LA-8902-MS

C.3

VAX/VMS Benchmarking

University of California

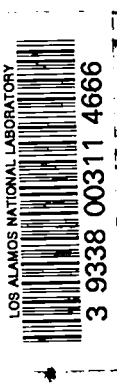
LOS ALAMOS SCIENTIFIC LABORATORY
Post Office Box 1663 Los Alamos, New Mexico 87545

UNITED STATES
DEPARTMENT OF ENERGY
CONTRACT W-7405-ENG. 36

LA-8902-MS
UC-32
Issued: July 1981

VAX/VMS Benchmarking

Larry Creel



VAX/VMS BENCHMARKING

by

Larry Creel

ABSTRACT

Primary emphasis in this report is on the performance of three Digital Equipment Corporation VAX-11/780 computers at the Los Alamos National Laboratory. Programs used in the study are part of the Laboratory's set of benchmark programs. The VAX-11/780 computers each had slightly different configurations that affected the performance of several of the benchmarks. Execution times of these programs on the VAX-11/780s are also compared to those on the Control Data Corporation (CDC) 6600 and Cyber 73 computers.

INTRODUCTION

This report gives the results of benchmark studies made on three Digital Equipment Corporation (DEC) VAX 11/780 computers at the Los Alamos National Laboratory. The benchmark programs used in the study are part of the Laboratory benchmark set, and are characteristic of the type of applications run on the Laboratory computers [1]. They are described in Appendix A. Execution times of these programs on the VAX computers are compared to those on the Control Data Corporation (CDC) 6600 and Cyber 73 computers. Histograms comparing these performances are given in Appendix B.

VAX 11/780 COMPUTERS USED IN THE STUDY

We refer to the three VAX computers used in the benchmark study as VAX1, VAX2, and VAX3. VAX1 and VAX3 have floating point accelerators (FPAs), which are optional features that can provide substantial performance increase. VAX1 uses Version 1.6 of the VAX/VMS (Virtual Address Extension/Virtual Memory System) operating system; VAX2 and VAX3 use Version 2.0.

Paging Subsystem

Although the VAX 11/780s are not large computers, their virtual memory system enables them to run very large codes. It does this through a paging subsystem that requires all the executing code to reside in virtual memory (main memory and auxiliary) while only a subset of the code's pages resides in physical memory (a page is 512 8-bit bytes). This subset is called the working set. Working-set sizes vary with the operating system and are controlled by the system manager. VAX1 had a maximum size of 2000 pages; VAX2 and VAX3 each had a maximum size of 256 pages.

The paging subsystem is very complex (see Appendix C) and can radically affect performance. For example, the results of the program that computes megaflop rates suggested that the VAX1 should be somewhat faster than a CDC Cyber 73. Execution times of most of the benchmark codes corroborated this assumption; however, LABMK14 ran five times longer on VAX1 than it did on a Cyber 73. The problem was that LABMK14 had incurred over 5 million page faults. Further investigation revealed that LABMK4 and LABMK8 were also experiencing heavy page-fault activity.

Page Faulting

Page faulting occurs on VAX/VMS when a page is referenced that is not in the working set. All three of these codes performed vector operations, including matrix transposes, which meant that several pages were being referenced frequently. Because the size of the working set was only 150 pages, it could not hold all the pages that were referenced. The solution in this instance was to increase the size of the working set. Table I gives the amount of page faults incurred by the three programs before and after a working set size adjustment on VAX1. Increasing the working set size resulted in a remarkable increase in performance. Table II gives the number of page faults incurred by each VAX for each program in the job mix. Working-set sizes for each system were set as high as possible to minimize faulting.

A special subroutine was added to each benchmark code that would determine from the system the number of page faults incurred as well as the elapsed CPU time (Fig. 1). Page faults come in two varieties: in-core and disk. Times for disk page faulting varied greatly, ranging from 1230 to 8470 μs on VAX3. The mean time was 3156 $\pm$744 μs. Because of the uncertainty of a disk page-fault time, no effort was made to obtain statistics for it on VAX1 or VAX2. The mean time required for an in-core page fault on VAX1 is 385 $\pm$3 μs. VAX2 has a mean of 356 $\pm$3 μs; VAX3 has a mean of 367 $\pm$2 μs. Over a million page faults were generated on each VAX configuration to come up with these values.

TABLE I

EFFECTS OF INCREASING WORKING-SET SIZE ON PROGRAMS WITH HEAVY PAGING ACTIVITY ON VAX1 (SINGLE PRECISION)

Program Name	Working-Set Size	Number of Page Faults	CPU Time (s)
LABMK4	150	82,282	522.1
	300	534	476.7
LABMK8	150	634,342	1021.5
	300	69	771.9
LABMK14	150	5,327,711	2600.0
	375	9	446.0

TABLE II

NUMBER OF PAGE FAULTS INCURRED BY BENCHMARK PROGRAMS

Program Name	Single Precision			Double Precision		
	Number of Page Faults per System			Number of Page Faults per System		
	VAX1	VAX2	VAX3	VAX1	VAX2	VAX3
LABMK1	1,354	3,662,067	3,160	1,381	3,046	203,978
LABMK4	534	82,254	82,253	1,054	21,150	165,004
LABMK5	25	82	73	55	25	3
LABMK6	0	1	1	0	1	1
LABMK8	69	539	303	83	1,653,076	1,653,011
LABMK11	30,800	77,200	75,200	94,400	57,200	57,200
LABMK14	9	89,741	97,929	7	853,591	843,906
LABMK15	0	60,600	36,300	0	300	300
LABMK18	43	228	238	83	1,088,593	1,088,852
LABMK21	3	5	5	0	3	3
LABMK22	0	0	0	0	0	0
NFP100	0	0	0	0	0	0

