REUSING FORTRAN IN AN ADA DESIGN

William F. Parrish, Jr.
Naval Training Systems Center
Orlando, FL 32813-7100

Kent G. Trewick
Naval Training Systems Center
Orlando, FL 32813-7100

ABSTRACT

Many simulator processes and algorithms have been implemented in FORTRAN. Some examples are ocean models, aircraft avionics models, and sonar sensor models. As we begin writing training device software in Ada®, it is important that we consider reusing existing FORTRAN code. This is particularly true for FORTRAN based trainers undergoing major software modifications. Various techniques for interfacing Ada and FORTRAN designs are investigated. Benchmarks are presented comparing an all FORTRAN or all Ada implementation to a combined FORTRAN/Ada implementation. Problems concerned with calling FORTRAN subroutines from Ada procedures and tasks and vice versa are explored. Differences in arithmetic types between the two languages are also explored. Particular emphasis is placed on the effect that a combined Ada/FORTRAN implementation has on computer resources. This consideration is of major importance when modifying an existing trainer where spare time and memory may be very limited.

INTRODUCTION

Current Navy and DOD instructions require that weapon system training device software be written in Ada. In some instances the reuse of existing FORTRAN code should be considered. There are primarily two cases where this requires consideration. The first case is that of a major software modification where most of the existing software is written in FORTRAN. OPNAVINST 5200.28 requires that Ada be used for major upgrades. A major upgrade is defined by DOD Directive 3405.2 as a redesign or addition of one-third or more of the software. The other case where the reuse of FORTRAN is important is in those instances where it is desirable to use existing FORTRAN models. Many potentially reusable FORTRAN models exist. For example, the Naval Training Systems Center has an ocean model that is written in FORTRAN that has been furnished as Government Furnished Information on several training device contracts.

Numerous factors should be considered when integrating FORTRAN and Ada designs and code. This paper examines many of these factors. A series of benchmarks were run and results are compared. Memory requirements and execution times are compared between languages.

BENCHMARK DESCRIPTIONS AND RESULTS

Three benchmark programs were used. The first benchmark program is a simple program that computes the number of prime numbers in a specified range. This program was chosen because of simplicity and the ease with which execution time can be varied. The prime number program also facilitates easy comparison of arithmetic types. The second benchmark program is a mathematical routine that calculates a simple sine and exponential function using infinite series. This program was chosen because it represents a cyclic activity when used to generate a table of values. The third benchmark is a modified version of the Dhrystone (9) benchmark. Modifications were made to the benchmark by the University of Central Florida to closely approximate the mix of high level language statements found in a typical training simulator program.

The first two benchmarks were implemented with and without Ada tasks. This was done so that a comparison could be made between the execution times with tasks on a parallel processor machine and a single processor machine.

Several different implementations were considered. Each benchmark was implemented in both Ada and FORTRAN as well as a combination of Ada and FORTRAN where appropriate. In the combination implementations the Ada program calls a FORTRAN subroutine and the FORTRAN program calls an Ada procedure or subroutine. All benchmarks were run on a VAX 11/780 and many of the benchmarks were run on a Sequent Balance 8000, and a Zenith Z-248 with an 80287 math coprocessor. Sufficient time was not available to run all benchmarks on the Sequent Balance 8000 and the pragma INTERFACE for FORTRAN was not implemented in the Alsys compiler used on the Zenith Z-248. The results of these comparisons are shown in Tables 1 and 2. Table 1 shows the execution times and Table 2 shows the storage requirements for the various implementations. As can be seen from Table 1 the execution times are similar for all implementations on the same machine but vary greatly between machines. Notice that the MATH_TASKS benchmark took longer on all machines. This is due to task switching. Each task is called several hundred times in the MATH_TASKS benchmark. The tasks in the PRIME_TASKS benchmark are only called once. Therefore, it ran in approximately the same amount of time as the other implementations. The time penalty associated with task context switching for the MATH_TASKS benchmark should disappear if run on a parallel processor architecture with each task executing on its own processor. However, allocation of tasks to processors does not happen automatically. The MATH_TASKS benchmark also took longer to run on the Sequent Balance machine even though it has a parallel processor architecture. The benchmark executed as if it were running on a sequential processor.

Table 2 shows that the object code generated by the Ada compilers is considerably more than the object code generated by the FORTRAN compilers. Programs with Ada tasks require even more object code as one might expect. As can be seen from Table 2 different vendor's compilers generate significantly different amounts of object code. This finding is consistent with an Ada research report written by the University of Central Florida (7).

