finally, it cannot be assumed that running benchmarks for a hosted system in batch during low usage (such as, at 11 pm) guarantees standalone conditions (References 7 and 8). Therefore, the FD benchmarks to test individual design alternatives were run on the weekend to minimize these effects.

THE FIRST APPROACH -- SIMULATOR

Several of the design alternatives examined by this study were tested and analyzed in the context of two FDD simulators. Alternatives were chosen to be implemented in the context of these simulators for the following reasons:

1. They were simulator-specific, e.g., different ways of implementing the scheduler.
2. They could be implemented in an isolated part of the simulator where their impact could easily be measured using the VAX PCA.
3. They could be implemented in an isolated part of the simulator and still have a measurable effect on the time required for a 20-minute simulation run.

Baselined versions of the simulators were used to test each of the design alternatives. CPU times were obtained for 20-minute simulation runs of the baselined versions from the log files created by batch runs. PCA was used to obtain a profile of the simulators. These profiles showed what percentage of the CPU time was spent in each Ada package of the simulator. The VAX manual *Guide to VAX Performance and Coverage Analyzer* (Reference 9) contains more information on PCA.

Design alternatives were incorporated into the baselined versions of the simulators. New CPU times were obtained for 20-minute simulation runs from the log files created by batch runs and new profiles obtained using PCA. The following two figures show the accounting information contained in a batch log file and a sample of PCA output. From these two pieces of information, the impact of each design alternative was assessed.

Sample PCA Output

```
                      VAX Performance and Coverage Analyzer
                   CPU Sampling Data (11219 data points total) - "*"

Bucket Name          +----+----+----+----+----+----+----+----+----+----+
PROGRAM_ADDRESS\     |
 UTILITIES_ . . . .  |**************************************************   23.2%
 SIMULATION_SCHEDULE|***********************                              10.7%
 SEARCH_STRING  . .  |****************                                      7.5%
 SPACECRAFT_ATTITUDE|****************                                       7.5%
 DATABASE_MANAGER .  |***************                                       7.0%
 ADDING_UTILITIES .  |**********                                            4.7%
 EARTH_SENSOR . . .  |********                                              3.7%
 UTILITIES_LONG_  .  |******                                                3.0%
 DATABASE_TYPES_  .  |******                                                2.9%
 SPACECRAFT_WHEELS   |******                                                2.9%
 AOCS_PROCESSOR . .  |******                                                2.6%
 SPACECRAFT_EPHEMERI|*****                                                  2.5%
 ENVIRONMENTAL_TORQU|*****                                                  2.3%
 THRUSTERS  . . . .  |*****                                                 2.2%
 GEOMAGNETIC_FIELD   |*****                                                 2.2%
 DEBUG_COLLECTOR  .  |****                                                  2.1%
 MAGNETOMETER . . .  |****                                                  1.7%
 SADA . . . . . . .  |***                                                   1.4%
 SOLAR_SYSTEM . . .  |***                                                   1.2%
                     +----+----+----+----+----+----+----+----+----+----+
```

Sample Batch Log File Accounting Information

```
Accounting information:
 Buffered I/O count:          109      Peak working set size:  4096
 Direct I/O count:           1132      Peak page file size:   15304
 Page faults:               11766      Mounted volumes:           0
 Charged CPU time:    0 00:06:45.08    Elapsed time:     0 00:09:02.47
```

THE SECOND APPROACH -- PIWG

Design alternatives not isolated to a particular part of either of the simulators were tested using the PIWG structure of measurements. The PIWG structure of measurements is based on the concept of a control loop and a test loop. The test loop contains everything in the control loop and one alternative to be measured. The CPU time is sampled before the execution of each loop and after many iterations of each loop. If the test loop time duration is not considered stable, the process is repeated with a greater number of iterations; this is accomplished through an outer loop surrounding the test and control loops. To be considered stable, the test loop time duration must be greater than a predefined minimum time. If this condition is met, the test loop time duration is compared against the control loop time duration, and the number of iterations is compared against a predefined minimum number of iterations. If the test loop time is greater than the control loop time or the minimum number of iterations has been exceeded, the results are considered stable, and the CPU time for the design alternative is calculated. The time for the alternative is the difference between the amount of CPU time taken for the control loop and the amount of CPU time taken for the test loop, divided by the total number of iterations performed. Collecting control loop and test loop CPU times, calculating design alternative times, and testing for stability were done using PIWG's Iteration package in the test drivers for this study.

