THE EFFICIENCY OF FORTRAN IN SIMULATION COMPUTERS

FRANK A. SIGMUND
Simulator Systems Engineering Department
Goodyear Aerospace Corporation
Akron, Ohio

INTRODUCTION

The task of initially selecting and sizing computer systems for training simulators is becoming more difficult. Current Air Force and Navy simulator specifications state that the software shall be programmed in FORTRAN to the maximum degree technically feasible. Although the use of a higher level language such as FORTRAN is desirable in many respects for both simulator user and contractor, it entails an extensive new effort for the contractor during computer selection and sizing. Previously, software for simulators was programmed in assembly language. A thorough understanding of the adverse impact of FORTRAN on computer loading is necessary to reduce the risk involved with computer selection.

COMPUTER SELECTION CONSIDERATIONS

The various types of computers and the size of their associated operational programs for a number of Air Force simulators are shown in Table 1 which was extracted from Reference 1. Twelve different types of computers are used in the twenty-one simulators identified. From one to four computers are used per simulator. The size of these computer systems as indicated by words of core in the table varies from 30K to 194K (where K=1000). In summary, there is a wide variation in the type and size of computer systems used in simulators.

Selection of a computer system must include an assessment of performance as well as factors such as cost, reliability, maintainability, configuration control, data and availability of equipment. A discussion of computer performance follows.

To select a computer system for a simulator application, the processing requirements must be determined from the amount and fidelity of the simulation required. All data transfers must be analyzed to determine the proper type and speed for each interface. Simulator computers are typically compute limited, not input/output limited.

In the past when simulator software was programmed in assembly language, a computer loading estimate was developed using the instruction set of the candidate computer. This estimate determined the approximate number of computer memory words required and was subdivided into instructions and data. From the estimated number of instructions and instruction execution times, the time required to perform all computations and input/output was estimated. The computer system was then sized as far as memory and central processing units are concerned. For present simulator procurements, this method is no longer adequate because it does not consider the effect of FORTRAN on the computer loading estimate. This paper will discuss and measure several aspects of that effect.

For a real-time application, it is necessary to determine whether the computer system being considered is sufficiently fast to perform all computations and input/output and whether the resultant computer loading is within the real-time constraint of the system. For digital flight simulators, the real-time constraint is a stringent one where solution rates of up to 60 times per second may be required.

Resolution and accuracy requirements also are considered before a computer system is selected. These requirements are analyzed for both computations and input/output operations to determine if the word length of the considered computer is adequate.

Optional features of the considered computer also must be evaluated, such as additional hardware interrupts or other processor options. Throughout a computer system analysis, the cost and cost effectiveness of various components and features are considered.

TABLE 1. COMPUTATION SYSTEMS COMPENDIUM*

TRAINING DEVICE	COMPUTER	COMPUTERS/ SIMULATOR	LENGTH OF THE OPERATIONAL COMPUTER PROGRAM SYSTEM (WORDS OF CORE)
C-135B	MARK I	1	NOT AVAILABLE
C-141A	CDC 921	2	38K
C-141A	SEL 840A	1	34K
F-4E	GP4B (SINGER)	1	92K
C-5A	SEL 840A/840MP	2	63K
F-111A	GP4	1	92K
FB-111A (BOMB/NAV)	SIGMA 5	2	88K
FB-111A	SIGMA 5	3	180K
A-7D	DC 6024/1	1	40K
HH-53C	DC 6024/3	1	30K
CH-3	DC 6024/3	1	30K
F-111D	GP4B	2	194K
F-111F	GP4B	2	175K
F-15	DC 6024/4	2	103K
T-37	DC 6024/4	3/4 COCKPITS	42K
T-38	DC 6024/4	3/4 COCKPITS	49K
B-52 (MOD)	DC 6024/5	1	30K
SEWT (SIMULATOR FOR ELECTRONIC WARFARE)	SEL 86	1	50K
ASUPT (ADVANCED SIMULATOR FOR UNDER-GRADUATE PILOT TRAINING)	SEL 86	3	FLIGHT 83K VISUAL 32K (FORTRAN)
SAAC (SIMULATOR FOR AIR-TO-AIR COMBAT)	SIGMA 5	4	80-100K (FORTRAN)
UNTS (UNDERGRADUATE NAVIGATION TRAINING SIMULATOR)	HONEYWELL H 716	41/52 STATIONS	COMPLEX 51K 13 EA RADAR CONTROL 9K (1 TIME)