Maximum working set size for VAX1 is 2000 pages.
Maximum working set size for VAX2 is 256 pages.
Maximum working set size for VAX3 is 256 pages.

```
      SUBROUTINE TIMER(CPUTIM,IFAULT)
C
C     THIS ROUTINE RETURNS THE CPU TIME IN THE VARIABLE CPUTIM AND
C     THE NUMBER OF PAGE FAULTS IN IFAULT.  THE RESOLUTION OF CPUTIM
C     IS NO BETTER THAN 10 MILLISECONDS.
C
C       INTENDED USE IS THE FOLLOWING.....
C
C         CALL TIMER (T1,I1)
C    ....DO OPERATIONS....
C         CALL TIMER(T2,I2)
C
C     THEN THE ELAPSED CUP TIME EQUALS (T2-T1)   AND THE NUMBER OF
C      ELAPSED PAGE FAULTS IS (I2-I1).
C
C

    REAL*4 CPUTIM,PTIM
    INTEGER*4 ITIM,BUFADR,IFAULT,BUFPFL
    INTEGER*2 ITMLST(14)
    EQUIVALENCE (BUFADR,ITMLST(3)),(BUFPFL,ITMLST(9))
    PARAMETER JPI$_CPUTIM = '00000407'X,JPI$_PAGEFLTS = '0000040A'X
    DATA ITMLST/4,JPI$_CPUTIM,4*0,4,JPI$_PAGEFLTS,6*0/
    BUFADR=%LOC(ITIM)
    BUFPFL=%LOC(IFAULT)
    CALL SYS$GETJPI(,,,ITMLST,,,)
    CPUTIM=FLOAT(ITIM)/100.0
    RETURN
    END
```

Fig. 1. Subroutine to detect page faults and measure elapsed CPU time.

RUNNING THE BENCHMARK PROGRAMS

The Laboratory benchmark programs are coded in ANSI-standard FORTRAN, which required very few changes to enable them to run on a VAX system. After the benchmark codes were converted and running on one system (VAX1), they were written to tape and transferred to the other two systems. Writing and reading tapes from one VAX to another was found to be extremely easy.

On each system, the timing routine (the utility that gets the elapsed CPU time from the operating system) was checked to ensure that it was running correctly. A timing verification program and a stopwatch were used for this validation [2].

On the CDC Cyber 73 and 6600 computers, the benchmark programs were run during dedicated system time (DST); on the VAXs, they were run during periods of inactivity. Because the benchmark codes are for

the most part compute bound, elapsed wall-clock time should be very close to the elapsed CPU time if the system is inactive. Wall-clock times were within 1% of the VAX CPU times.

Three of the programs in the mix (LABMK11, LABMK15, LABMK22) were not actually run as long as their reported times imply. LABMK11 contains a loop that is executed 400 times. All of the program calculations are contained in that loop. If left unmodified, the program would execute for an excessive amount of time. It was modified, therefore, to execute the loop only 1 time, and its subsequent execution time was multiplied by 400. LABMK15 and LABMK22 are similar in this respect except the loop executed 300 times. If these programs were run without scaling, their actual run time would be less than the scaled run times for the following reasons:

1. A certain amount of start-up time before the actual execution of the loop will have been erroneously multiplied by the scale factor.

2. The number of page faults incurred by one of these programs could very easily be the same in both the scaled and unscaled versions; once a page is faulted into the working set, it could very easily stay there.

Clearly, small discrepancies may be introduced into the results by performing this scaling technique; however, we feel it was the best solution to enable execution of programs LABMK11, LABMK15, and LABMK22.

Table III gives the execution times of the benchmark programs on each of the systems.

Effect of Using a Floating-Point Accelerator

To show the effects of the floating-point accelerator on execution times, we removed it from VAX3. Table IV shows that without the floating-point accelerator, single-precision jobs ran 1.5 times as long as before, and double-precision jobs ran more than twice as long.

VAX3 without a floating-point accelerator is very similar to VAX2. The only difference between them is the amount of main memory each system has. No significant change in execution times was observed, and none was expected because the load on the system was very light. Although this fact does not allow any statement to be made regarding the effect of increasing the main memory, it does underline the fact that the reason the VAX2 does not perform as well as VAX1 and VAX3 is its lack of a floating-point accelerator, rather than some deficiency in the size of main memory.

TABLE III

ELAPSED WALL-CLOCK AND CPU RUN TIMES FOR LOS ALAMOS BENCHMARK CODES
(Time in s)

Program Name	Cyber 73 (Opt = 1)[a]		Cyber 73 (Opt = 2)[a]		CDC 6600 (Opt = 1)		CDC 6600 (Opt = 2)		VAX1 (SP)	
	WC Time	CPU Time	WC Time	CPU Time	WC Time	CPU Time	WC Time	CPU Time	WC Time	CPU Time
LABMK1	776.0	747.7	478.0	457.0	290.0	287.3	204.0	201.8	1044.1	1041.7
LABMK4	904.0	867.7	783.0	752.2	268.0	264.1	233.0	230.3	476.8	476.7
LABMK5	771.0	740.9	761.0	731.4	251.0	248.6	251.0	247.5	511.3	509.0
LABMK6	2400.0	2326.2	1800.0	1656.6	600.0	583.2	600.0	495.9	1718.3	1683.2
LABMK8	1330.0	1268.5	1190.0	1131.7	350.0	340.9	240.0	238.0	772.2	772.0
LABMK11	6800.0	6640.0	6400.0	6156.0	2400.0	2156.0	2000.0	1940.0	5064.0	5044.0
LABMK14	479.0	460.0	431.0	414.0	135.0	134.0	121.0	119.5	446.0	446.0
LABMK15	2100.0	1854.6	1800.0	1633.5	600.0	547.2	600.0	481.5	1576.0	1515.0
LABMK18	810.0	777.1	750.0	714.8	190.0	188.9	140.0	134.8	453.8	452.2
LABMK21	N/A	111.9	N/A	95.0	N/A	42.7	N/A	37.7	73.2	73.2
LABMK22	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1026.0	939.0