Table 1 Comparison of Execution Times

Machine/Model	Implementation FOR	F/A	A/F	Ada
DEC VAX 11/780				
PRIME_PROC	30.05	33.19	30.29	33.15
PRIME_TASKS	N/A	N/A	N/A	32.59
MATH_PROC	9.07	9.87	8.87	9.97
MATH_TASKS	N/A	N/A	N/A	22.93
SEQUENT BALANCE 8000				
PRIME_PROC	94.1	N/A	N/A	184.8
PRIME_TASKS	N/A	N/A	N/A	184.3
MATH_PROC	15.8	N/A	N/A	31.0
MATH_TASKS	N/A	N/A	N/A	41.9
ZENITH Z-248				
PRIME_PROC	546.46	N/A	N/A	802.96
PRIME_TASKS	N/A	N/A	N/A	696.29
MATH_PROC	63.27	N/A	N/A	92.10
MATH_TASKS	N/A	N/A	N/A	99.74

Times are in seconds
FOR: All FORTRAN Implementation
F/A: FORTRAN Implementation Calling an Ada Procedure
A/F: Ada Implementation Calling a FORTRAN Subroutine
Ada: All Ada Implementation
N/A: Not available

PRIME_PROC: Prime Number Program with Procedures
PRIME_TASKS: Prime Number Program with Tasks
MATH_PROC: Math Program with Procedures
MATH_TASKS: Math Program with Tasks

Table 2 Comparison of Storage Requirements
(Bytes)

Machine/Model	Implementation FOR	F/A	A/F	Ada
DEC VAX 11/780				
PRIME_PROC	2,764	5,592	7,570	7,464
PRIME_TASKS	N/A	N/A	N/A	8,308
MATH_PROC	2,864	3,652	4,772	4,928
MATH_TASKS	N/A	N/A	N/A	6,196
SEQUENT BALANCE 8000				
PRIME_PROC	24,576	N/A	N/A	97,292
PRIME_TASKS	N/A	N/A	N/A	120,220
MATH_PROC	14,336	N/A	N/A	18,524
MATH_TASKS	N/A	N/A	N/A	41,432
ZENITH Z-248				
PRIME_PROC	35,528	N/A	N/A	43,297
PRIME_TASKS	N/A	N/A	N/A	61,137
MATH_PROC	32,152	N/A	N/A	14,577
MATH_TASKS	N/A	N/A	N/A	33,821

FOR: All FORTRAN Implementation
F/A: FORTRAN Implementation Calling an Ada Procedure
A/F: Ada Implementation Calling a FORTRAN Subroutine
Ada: All Ada Implementation
N/A: Not available

PRIME_PROC: Prime Number Program with Procedures
PRIME_TASKS: Prime Number Program with Tasks
MATH_PROC: Math Program with Procedures
MATH_TASKS: Math Program with Tasks

Table 3 shows the results obtained from running the modified Dhrystone benchmark on all three machines. The modified Dhrystone consists of the original Dhrystone with some of the integer operations changed to floating point operations. In addition a FORTRAN version of the modified Dhrystone was prepared in order to facilitate comparison of FORTRAN and Ada code. The modified Dhrystone was originally available only in an Ada version included in a research report written by the University of Central Florida. The results obtained on the VAX 11/780 indicated that VAX Ada and VAX FORTRAN were almost identical in execution speed, with VAX Ada having a slight edge. The results obtained with the Zenith Z-248 would suggest that the Alsys Ada compiled code executes twice as fast as Microsoft's Fortran compiled code. This conclusion is at variance with the results obtained from executing the other benchmarks, which showed that the Ada programs took significantly longer to execute than the FORTRAN programs. A closer look at the figures shows even more discrepancies. For example the Ada modified Dhrystone benchmark ran at approximately three quarters the speed of the same program on the VAX 11/780, but the prime numbers program took 24 times as long to execute on the Zenith as on the VAX. Similarly, the FORTRAN version of the modified Dhrystone program took approximately one third the time on the Zenith as it did on the VAX, while the FORTRAN version of the prime numbers program took approximately 18 times as long on the Zenith as it did on the VAX. It should be noted that both the prime numbers and the math benchmarks are floating point intensive, while the modified Dhrystone does very few floating point operations. However, the prime numbers and the math benchmarks give results which can be checked for accuracy. In order to give accurate results, they must perform the same operations on all machines. The modified Dhrystone does not provide results other than timing data, and in order to verify that each step is being executed, some sort of debugging tool would be needed. No debugging tools were available for the Zenith to investigate the possibility that some modified Dhrystone steps were not being performed. Table 4 contains a list of execution times compared to the VAX 11/780 for all the benchmarks. The VAX has been assigned an execution time of 1 as a reference.

Table 3 Modified Dhrystone Execution Times.