*THE DATA IN THIS TABLE WAS EXTRACTED FROM AIR FORCE MASTER PLAN SIMULATORS FOR AIRCREW TRAINING. FINAL REPORT, DECEMBER 1975

Since a real-time system often must be changed to meet additional requirements, system expansion capabilities are analyzed. The system should be capable of expanding in a modular manner with virtually no disturbance to current operations.

SIGNIFICANT COMPUTER PERFORMANCE CHARACTERISTICS

For simulators currently being specified, performance characteristics of greatest significance are:

Speed

FORTRAN efficiency

Multiprocessor capability

Increased addressing capability

Speed, in terms of IPS (instructions per second), has always been and continues to be of primary importance in simulator computer selection. FORTRAN efficiency, which is the subject of this paper, has become a critical item because of the requirement to use FORTRAN as the primary programming language. Multiprocessor capability and direct addressing of increasingly larger blocks of memory have become significant items because of increasing simulation requirements and computer spare time and memory requirements of up to 40%.

FORTRAN VS. ASSEMBLY LANGUAGE EFFICIENCY

A study was formulated in order to evaluate the various coding types that exist in a simulator and determine which types could be efficiently programmed in FORTRAN. Simulation programs written in Harris SLASH 4 assembly language from the F-15 Flight Simulator developed by Goodyear Aerospace were used as the input for this study.

A computer program was developed to read Harris assembly language source code, classify the code into various types and print the code type distribution data.

This data was examined to determine which coding types should be studied in detail. The criteria for selection included a significant amount of use in several programs and a type which could be isolated for study. The selected coding types were then evaluated for the effects of the FORTRAN compiler on the size and running time of the code. The areas selected for study were bit manipulations, logical operations, floating point arithmetic and basic instructions. These types of processing were studied by comparing abbreviated FORTRAN and assembly versions of F-15 simulator programs.

The Harris FORTRAN compiler had as an optional feature an optimizer which processed the FORTRAN output and produced more efficient code in several areas. Quantitative results of optimized FORTRAN vs assembly language efficiency for the various processing types are shown in Table 2. A discussion of each type of processing follows.

BIT OPERATIONS

Bit operations are defined as those which manipulate a single bit either via a register or directly in memory. It was theorized that these would provide severe difficulty for FORTRAN to handle efficiently because there is no direct reference to individual bits in FORTRAN. The HUD control (HUDCTL) program was selected for the test. Portions of HUDCTL were extracted and coded in FORTRAN and assembly language. These portions were considered to be representative of simulator bit processing. There were two FORTRAN versions generated. One version used a larger data matrix with one word representing each possible discrete signal.

Four runs were made through different paths of the program logic. This was done to ensure that several code variations were exercised for a larger data sample. The results in Table 2 were based on the average execution time for these four runs. It was concluded that, for the Harris SLASH 4, FORTRAN cannot efficiently handle bit manipulations. This computer is designed to efficiently manipulate single bits and is more efficient than some other computers in this respect. However, the FORTRAN compiler is not designed for bit manipulation and, therefore, does not take advantage of the machine capability. Nor does the optimizer significantly improve the results for this type of processing. The use of one word per signal instead of a packed format improved the program time and core efficiency but required a much larger core area for the data. In this example, it was assumed that the 20 data words were fully packed with 24 discretes

TABLE 2. OPTIMIZED FORTRAN VS ASSEMBLY LANGUAGE EFFICIENCY, HARRIS FORTRAN

TYPE OF PROCESSING	SIZE OF ASSEMBLY SAMPLE (WORDS)	% SIZE EXPANSION OVER ASSEMBLY LANGUAGE	% TIME EXPANSION OVER ASSEMBLY LANGUAGE
BIT (PACKED DATA)	97	186	185
BIT (UNPACKED DATA)	51	41	33
BASIC (TRANSFER, ADD, SUBTRACT, COMPARE, BRANCH)	178	11	10
FLOATING POINT	445	12	14
LOGICAL	106	5	7
AVERAGE OF ABOVE (EXCLUSIVE OF BIT, PACKED DATA)	-	17	16

in each. The word version, therefore, required 480 words for data instead of 20. This method, though much better than the packed format method, is still not efficient enough for a bit manipulating program. An alternative to the use of FORTRAN would be to code programs with a large amount of bit manipulation instructions in assembly language or to use in-line assembly language code for this type of processing. If this is not possible, and timing is a critical factor, the use of one signal per word is another possible alternative if sufficient core is available.