Program Name	VAX1 (DP)		VAX2 (SP)		VAX2 (DP)		VAX3 (SP)		VAX3 (DP)	
	WC Time	CPU Time	WC Time	CPU Time	WC Time	CPU Time	WC Time	CPU Time	WC Time	CPU Time
LABMK1	1038.0	1037.8	1332.8	1332.4	1332.2	1321.8	1036.4	1036.2	1026.0	1026.0
LABMK4	731.6	724.4	825.1	779.6	2378.1	1949.1	584.4	559.9	909.9	861.6
LABMK5	810.0	810.0	793.7	793.2	2692.3	2691.7	494.6	492.8	803.9	803.8
LABMK6	2364.0	2319.3	3181.7	3133.4	6205.7	6149.9	1671.9	1638.3	2347.2	2305.6
LABMK8	1305.5	1304.9	1223.9	1223.6	3674.8	3674.1	772.9	772.7	1862.3	1862.0
LABMK11	7915.6	7888.0	6608.0	6586.0	17172.0	17128.0	4809.2	4788.0	7692.0	7672.0
LABMK14	637.2	637.0	896.0	896.0	1794.5	1794.0	497.2	497.0	937.7	935.0
LABMK15	2219.0	2142.0	3252.0	3075.0	6273.0	6006.0	1557.0	1491.1	2172.0	2093.9
LABMK18	802.3	798.4	766.5	766.1	2427.9	2426.3	452.0	451.9	12116.4	1211.2
LABMK21	106.1	106.0	105.4	105.2	257.3	257.1	73.7	73.6	104.8	104.8
LABMK22	1731.9	1650.0	1656.3	1578.2	4575.0	4484.7	1014.0	951.0	1626.0	1551.3

[a] OPT = 1 and OPT = 2 refer to the degree of optimization performed by the Fortran compiler; valid only for CDC 6600 and CYBER 73.

CPU = Central Processing Time
DP = Double Precision
SP = Single Precision
OPT = Optimization
WC = Wall Clock

TABLE IV

EXECUTION TIMES WITH AND WITHOUT A
FLOATING POINT ACCELERATOR ON THE VAX3
(Time in CPU s)

	Single Precision			Double Precision		
Program Name	Runtime w/o FPA T1	Runtime with FPA T2	Ratio T1/T2[a]	Runtime w/o FPA T3	Runtime with FPA T4	Ratio T3/T4[b]
LABMK1	1,328.10	1,036.20	1.28	1,332.70	1,026.00	1.30
LABMK4	786.70	559.90	1.41	1,925.00	861.60	2.23
LABMK5	792.80	492.80	1.61	2,699.00	803.80	3.36
LABMK6	3,130.60	1,638.30	1.91	6,140.10	2,305.60	2.66
LABMK8	1,222.80	772.70	1.58	3,808.60	1,862.00	2.05
LABMK11	6,600.00	4,788.00	1.38	1,7164.00	7,672.00	2.24
LABMK14	905.00	497.00	1.82	1,820.00	935.00	1.95
LABMK15	3,035.70	1,491.10	2.04	5.984.70	2,093.90	2.86
LABMK18	765.78	451.90	1.69	2,434.80	1,211.20	2.01
LABMK21	105.40	73.60	1.43	257.10	104.80	2.45
LABMK22	1,545.10	951.00	1.62	4,481.70	1,551.30	2.89

[a] T1/T2 is the ratio of elapsed CPU time (single precision) without an FPA to the ratio of elapsed CPU time with an FPA.

[b] T3/T4 is the ratio of elapsed CPU time (double precision) without an FPA to the ratio of elapsed CPU time with an FPA.

Integer vs Real-Number Arithmetic

The VAXs did not do as well on integer arithmetic as they did on real-number arithmetic. From Table III it is seen that the execution time of a benchmark program was usually better on a VAX with a floating-point accelerator (VAX1, VAX3) than it was on a Cyber 73. However, on VAX1 and VAX3, the integer arithmetic code LABMK1 ran twice as long as the Cyber's best time and 1.3 times as long as its worst time. There was no significant difference between single- and double-precision run times for LABMK1 because the Fortran statement declaring double precision applied only to real variables.

The differences in execution rates between VAX1 and VAX3 are the results of page faulting. Page faulting is almost nonexistent on VAX1. The poor performance of VAX2 in comparison to the other machines is due to its lack of floating-point accelerator and its upper bound of 256 pages for working-set size. The fact that VAX2 has less main memory than VAX1 or VAX3 is of no consequence because there was no load on the system other than the running benchmark code itself. In fact, the available 1.5 megabytes were not fully used.

Double-Precision Execution Times

The double-precision execution times for the benchmark codes were considerably longer than the single-precision rates. Table V gives a breakdown for each VAX system. Again, VAX2 suffers from the lack of a floating-point accelerator. On a VAX system with a floating-point accelerator, the run time of a code with double precision is at least half again as long as with single precision. For example, a VAX with a floating-point accelerator can perform about 290 000 floating-point operations per second in single precision; 160 000 in double precision. For a VAX without a floating-point accelerator, the numbers are roughly 160 000 and 60 000 floating-point operations for single and double precision, respectively. Ratios for VAX3 are higher than VAX1 because VAX3 incurred more page faults in double precision than VAX1.

Measured Megaflop Rates

The megaflop rates given in Table VI are realistic rates and are not intended to represent some maximum or minimum rates. The operations contained in the megaflop program are extracted from real application codes. On the basis of the rates in this table, a code on a VAX (with a floating-point accelerator) should run at about the same speed in double precision as the same code on a Cyber 73 in single precision with an optimization level of 1.

TABLE V

COMPARISON OF DOUBLE-PRECISION EXECUTION TIMES
TO SINGLE-PRECISION EXECUTION TIMES

Program Name	VAX1 T1/T2[a]	VAX2 T1/T2	VAX3 T1/T2
LABMK4	1.52	2.50	1.54
LABMK5	1.59	3.39	1.63
LABMK6	1.38	1.96	1.41
LABMK8	1.69	3.00	2.41
LABMK11	1.56	2.60	1.60
LABMK14	1.43	2.00	1.88
LABMK15	1.41	1.95	1.40
LABMK18	1.77	3.17	2.68
LABMK21	1.45	2.44	1.42
LABMK22	1.76	2.84	1.63
Mean	1.56±0.0021	2.59±0.0266	1.76±0.0195

[a] T1/T2 is the ratio of elapsed CPU time in double precision (T1) to the elapsed CPU time in single precision (T2).