Machine	Ada*	FORTRAN**
VAX 11/780	94,131	93,743
Sequent Balance 8000	22,740	26,061
Zenith Z-248	72,209	35,685

* Lines of Ada code executed per second.
** Lines of Fortran equivalent Ada Code executed per second.

Digital Equipment Corporation's Ada compiler version 1.1 and FORTRAN compiler version 4.6 were used. VERDIX Corporation's VADS® Ada compiler version 5.41 and DYNIX® FORTRAN compiler version 2.6 were used on the Sequent Balance running the DYNIX operating system. DYNIX is a version of UNIX®. The Alsys Ada compiler version 1.2, and the Microsoft Fortran compiler version 3.2 were used on the Zenith Z-248.

MACHINE	PNUM			MATH			MODDHRY	
	ADA		FORTRAN	ADA		FORTRAN	ADA	FORTRAN
	TASK	PROCS		TASK	PROCS			
VAX 11/780	1.00	1.00	1.00	1.00	1.00	1.00	1.00	100
SEQ. BAL 8000	5.66	5.57	3.13	1.83	3.11	1.74	4.14	3.59
ZENITH Z-248	21.37	24.22	18.18	4.35	9.24	6.98	1.30	2.63

PNUM:	PRIME NUMBER PROGRAM
MATH:	MATH PROGRAM
MODDHRY:	MODIFIED_DHRYSTONE PROGRAM
SEQUNT BAL 8000:	SEQUENT BALANCE 8000
ADA:	ADA IMPLEMENTATION
FORTRAN:	FORTRAN IMPLEMENTATION
TASK:	TASK IMPLEMENTATION IN ADA
PROC:	PROCEDURE IMPLEMENTATION IN ADA

Table 4 Execution Times Compared to the VAX 11/780

MACHINE ENVIRONMENTAL CONSIDERATIONS

The most important consideration is that the Ada compiler being used must implement the pragma INTERFACE. Equally important the FORTRAN compiler must allow subroutine calls to and from other languages. The linker or program that generates an executable module must be able to resolve address entries and the passing of parameters. Another nontrivial consideration is differences in arithmetic speed between Ada and FORTRAN for essentially the same precision. A comparison of floating point types is shown in Table 5 for DEC® Ada and FORTRAN. Table 6 shows the CPU time required to compute the number of prime numbers between 1 and 100,000 using the various floating point representations. The *PRIME_PROC* benchmark was used to compute the prime numbers on a VAX 11/780 with a floating point processor. LONG_FLOAT took more than 143 times as much time as REAL*8 even though they are both 64 bits and essentially the same machine representation. Apparently the FORTRAN compiler uses the hardware floating point processor and the Ada compiler does not. Hopefully this difference will be corrected in a later version of the Ada compiler. As can be seen from Table 6, very high precision arithmetic takes a long time in both languages.

Ada features strong data typing of objects. However, the Ada compiler cannot check the type of a variable in another language. Hence it is easy to get erroneous results due to a type mismatch. For example, a FORTRAN subroutine can return an integer result to an object of type Float in Ada.

OTHER CONSIDERATIONS

Usually the government buys trainers with 50 percent spare execution time and main memory. Sometimes the spare capacity is less than 50 percent. Figure 1 shows the relative cost per instruction versus spare time and memory capacity. It can be seen that cost goes up considerably for additional instructions that must be written once the 50 percent spare capacity is exceeded. Cost increases for several reasons. Probably the most significant reason is that code must be written more efficiently. It may even be necessary to rewrite some of the code that was not intended to be modified in order for an upgrade to fit. Therefore, when considering doing an upgrade in Ada, a thorough timing and sizing analysis is a must. A trade off analysis should be done to determine if it would be more cost effective to change hardware to stay below the 50 percent spare capacity point.

Table 5 Comparison of Arithmetic Types

Type	Language: Ada	Language: FORTRAN
FLOAT and REAL*4	32 bits Precision 6 decimal digits Range 0.29E-38 to 1.7E38	32 bits Precision 7 decimal digits Range 0.29E-38 to 1.7E38
LONG_FLOAT and REAL*8	64 bits Precision 15 decimal digits Range 0.6E-308 to 0.9E308	64 bits Precision 15 decimal digits Range 0.56G-308 to 0.9G308
LONG_LONG_FLOAT and REAL*16	128 bits Precision 33 decimal digits Range 0.84E-4932 to 0.59E4932	128 bits Precision 33 decimal digits Range 0.84Q-4932 to 0.59Q4932

Note: Ranges include both positive and negative numbers

Table 6 Execution Time Versus Arithmetic Types

FORTRAN		ADA	
REAL*4	00:00:26.80	FLOAT	00:00:26.60
REAL*8	00:00:39.81	LONG_FLOAT	01:35:54.54
REAL*16	01:46:43.49	LONG_LONG_FLOAT	01:54:27.35