No FORTRAN compilers of the candidate computers for simulators have the capability to handle individual bits efficiently. Some of these computers also lack the capability to handle bits efficiently even in assembly language because of different machine architecture. For these computers, the difference between assembly language and FORTRAN may not be as great.

LOGICAL OPERATIONS

Logical operations are defined as the AND, OR and XOR functions. It was theorized that the Harris SLASH 4 FORTRAN could efficiently handle a program of this type which masks and manipulates several bits at one time. The Built-In-Test (BIT) program was selected as representative of significant use of this kind of code. Selected portions of BIT were coded both in FORTRAN and assembly language.

Six runs were made to exercise various program paths and ensure that coding was identical in both the assembly and FORTRAN versions. The results indicate that with optimization, FORTRAN is almost as efficient as assembly language. This indicates that programs with logical operations could be coded in FORTRAN for the Harris SLASH 4 computer. Some other computer systems, however, do not have the AND, OR, and XOR operators available for masking operations except as intrinsic functions. If these compilers are used, and generate function calls instead of simple instructions, the efficiency will be greatly reduced. For these systems, their standard FORTRAN may be adequate for logic functions. If these compilers generate direct code as efficiently as the Harris compiler instead of function calls, they would be adequate.

FLOATING POINT

Floating point arithmetic operations comprise a major portion of simulator programs and, thus, it is important that FORTRAN efficiently compile it. The Forces and Moments (FCMOM) program has many floating point equations and had previously been compiled by the Harris optimizing compiler. The generated code was studied for its efficiency. It was found to be very efficient, as expected. There is no problem in using FORTRAN for floating point equations. Since FCMOM is 90% floating point operations, it is a good benchmark for assessing FORTRAN floating point efficiency. The FCMOM program will be discussed in more detail in the benchmark section of this paper.

BASIC INSTRUCTIONS

The category of basic instructions includes integer arithmetic, transfers, compares, branches. A benchmark containing these instruction types was generated from the Tactics Data Save (TACSAV) program. It was anticipated the FORTRAN optimizing compiler would handle these types of instructions very efficiently. The benchmark results, presented in Table 2 indicate this is true. Programs composed primarily of these basic instructions could, therefore, be efficiently programmed in FORTRAN.

OTHER INSTRUCTION TYPES

Shift instructions were found to be a small percentage of F-15 simulator instructions and, therefore, considered to be insignificant. The Harris compiler has special instructions (SHIFT and ROTAT) which enable efficient shift operations. Other compilers rely on the optimizer to translate a power of 2 into a shift operation. This also provides efficient coding except in the case when end around shifting is desired. This operation, though not handled directly by many compilers, is not critical to simulator programs. Thus, there appears to be no problem with most shift instructions being handled by FORTRAN.

Byte instructions were also found to be a small percentage of F-15 simulator instructions. Since these instructions really are designed to handle the lower 8 bits of a word in the accumulator, it appears that there will be no significant change in efficiency in using FORTRAN for programs which contain these instructions.

Input/output and interrupt handling functions are very ill-suited to FORTRAN. It is, therefore, necessary that routines with these instructions be written in assembly language or in FORTRAN with some in-line assembly code in special areas. FORTRAN, unlike assembly language, does not provide a direct input/output capability. The calling of general-purpose I/O subroutines by FORTRAN in lieu of direct input/output is unacceptable because of their inherent inefficiency. There is no capability in FORTRAN to save registers as is required for an interrupt routine. Therefore, simulator programs which contain I/O functions or which are interrupt initiated cannot be coded in FORTRAN.