TABLE VI

MEGAFLOP RATES AS GIVEN BY
LOS ALAMOS MEGAFLOP PROGRAM NFP100

Computer Name	Optimization Level	Precision Type	Megaflop[a] Rate/s
CYBER 73	1	Single	0.17
CYBER 73	2	Single	0.18
CDC 6600	1	Single	0.60
CDC 6600	2	Single	0.74
VAX1	N/A	Single	0.29
VAX1	N/A	Double	0.16
VAX2	N/A	Single	0.16
VAX2	N/A	Double	0.06
VAX3	N/A	Single	0.29
VAX3	N/A	Double	0.16

[a]Millions of floating-point operations.

A CDC 6600 (also with an optimization level of 1) should run twice as fast as the same VAX in single precision. Plots in Appendix B show that this approximation is a good one in most cases. Notable exceptions are LABMK1 and LABMK14.

SUMMARY

The VAX 11/780 with the VAX/VMS operating system lends itself well to scientific applications. The execution rates of the benchmark codes on the VAX system with a floating-point accelerator were usually faster than those of a CDC Cyber 73 but not as fast as a CDC 6600. Performance of the VAX/VMS system is greatly affected by (1) the presence of a floating-point accelerator, (2) whether the code is single or double precision, and (3) the amount of page faulting incurred.

REFERENCES

1. Ann H. Hayes and Ingrid Y. Bucher, "LASL Computer Benchmark Performance 1979," Los Alamos National Laboratory report LA-8689-MS (February 1981).

2. Larry Creel, "Verification of Timing Routines," to be published in Proc. Computer Performance Evaluation Users Group, Orlando, Florida (October 1980).

APPENDIX A

PROGRAM DESCRIPTIONS

Program No. 1 - Monte Carlo.
- -347 source lines.
- -No I/O.
- -Code is compute bound and uses integer arithmetic. Computers using 24-bit integer arithmetic will produce incorrect answers.
- -Output consists of the self-contained "input" values and several statistical computed values, plus execute time.
- -Field length required to execute: 104000B.

Program No. 4 - Fast Fourier Transform code.
- -230 source lines.
- -Not vectorizable.
- -No I/O.
- -Output consists of the solution array for a subset of the steps computed, plus execution time.
- -Field length required to execute: 215000B.

Program No. 5 - Equation-of-state kernel.
- -720 source lines.
- -Not significantly vectorizable.
- -No I/O.
- -Code has built-in test for error checking and accuracy similar to tests in Program 2. Execute time is a linear function of an internal variable.
- -Output consists of setup data, relative errors of selected output variables, and total execution time.
- -Field length required to execute: 50000B.

Program No. 6 - Linear system solver - solves systems of the order of 100.
- -320 source lines.
- -Not vectorizable.
- -No I/O.
- -Output consists of the first three elements of the solution vector, the Central Processor Unit (CPU) time in each subroutine for one case, and the total time for execution.
- -Field length required to execute: 34000B.

Program No. 8 - Vector calculations.
- -189 source lines.
- -Code calls five separate routines to perform a variety of vector operations. Vector lengths range from 20 to 5000.
- -Three of the routines are vectorizable, two are not.
- -No I/O.
- -Output consists of total CPU time in each subroutine for each vector length, the average time/element for each routine, and the total time to execute the code.
- -Field length required to execute: 105000B.

Program No. 11 - Particle pusher kernel widely used in particle-in-cell calculations.

- -740 source lines.
- -Depending on compiler used, code is vectorizable. XFC, both vectorized and nonvectorized, performed significantly better than CFT on this code.
- -No I/O.
- -Output consists of time/particle in each subroutine called.
- -Field length required to execute: 70000B.

Program No. 14 - Matrix calculations including multiplication and transpose.

- -343 source lines.
- -Code goes through eight cases of 100 x 100 matrix, with variations of multiplication.
- -Code uses LINPACK routines and is highly vectorizable.
- -No I/O.
- -Output consists of the matrix diagonal and the total execution time for all eight cases.
- -Field length required to execute: 111000B.

Program No. 15 - Linear system solver for systems of the order 100.

- -399 source lines.
- -Vectorizable on all compilers.
- -No I/O.
- -Output consists of the first three elements of the solution vector, the CPU time in each subroutine, and total execution time.
- -Field length required to execute: 34000B.

Program No. 18 - Vector calculations (a variation of Program 8).

- -147 source lines.
- -Vectorizable--consists of the three routines found in Program 8, which vectorize well while eliminating the remaining two, which do not vectorize. Separate cases use vector lengths ranging from 20 to 5000.
- -No I/O.
- -Output consists of the total CPU time in each subroutine for each vector length, the average time/element for each routine, and the total time to execute the code.
- -Field length required to execute: 105000B.

Program No. 21 - Integer Monte Carlo.

- -370 source lines.
- -Not vectorizable.
- -Moderate I/O activity.
- -Output consists of selected table values, total execution time, and times for individual phases of the program.
- -Field length required to execute: 30000B.

Program No. 22 - Linear system solver for systems of the order 100 (a variation of Program 15).

- -423 source lines.
- -Vectorizable on all compilers.
- -No I/O.
- -Code has newer algorithms for matrix solving than Program 15, otherwise is identical to Program 15.

-Output consists of the first three elements of the solution vector, the CPU time in each subroutine and total execution time.
-Field length required to execute: 34000B.

Program MFLOPS - set of benchmark kernels to measure number of floating point operations/second.
570 source lines.
-Vectorizable on all compilers.
-Little I/O.
-Output consists of the megaflop rate for each kernel, plus average megaflop rate overall.
-Field length required to execute: 21000B.