All test drivers used in this study were called three times from a main driver routine so that the CPU time for a given design alternative could be averaged for more accuracy. All results were averaged and recorded using PIWG's I/O package and report generator procedure. The following is a sample PIWG report.

Sample PIWG Report

```
Test Name:   Generic_A                       Class Name:  Matrix - Gen
CPU Time:      117.2  microseconds
Wall Time:     117.2  microseconds           Iteration Count:   128
                                             Number of samples: 3
Test Description:
Use of generic matrix processing
- Generic package for 3x3 matrix


Test Name:   Generic_C                       Class Name:  Matrix - Gen
CPU Time:      117.2  microseconds
Wall Time:     117.2  microseconds           Iteration Count:   128
                                             Number of samples: 3
Test Description:
Use of generic matrix processing
- NonGeneric package for 3x3 matrix
```

TEST OVERVIEW

Ten test groups were developed. Each test group represented a design or implementation issue relevant to current FDD applications. The test groups were chosen as a result of an in-depth analysis of several PCA runs with two FDD simulators. If a certain design alternative or language feature appeared to consume a relatively large portion of central processing unit (CPU) time or memory, it was analyzed, measured, and quantified in this study. The design alternatives or language features consuming a relatively small portion of CPU time or memory were not studied further. Therefore, the test groups presented here are intended to be a representative sampling, rather than an exhaustive sampling, of current design and implementation approaches. The test groups are presented in two categories: design-oriented tests and implementation-oriented tests.

Design-Oriented Tests

Following is a brief description of the purpose of each design test group performed on the Ada performance study.

Group 1: Scheduling. This test group contained three tests that addressed the run-time cost of various scheduling alternatives. This test compared a event-driven design against a time-driven design and a hard-coded design. The event-driven design maintains a prioritized (sorted) queue of event identifiers that specifies the time-step and next simulation event. The time-driven design iterates over an array of event identifiers for each fixed time-step. The hard-coded design contains the event (procedure) calls in the source code. With the event-driven design the user may vary the order and frequency of each event. In the time-driven design the user may only vary the order of the event. In the hard-coded design there are not options available to the user. The implications of the different approaches were analyzed and contrasted. The results of this test group provided the applications designers with information necessary to make trade-off decisions among flexibility, accuracy, and performance.

Group 2: Unconstrained Structures. Leaving data structures unconstrained allows greater user flexibility and enhances future reusability. However, the additional run-time code that may be generated can impose a significant run-time and memory overhead. This group measured the expense of unconstrained records and arrays and proposed viable alternatives.

Group 3: Initialization and Elaboration. This test group addressed initialization of static and dynamic data using various combinations of elaboration-time and execution-time alternatives. This test group was relevant for applications requiring minimal initialization time.

Group 4: Generic Units. The benefits of using generic units are reduced source-code size, encapsulation, information hiding, decoupling, and increased reuse (Reference 10). However, many Ada compilers implement this language feature poorly. This test group addressed the options available with the compiler implementation and how well these options were implemented.

Group 5: Conditional Compilation. The ability to include additional "debug code" in the delivered system adds to the system size and imposes a run-time penalty even if it is never used. The test group analyzed the current approach and proposed flexible alternatives for future systems. The results of this test group can have applications beyond "debug code" elimination.

Group 6: Object-Oriented Programming. Two of the fundamental principles of object-oriented programming (OOP) are polymorphism and inheritance. Ada does not directly support these principles. However, the designer may simulate the effect of inheritance and polymorphism through the use of variant records and enumeration types. These OOP principles, whether direct or indirect, incur certain run-time overhead and problems (Reference 11).

Implementation-Oriented Tests

Following is a brief description of the purpose of each implementation test group performed on the Ada performance study.

Group 7: Matrix Storage. The most basic, and perhaps the most common, mathematical expressions in flight dynamics applications involve matrix manipulations. This group addressed row-major versus column-major algorithms to quantify the performance implications.

Group 8: Logical Operators. The Ada LRM clearly defines the behavior of logical expression evaluation. The *Ada Style Guide* (Reference 12) recommends avoiding the short-circuit forms of logical operators for performance reasons. The implications of this recommendation in the flight dynamics environment were measured and analyzed.