All times are in Hours:Minutes:Seconds
Model: Prime Numbers with Procedures

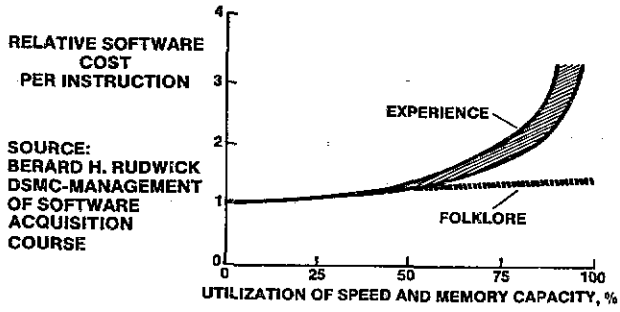

FIGURE 1. HARDWARE CONSTRAINTS VERSUS SOFTWARE COST

Benchmarks representative of the Ada code to be implemented should be run to obtain timing and sizing estimates. Differences in execution times are expected for different computers. However, this paper and others show that different Ada compilers generate significantly different amounts of object code for the same source code.

Another important consideration is that of the real time debugger to be used. Does it support the use of source code in two different languages? Of course, it is important that the debugger support the host/target environment to be used. However, this issue is independent of the high order language being used.

Life cycle cost should be given prime consideration when performing a major software upgrade. As more and more software is written in Ada, we can expect the cost of maintaining FORTRAN code to increase. Therefore, when considering a major software upgrade, the program manager should consider rewriting in Ada all of the code that is likely to change during the life cycle of the trainer.

FUTURE PLANS

In order to gain more experience integrating FORTRAN and Ada, the benchmarks will be run on several other machines. Two parallel processor machines as well as other microprocessors will be used.

Plans are currently underway to rewrite some of the Passive Acoustic Analysis Trainer's FORTRAN modules in Ada. This will allow the Naval Training Systems Center to gain practical experience integrating Ada into a FORTRAN based trainer.

SUMMARY

There appears to be no technical reason why Ada cannot be used for major upgrades of existing FORTRAN designs. Also it appears very feasible to reuse FORTRAN models and code in a new Ada design. However, some vendor's software products are easier to interface than others. All software products should improve in the future as it becomes clearer that interfacing Ada to FORTRAN and other languages is very desirable. Existing training device code should be reused when it is cost effective to do so.

The purpose of this paper is to point out issues that should be considered when planning to integrate FORTRAN and Ada. General assumptions should not be made based on the data presented. For example, it should not be assumed that Ada compilers produce twice as much object code as FORTRAN compilers. The issues described in this paper requiring consideration should be examined in the context of the planned implementation. Of all issues that should be considered, timing and sizing appear to be the most critical.

REFERENCES

1. Ada Language Reference Manual, ANSI/MIL-STD-1815A, 1983.
2. OPNAV INSTRUCTION 5200.28, 1986.
3. DOD Directive 3405.2, 1987.
4. Management of Software Acquisition Course, Defense Systems Management College, Ft. Belvoir, VA, 1987.
5. Cohen, Normal H., Ada as a Second Language. McGraw-Hill 1986.
6. VAX™ Ada Language Reference Manual, Digital Equipment Corporation, 1985.
7. Hughes, Charles E., Knowles, Henry and Lacy, Lee, "Ada Risk Assessment Report", University of Central Florida, 1986.
8. VERDIX Ada Development System Sequent/DYNIX Users Manual, VERDIX Cooperation, 1987.
9. Weicker, R., "Dhrystone: A Synthetic Systems Programming Benchmark", CACM 27,20, pp. 1013-1030, 1984.

About The Authors

William F. Parrish, Jr. is a Supervisory Electronics Engineer in the Surface/Submarine Warfare Software Branch at the Naval Training Systems Center and is currently involved in procuring trainers with Ada software. He has over 20 years of software development experience. Prior to his employment with the Navy, he was a Senior Staff Engineer with Sperry Rand Corporation. Mr. Parrish holds a BSE degree in Electrical Engineering and a MSE degree in Computer Engineering from the University of Alabama in Huntsville.

Kent G. Trewick is an Electronics Engineer in the Surface/Submarine Warfare Software Branch at the Naval Training Systems Center where he is the Software Engineer for several trainers that are being procured. One of his trainers is the HARPOON HSCICS-1A Operator Team Trainer whose software is being written by McDonnell Douglas Astronautics in Ada. Mr. Trewick holds a BSE degree in Electrical Engineering from North Carolina A&T State University.