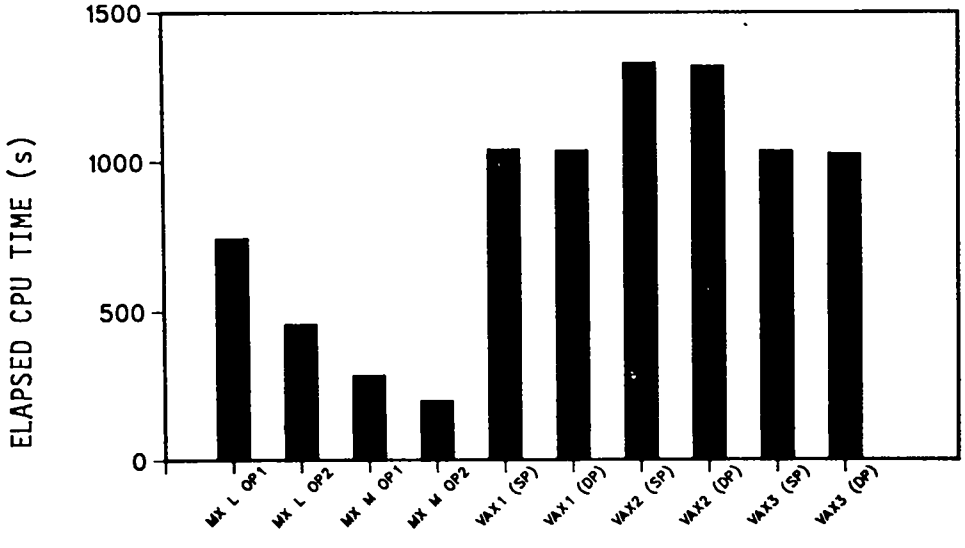
APPENDIX B

HISTOGRAMS COMPARING COMPUTER PERFORMANCES

The histograms in this appendix show the results of running the standard Los Alamos benchmark programs on a CDC Cyber 73, a CDC 6600, and the three VAX 11/780 computers used in the benchmark study.

INTEGER ARITHMETIC MONTE CARLO CODE

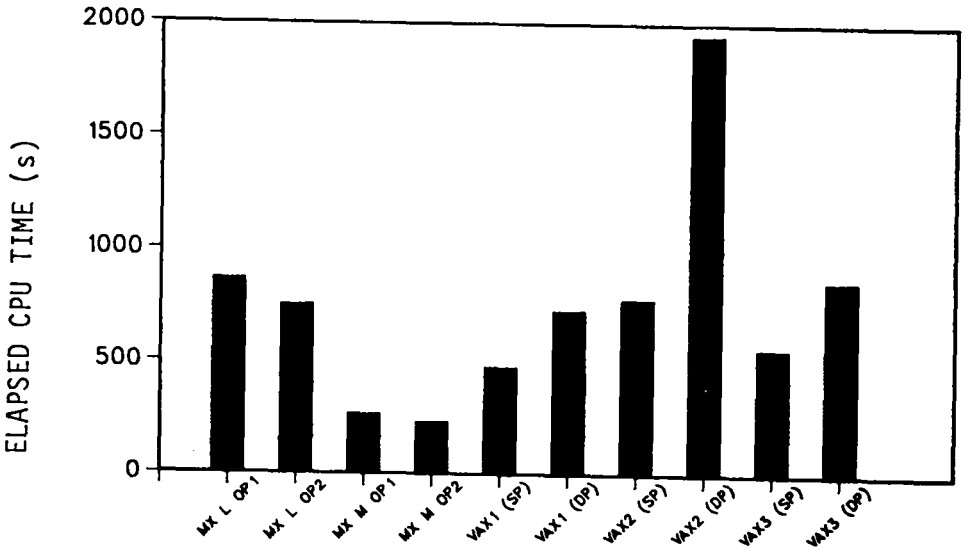
PROGRAM LABMK1



MX L = Los Alamos CDC Cyber 73
MX M = Los Alamos CDC 6600
OP1 = optimization level 1
OP2 = optimization level 2
SP = single precision
DP = double precision