Group 9: Pragma Inline. Flight dynamics simulators contain a large number of procedure and function calls to simple call-throughs and selectors. The overhead of making these calls can slow the performance of any simulator. This test measured the use of **pragma** INLINE as an alternative to calling a routine.

Group 10: String-to-Enumeration Conversion. Current flight dynamics simulators contain a central logical data base. The physical data are distributed throughout the simulator in the appropriate packages. The logical data base provides *keys* (strings) that map into the physical data. The logical data base converts these strings to the appropriate enumeration type to retrieve the corresponding data. This test assessed the performance implications of this approach.

Test Documentation

Each performance test in this report is described in this section in the following format:

Purpose. Each test was designed with a specific design or implementation alternative in mind. The rationale for the choice of the alternatives tested results from analysis of existing Ada systems developed in the FDD.

Method. Some tests were performed as changes to an existing system, while other tests were performed by creating new, special-purpose software. The basis for each method was one of two approaches: DEC's PCA measurement tool or the PIWG structure of measurements. The details of the method(s) used for each test are described.

Results. The result of executing a test is some combination of CPU time and object code size. Most tests were designed to measure the CPU run time in microseconds (μs). In some cases the object code size in bytes is relevant. The data resulting from each test run are provided.

Analysis. In many cases, detailed analysis of the test results is necessary to understand the implications for future projects. The analysis performed is summarized, and the implications are highlighted.

RESULTS

As a result of this performance study, more accurate estimation of run-time performance for future FDD simulators is possible. Assuming future dynamics simulators are similar in function to GOADA, a more accurate performance estimation is possible given the following information:

1. The run-time performance for a typical run of the GOADA simulator is 6 minutes, 45 seconds for a 20-minute simulation. This yields a 1:3 simulation time to real-time ratio.

2. The performance profile generated by PCA of a typical GOADA run shows the distribution of the CPU run time resource throughout the simulator.

3. The measured results of this study that lead to more efficient design and implementation alternatives.

The following table combines the results of the Ada Performance Study with the PCA performance profile of GOADA. Each row of the table is measured against the baseline of 6 minutes and 45 seconds of CPU time to perform a 20 minuter simulation.

Impact of Measured Performance Results on Dynamics Simulators

Alternative	GOADA	Study Results	Difference
1. Looping scheduler	10.7%	2.2%	8.5%
2. Bypass logical data base	14.5%	1.8%	12.7%
3. Conditional compile debug code	2.1%	0.0%	2.1%
4. Use static data structures	45.0%	13.0%	32.0%
5. Optimized utility packages	26.6%	5.3%	21.3%
Total Percentages	98.9%	22.3%	76.6%

The first row of the table shows the performance difference between the baseline scheduler in GOADA and the looping scheduler alternative (see test group 1, scheduling). Another option is to use the "hard-coded" approach for the scheduler. However, the hard-coded approach sacrifices all flexibility in the interest of performance. For this reason, the more flexible "looping" alternative is recommended.

The second row highlights the difference between accessing the logical data base and accessing the physical data directly (see test group 10, string-to-enumeration conversion). This striking improvement came from removing one string-to-enumeration type conversion from the main simulation loop. The third row recommends the conditionally compiled debug code (see test group 5, conditional compilation). The fourth row is the estimated result of using a static record structure instead of a dynamic structure in all simulator packages (see test group 2, unconstrained structures).

The fifth row is based on the result of comparing GOADA's baseline matrix multiply function to the optimized matrix multiply function (Reference 13). Since the FDD deals with mainly three dimensions, an optimized set of utilities can be developed on that basis. The fully optimized version required less than one-fifth of the CPU time required for the baseline version.

As this table shows, a dynamics simulator that is similar to GOADA and is implemented with the results of this study would consume 76.6 percent less CPU than the current version, more than quadrupling the speed. This would yield an upper bound estimate of 95 seconds to perform a 20-minute simulation run, or approximately a 1:13 simulation-time-to-real-time ratio.

This estimate is an upper bound for three reasons. First, this study examined a representative, rather than an exhaustive, list of design and implementation alternatives. That is, only those alternatives that held the most promise of a large performance difference were studied. There may be many other alternatives that offer only minor gains. However, the combined performance gain of all may be significant.