FORTRAN VS. ASSEMBLY LANGUAGE EFFICIENCY - CONCLUSIONS

The operations of programs were divided into coding types, and these types were evaluated for their use in FORTRAN. The following types were found to be inadequate for FORTRAN: single bit manipulation, input/output, and interrupt processing. The processing types which are amenable to FORTRAN are floating point arithmetic, logical functions, and basic instructions which include transfers, integer arithmetic, branches, compares, shifts and byte operations.

OPTIMIZATION CONSIDERATIONS

As mentioned earlier, the Harris FORTRAN compiler features an optimizer which processes the FORTRAN output and produces more efficient object code in several areas. This optimization is all local; that is, each statement is optimized independently of other statements. The result of this optimization is a reduction in module storage and execution time requirements. The actual reductions are dependent upon the amount of code in those areas that the optimizer operates on. According to Reference 2, for a typical module the number of memory words required to store the program exclusive of data words is reduced by 10 to 50%. The computation time exclusive of input/output time is reduced by 10 to 30%. This is corroborated by the actual results for the types of processing discussed above. These results in terms of percent expansion before and after optimization, are shown in Table 3. The average core savings exclusive of data for all five processing types in the Table was 43%. The average execution time savings was 29%. These savings are significant enough to demonstrate the importance of using an optimizing FORTRAN compiler in real-time training simulators.

Therefore, any computer selected for use in a training simulator which will be programmed in FORTRAN should have an optimizing compiler. The Harris optimizer is efficient enough to make the use of FORTRAN possible in simulators. The optimizer does not enable the efficient use of bit manipulation, I/O and interrupts. These instruction types would have to be coded in assembly language to maintain high program efficiency. Other computers which could be selected for simulation must have a FORTRAN optimizer at least as efficient as the Harris optimizer. Some of these systems have global optimizers, which optimize code across more than one statement at a time, thus providing increased efficiency.

A comparison of two optimizing compilers that are designed for execution on the same computer, the Interdata 8/32, is shown in Table 4. FORTRAN VI, which is a superset of the ANSI Standard (X3.9-1966), performs optimizations which include subscript evaluation by linearization, common index elimination, register allocation and transfer logic. FORTRAN VII which is being implemented according to one of the latest ANSI Standards (X3J3/56), performs more extensive optimizations including both machine independent and machine dependent optimizations. Table 4 shows results for the combined totals of the FCMOM and ELECT benchmark programs which are discussed in the following section of this paper. The FORTRAN VII compiler was not available for an actual benchmark run because the compiler was undergoing final acceptance test. However, projected results were obtained by hand-optimizing the code generated by the FORTRAN VI compiler according to the design specification of the FORTRAN VII compiler. From the table, it can be seen that a 32% savings in memory and a 23% savings in time are achieved with the compiler with more extensive optimization.

TABLE 3. BEFORE AND AFTER OPTIMIZATION, HARRIS FORTRAN

TYPE OF PROCESSING	% SIZE EXPANSION OVER ASSEMBLY LANGUAGE		% TIME EXPANSION OVER ASSEMBLY LANGUAGE	
	BEFORE	AFTER	BEFORE	AFTER
BIT (PACKED DATA)	208	186	189	185
BIT (UNPACKED DATA)	90	41	48	33
LOGICAL	31	5	24	7
FLOATING POINT	22	12	25	14
BASIC	93	11	62	10
	AVERAGE CORE SAVINGS (OPTIMIZED VS NON-OPTIMIZED) 43%		AVERAGE TIME SAVINGS (OPTIMIZED VS NON-OPTIMIZED) 29%	

TABLE 4. A COMPARISON OF TWO OPTIMIZING COMPILERS

COMBINED TOTALS OF FCMOM AND ELECT BENCHMARKS	INTERDATA 8/32	
	FORTRAN VI	FORTRAN VII*
MEMORY TOTAL (WORDS)	1277	867
% MEMORY SAVINGS	-	32
TIME TOTAL (MSEC)	1620	1250
% TIME SAVINGS	-	23

*PROJECTED RESULTS

BENCHMARK PROGRAMS

Two FORTRAN benchmark programs were used to measure the relative performance capabilities of potential simulation computers and their FORTRAN compilers. The benchmark programs were run on each computer. These programs are representative samples of the kinds of programs that are used in flight simulators. Two performance criteria were considered: (1) compiled program size and (2) program execution time. The characteristics of the tested programs on the detailed test results are presented below.