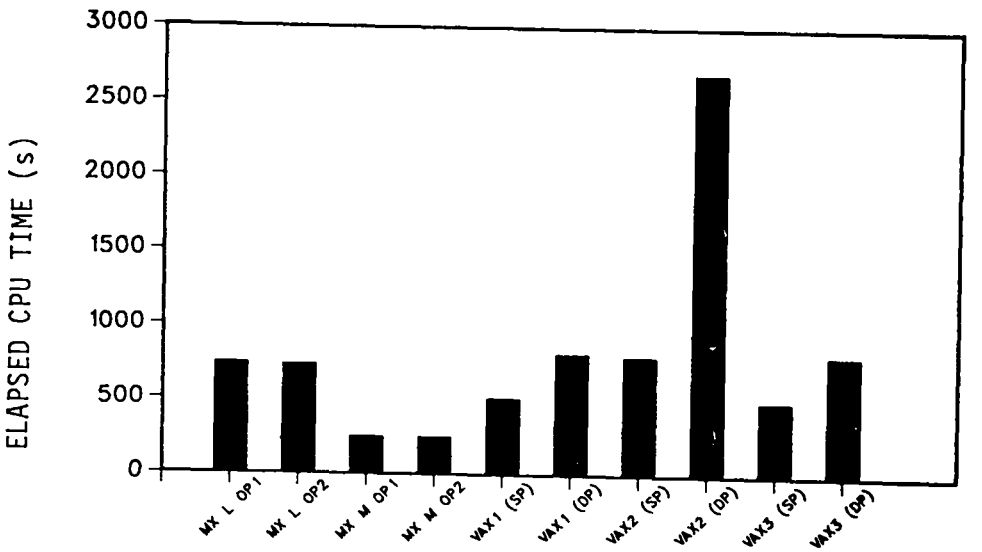
FAST FOURIER TRANSFORM CODE

PROGRAM LABMK4

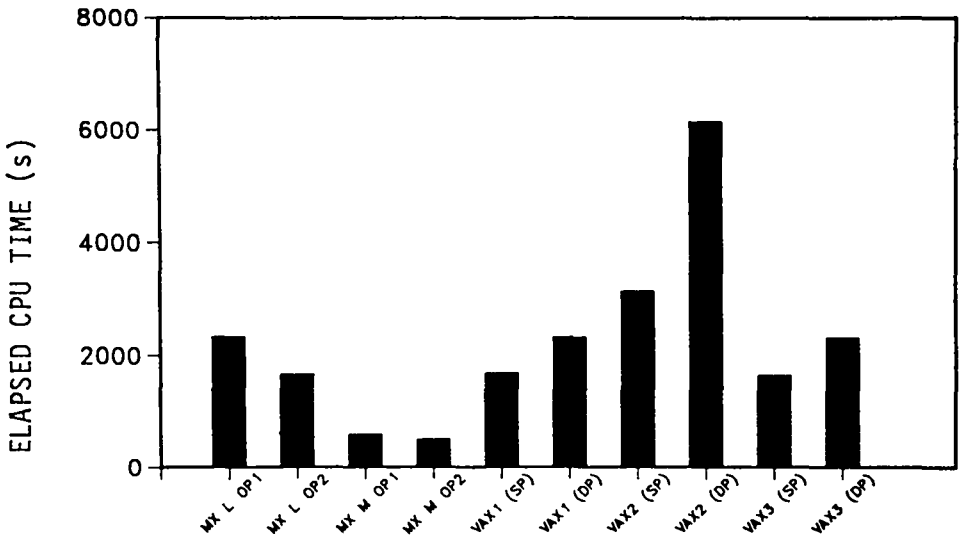


EQUATION OF STATE KERNEL

PROGRAM LABMK5

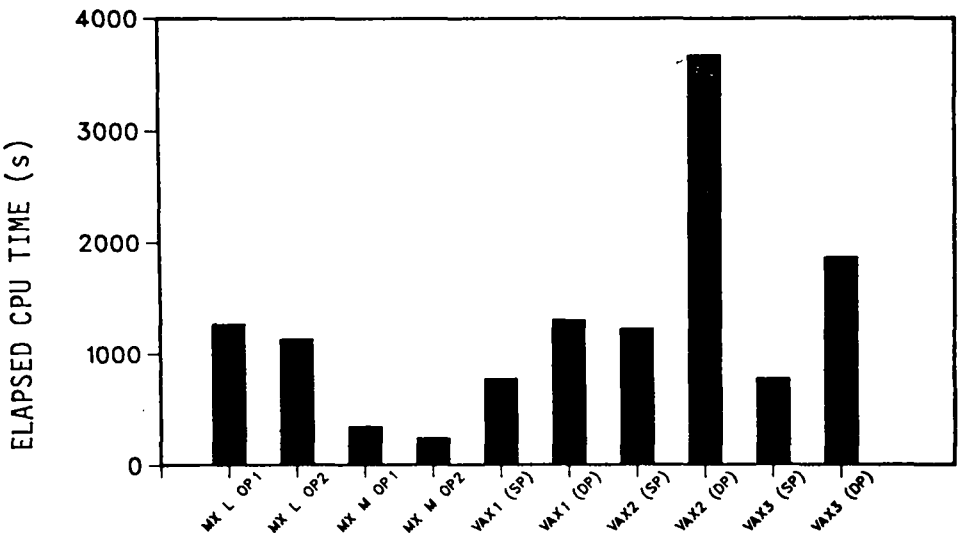


LINEAR SYSTEMS SOLVER
SOLVES SYSTEMS OF THE ORDER 100
PROGRAM LABMK6



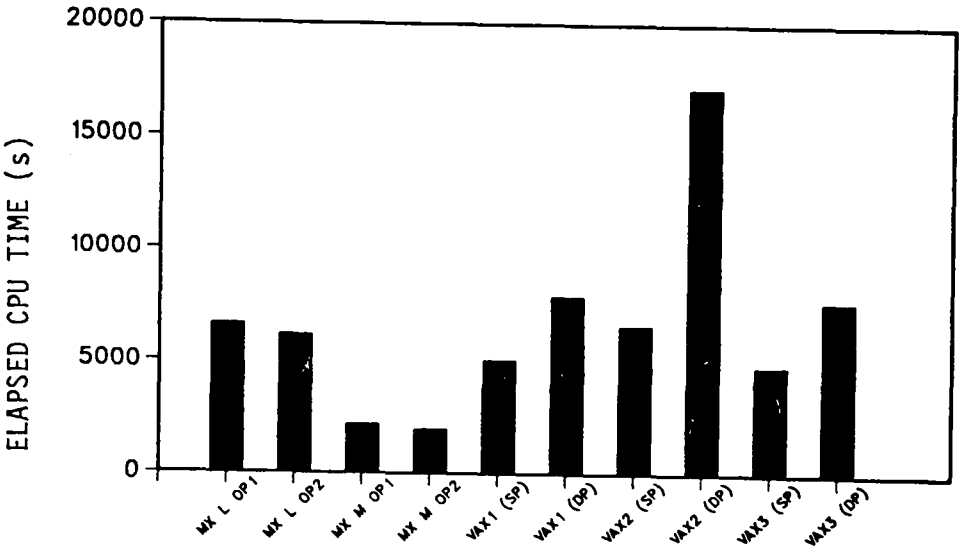
VARIETY OF VECTOR OPERATIONS

PROGRAM LABMK8



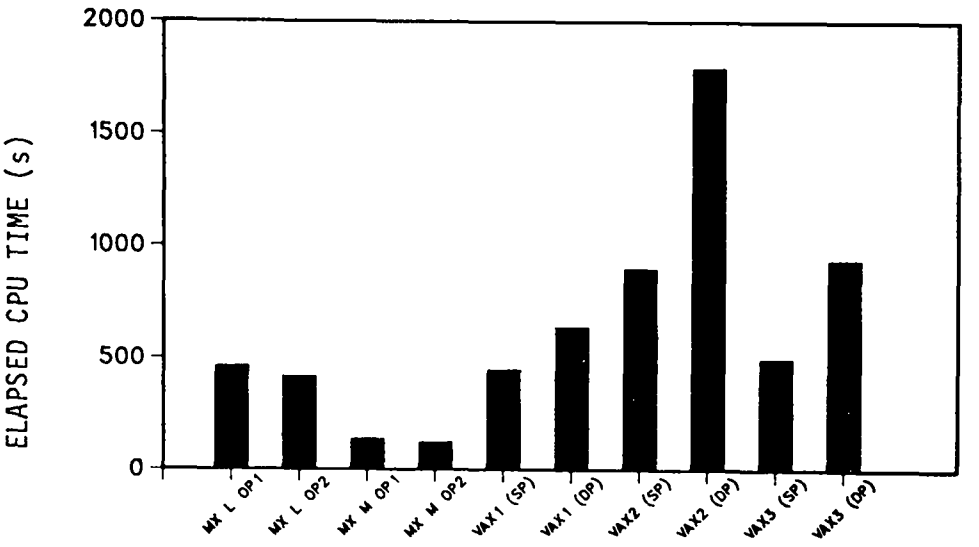
PARTICLE PUSHER CODE

PROGRAM LABMK11

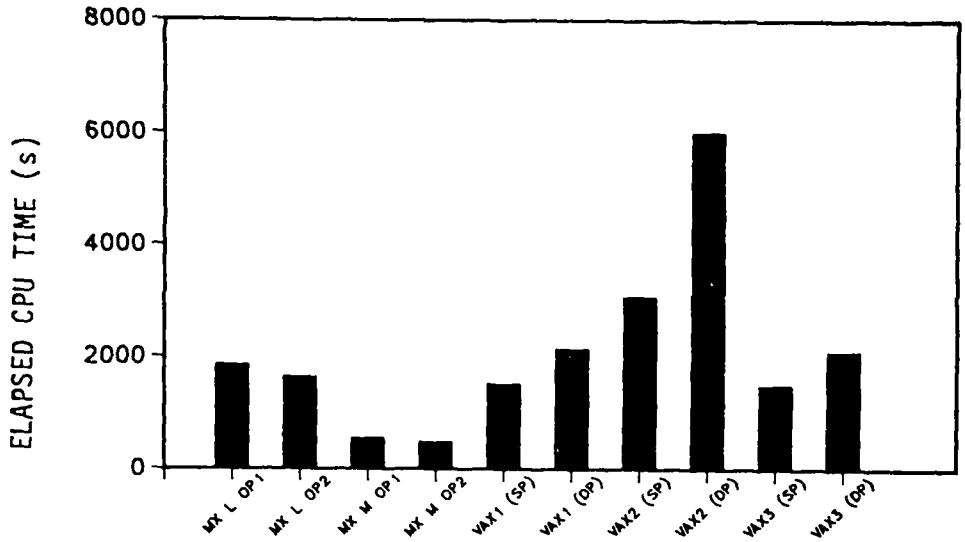


VARIETY OF VECTOR OPERATIONS

PROGRAM LABMK14

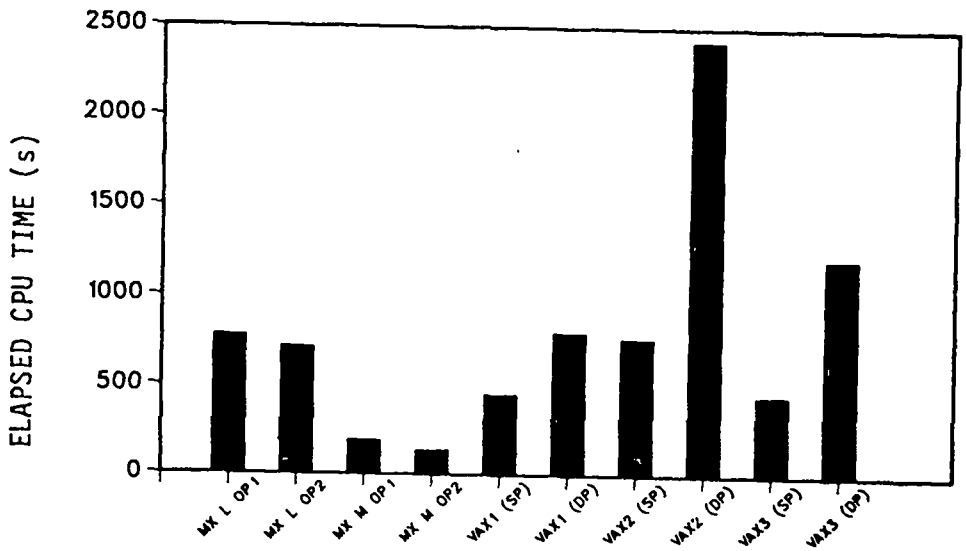


LINEAR SYSTEM SOLVER FOR MATRICES
OF THE ORDER 100X100 TO 800X800
PROGRAM LABMK15

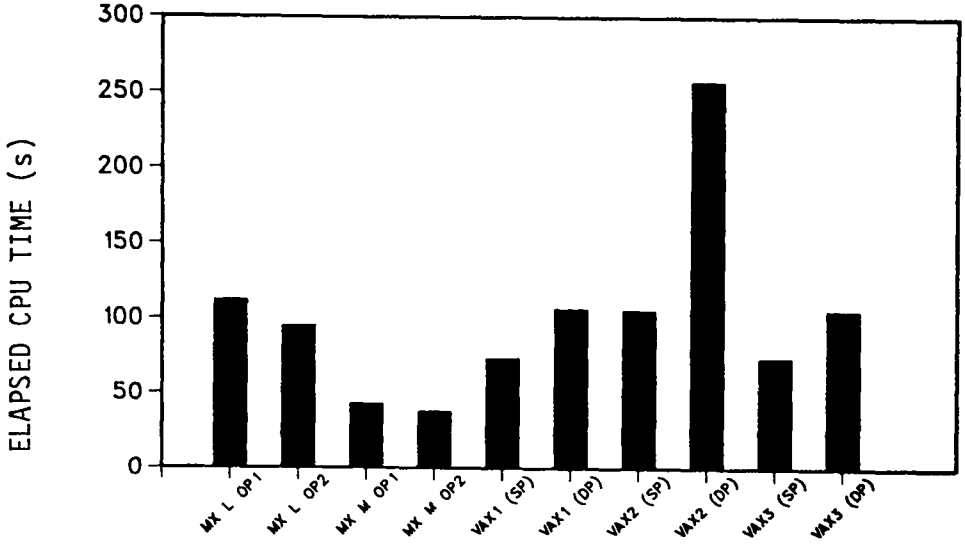


VARIETY OF VECTOR OPERATIONS

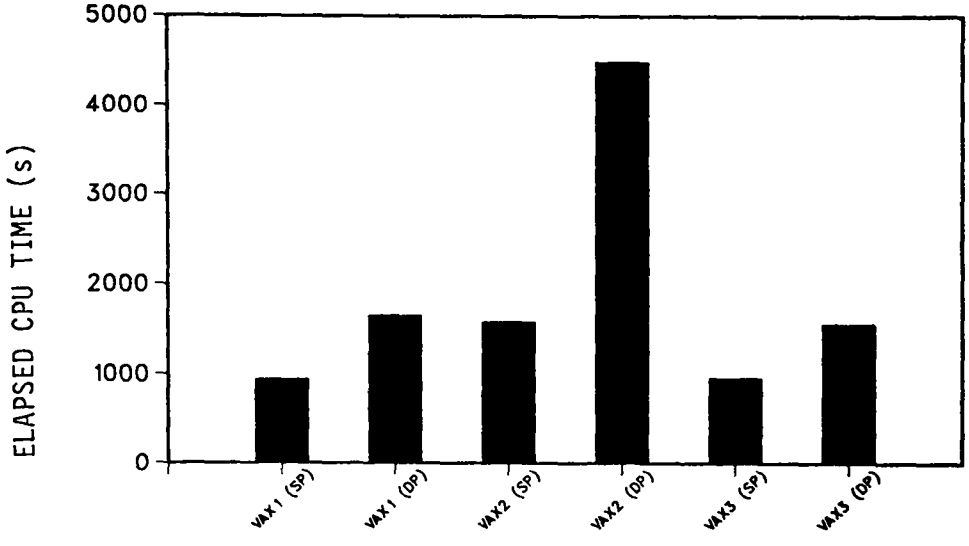
PROGRAM LABMK18



SCALAR MONTE CARLO CODE TO TRANSPORT
0.001- TO 20.0-MEV GAMMA RAYS IN A CARBON CYLINDER
PROGRAM LABMK21



LINEAR SYSTEMS SOLVER FOR SYSTEMS OF ORDER 100
VARIATION OF LABMK15
PROGRAM LABMK22



APPENDIX C

VAX/VMS PAGING SYSTEM

Because the VAX/VMS paging system can have such a pronounced effect on performance, a closer look is in order. The "VMS" in VAX/VMS is for "Virtual Memory System." This feature allows programmers to think they have almost unlimited memory and can write their programs accordingly, relying on the system to straighten things out. The system does this by requiring the executing code to reside wholly in virtual memory while only a subset of that code's pages (a page is 128 8-bit bytes of information) need be in main memory. This subset is called the "working set." Two other areas in main memory interact intimately with the working set. They are the "free list" and the "modified list."