Second, coding optimizations to GOADA, or any simulator, were not studied. The goal of the study was to identify those design and implementation alternatives that lead to optimal systems. Line-by-line micro-optimizations on a simulator only provide information on final efficiency and lack the needed information on *how* to systematically predict and achieve that level of efficiency.

Finally, the DEC Ada 1.5-44 compiler is a relatively error-free first attempt at an Ada compilation system. The next generation of Ada compilers, which includes DEC Ada 2.0, are now available. These second-generation compilation system includes improvements to the optimizer and code-generator. For example, simply compiling GOADA using DEC Ada 2.0 improved the simulator's performance by 7.4%.

CONCLUSIONS

The following statements summarize the results of the Ada Performance Study:

- Design and implementation decisions that favored fidelity over efficiency were the largest contributor to poor run-time performance. The design should continually be reevaluated against evolving user requirements and specifications.
- Ada simulators in the FDD can be designed and implemented to achieve run times comparable to those of existing FDD FORTRAN simulators. Inefficient systems indicate problems in the system design or the compiler being used.
- Current Ada compilation systems still have inconvenient features that may contribute to poor performance. Organizations using Ada should use available performance-analysis tools to assess their compilation systems.

Design changes are much more expensive than coding changes during final system testing. Often due to schedule and budget constraints, design changes are impossible. Therefore the important implication of the Ada performance study results is that new technology (in this case Ada and OOD) requires performance prototyping and benchmarking early in the design phase even in seemingly simple or straightforward cases.

The *Ada Performance Study Report* (Reference 2) contains a detailed analysis of each alternative studied and summarizes the results of this analysis with specific performance recommendations for future OOD/Ada development efforts in the FDD. Different application domains may be able to apply these results and recommendations. However, this does not preclude the necessity for application domain specific prototyping and benchmarking to determine the application specific performance issues.

REFERENCES

1. *Ada Programming Language*, American National Standards Institute/Military Standard 1815A, 1983 (ANSI/MIL-STD-1815A-1983)

2. Goddard Space Flight Center, SEL-91-003, *Software Engineering Laboratory Ada Performance Study Report*, E. Booth and M. Stark, July 1991

3. Goddard Space Flight Center, SEL-88-003, *Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis*, K. Quimby, et al., prepared by Computer Sciences Corporation, December 1988

4. Goddard Space Flight Center, FDD/552-90/010, *Software Engineering Laboratory (SEL) Study of the System and Acceptance Test Phases of Four Telemetry Simulator Projects*, D. Cover, prepared by Computer Sciences Corporation, September 1990

5 Goddard Space Flight Center, SEL-89-005, *Lessons Learned in the Transition to Ada from FORTRAN at NASA/Goddard*, C. Brophy, University of Maryland, November 1989

6. Clapp, R., and T. Mudge, *Rationale, Chapter 1 - Introduction*, Ada Performance Issues, Ada Letters, A Special Issue, Association of Computing Machinery, New York, NY, ISBN 0-89791-354-x, vol. 10, no. 3, Winter 1990

7. Clapp, R., and T. Mudge, Rationale, *Chapter 3 - The Time Problem*, Ada Performance Issues, Ada Letters, A Special Issue, Association of Computing Machinery, New York, NY, ISBN 0-89791-354-x, vol. 10, no. 3, Winter 1990

8. Gaumer, D. and D. Roy, *Results, Reporting Test Results*, Ada Performance Issues, Ada Letters, A Special Issue, Association of Computing Machinery, New York, NY, ISBN 0-89791-354-x, vol. 10, no. 3, Winter 1990

9. Digital Equipment Corporation, *Guide to VAX Performance and Coverage Analyzer*, June 1989

10. Goddard Space Flight Center, FDD/552-90/045, *Extreme Ultraviolet Explorer (EUVE) Telemetry Simulator (EUVETELS) Software Development History*, E. Booth and R. Luczak, prepared by Computer Sciences Corporation, 1990

11. Booch, G., *Object Oriented Design*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, ISBN 0-8053-0091-0, 1991

12. Goddard Space Flight Center, SEL-87-006, *Ada Style Guide*, E. Seidewitz et al., June 1986

13. Burley, R., "Some Data from Ada Performance Study" internal FDD memorandum, September 1990