The two programs which were used are FORTRAN versions of assembly language programs selected from the F-15 flight simulator. One program, which is part of the aerodynamics simulation subsystem, evaluates total aircraft forces and moments. The other program simulates one of the aircraft systems, the electrical system.

The Forces and Moments Program (FCMOM) consists almost entirely of arithmetic-type expressions. The number of executable source statements in the FORTRAN program is 83 as compared with 372 in the F-15 assembly language program. Both of these numbers are program size only and as such do not include storage of external variables. The test case that was selected corresponded to an airborne F-15 in a typical flight configuration.

The Electrical Program (ELECT) consists almost entirely of logical expressions. There are 277 executable source statements in the FORTRAN program as compared with 324 in the F-15 assembly language program. Four test cases were selected which correspond to typical operational conditions of the electrical system. An assumption that was made prior to coding the ELECT program was that each discrete input and output resided in a full word. This would have to be done by special I/O hardware or by software which would unpack and pack the discrete inputs and outputs, respectively.

The benchmark programs were run on the following computers:

Harris SLASH 4
Harris SLASH 7
Interdata 8/32
SEL 32/55
MODCOMP IV/25
DEC KL-10
NORSK NORD 50

The results of the test runs on each computer are shown in Table 5.

The size of the FORTRAN Memory Totals in the table is a measure of: (1) compiler efficiency and (2) instruction set power. The first of these two factors is more predominant. The size of the FORTRAN Time Totals in the table provides a measure of computer hardware execution speeds as well as the factors mentioned above.

All of the FORTRAN memory and run time data shown in the tables represent actual benchmark runs except for those marked with an asterisk (*) to designate "projected results."

The SEL 32/55 FCMOM results were generated by substituting specified instruction execution times of the floating point hardware for the times of the firmware instructions actually executed in the benchmark run. The NORD 50 benchmark results were updated from the actual results by factoring in the effect of a pending compiler modification which enables direct, rather than indirect, addressing of FORTRAN common variables. And, as mentioned earlier, the Interdata 8/32 FORTRAN VII results were obtained by hand-optimizing the code generated by the existing FORTRAN VI compiler according to the design specification of the FORTRAN VII compiler.

DISCUSSION OF BENCHMARK RESULTS

The objective of this portion of the paper is not to compare architectures of the seven computers shown in Table 5. However, it is appropriate to classify the computers into three groups: large scale, "midicomputer," and minicomputer. According to Theis[3], a "midicomputer" is a high-performance machine with a mid-length word size and a less than midsize pricetag. Classifying the computers then, the DEC KL-10 with its 36-bit word size is definitely a large-scale computer, the MODCOMP IV/25 with its 16-bit word size is a minicomputer, and the remaining five computers with their 24- and 32-bit word sizes are midicomputers.

As might be expected, the DEC KL-10 provided the best memory/time efficiency using 1% less memory and 40% less execution time than the baseline Harris SLASH 4 assembly language benchmarks. However, associated with

TABLE 5. BENCHMARK RESULTS

	ASSEMBLY	FORTRAN						
COMBINED TOTALS OF FCMOM & ELECT BENCHMARKS	HARRIS SLASH 4	DEC KL-10	HARRIS SLASH 4	HARRIS SLASH 7	INTERDATA 8/32 FORTRAN VII *	SEL 32/55 *	NORSK NORD 50 *	MODCOMP IV/25
MEMORY TOTAL (WORDS)	790	786	940	948	867	816	964	997
% MEMORY CHANGE (FORTRAN VS HARRIS SLASH 4 ASSEMBLY LANGUAGE)		-1	19	20	10	3	22	26
TIME TOTAL (MSEC)	1511	907	1754	1448	1250	1147	1460	2898
% TIME CHANGE (FORTRAN VS HARRIS SLASH 4 ASSEMBLY LANGUAGE)		-40	16	-4	-17	-23	-3	92