The free list is made up of pages that have been discarded by working sets. It is really a pool of unowned pages. A page enters at the bottom and goes out the top. If a page is referenced before it reaches the top, it returns to the working set that wanted it. If the page reaches the top, it "goes away." There is always a copy of an unmodified page in what is known as the "image file" on auxiliary storage.

The modified list is different. The pages it contains are unique. They can never just go away. When a modified page leaves the working set, it goes to the modified list. The page enters at the bottom and leaves at the top. Pages leave the modified list when the maximum number of pages the list can contain has been reached or when the free list falls below a minimum. When one of these things happens, a number of pages (down to some minimum number remaining) are removed from the modified list. The pages would like to go to the free list, but that cannot happen until they have been written out to auxiliary storage (an area called a "paging file"). So these pages go both to disk and to the free list.

If a page in the modified list is referenced before it reaches the top, it simply leaves the modified list and goes to the appropriate working set that called it. Remember that the whole code is in an image file on disk. When this process begins, some of these pages are initially faulted in. They go into a working set. As more and more pages are brought in, the working set is soon full. Now, what happens when additional pages come in and there is no room? The pages with the greatest time since last access leave. They go into the free list if they have not been modified or into the modified list if they have.

VAX/VMS capitalizes on program locality, a characteristic of a program that indicates how close or far apart the references to locations in virtual memory are over time. A program with a high degree of locality does not reference many widely scattered virtual addresses in a short period of time. For example, VAX/VMS brings

in more pages than those referenced in anticipation of their use, because it is much faster to get 10 consecutive pages from disk than to go out to the disk 10 times for a single page. The size of this cluster is a system parameter. As another example, suppose a page is referenced that is not contained in the working set? The system first checks the free list and the modified list. If the page is there, the system transfers that page and one more to the working set, again anticipating possible use. Program locality appears to be a sound concept and is used successfully by the VAX/VMS paging system.

Printed in the United States of America
Available from
National Technical Information Service
US Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

Microfiche $3.50 (A01)

Page Range	Domestic Price	NTIS Price Code	Page Range	Domestic Price	NTIS Price Code	Page Range	Domestic Price	NTIS Price Code	Page Range	Domestic Price	NTIS Price Code
001-025	$ 5.00	A02	151-175	$11.00	A08	301-325	$17.00	A14	451-475	$23.00	A20
026-050	6.00	A03	176-200	12.00	A09	326-350	18.00	A15	476-500	24.00	A21
051-075	7.00	A04	201-225	13.00	A10	351-375	19.00	A16	501-525	25.00	A22
076-100	8.00	A05	226-250	14.00	A11	376-400	20.00	A17	526-550	26.00	A23
101-125	9.00	A06	251-275	15.00	A12	401-425	21.00	A18	551-575	27.00	A24
126-150	10.00	A07	276-300	16.00	A13	426-450	22.00	A19	576-600	28.00	A25
									601-up	†	A99

†Add $1.00 for each additional 25-page increment or portion thereof from 601 pages up.