*PROJECTED RESULTS

the high performance of this large scale computer is a large scale price tag. The memory efficiency of the five midicomputers ranged from 22% more memory than baseline for the NORD 50 to only 3% more memory than baseline for the SEL 32/55. The execution time efficiency of these computers ranged from 16% more time than baseline for the Harris SLASH 4 to 24% less time than baseline for the SEL 32/55. The MODCOMP IV/25 provided the least efficient results (26% more memory and 92% more time) mainly because it is basically a 16-bit machine with a limited 32-bit capability added on. This results in two 16-bit memory accesses for each 32-bit word and an accompanying increase in execution time.

All of the computers in the table made use of optimizing compilers in the benchmark runs.

SUMMARY

A summary of the results of the limited analysis of FORTRAN efficiency performed for this paper are presented below:

- An average memory penalty of 17% and time penalty of 16% was incurred for four types of processing (Table 2) in a Harris FORTRAN vs assembly language efficiency comparison. This was further confirmed by the combined benchmark totals for the Harris SLASH 4 (Table 5) where the memory penalty was 19% and the time penalty was 16%. According to Trainor and Burlakoff[4], these results are consistent with published results for other comparable HOLS that have indicated a 10% to 20% penalty.

- An average memory savings of 43% and time savings of 29% (Table 3) was achieved in an optimizing vs. non-optimizing FORTRAN compiler comparison.

- A memory savings of 32% and time savings of 23% was achieved in a comparison of two optimizing compilers.

- Benchmark results for seven candidate simulation computers were presented exhibiting a wide range of memory and execution time efficiencies.

CONCLUSIONS

FORTRAN efficiency is an important factor in evaluating the performance of computers for training simulators. From the results of the analysis of FORTRAN efficiency summarized above, the conclusions are as follows:

- Minimal execution time and memory penalties are incurred for most types of processing which are performed in training simulators.

- An optimizing compiler is essential.

- An optimizing compiler that performs extensive optimization is highly desirable.

- A comparison of benchmark results must be performed to assess relative efficiencies of candidate computers.

REFERENCES

1. Air Force Master Plan Simulators for Aircrew Training, Final Report, ASD/XR-TR75-25, Deputy for Development Planning, Aeronautical Systems Division, Wright-Patterson Air Force Base, December 1975.

2. DC 6024 FORTRAN Compiler General Specification, Datacraft Corporation, September, 1973.

3. Theis, D.J., "The Midicomputer," Datamation, February, 1977, Vol. 23, No. 2, pages 73-82.

4. Trainor, W.L. and Burlakoff, M., "JOVIAL - 73 Versus Assembly Language - An Efficiency Comparison," NAECON'77 Record.

5. Sigmund, F.A., Final Report, Simulator Software Development IR&D Project No. 0509-76D11, GER-16416, Goodyear Aerospace Corporation, February 3, 1977.

6. Babel, P.S. and Birns, M.L., "A System Oriented Benchmark For Training Simulators," Ninth NTEC/ Industry Conference Proceedings, November 9-11, 1976.

7. Allen, F.E., and Cocke, J., "A Catalogue of Optimizing Transformations," Design and Optimization of Compilers, New Jersey: Prentice-Hall, 1972.

ABOUT THE AUTHOR

MR. FRANK SIGMUND is Special Programs Group Leader in the Simulator Systems Engineering Department at Goodyear Aerospace in Akron, Ohio. He is responsible for proposal support, International Research and Development (IR & D) project execution, and support of other programming groups. He has been principal investigator on several IR & D projects, the most recent of which investigated the use of FORTRAN and structured programming in simulator software development. He has been lead software engineer on a number of Navy and Air Force flight training simulators.
Prior to his work at Goodyear Aerospace, Mr. Sigmund was with General Electric at its Computer Department in Phoenix and Cleveland. While there, he programmed an assembler for a multiprogramming disc system and provided technical support for district marketing. In Syracuse, New York, also with General Electric, he programmed the 412L air weapons control system. Mr. Sigmund served as a second lieutenant in the U.S. Army Transportation Corps at Ft. Eustis. He has a B.S. degree in physics from John Carroll University in Cleveland.