NASA Technical Memorandum 88956

A Comparison of Five Benchmarks

Janice E. Huss and James A. Pennline
Lewis Research Center
Cleveland, Ohio

February 1987

NASA

A COMPARISON OF FIVE BENCHMARKS

Janice E. Huss*
Utica College
Utica, New York

and

James A. Pennline
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

SUMMARY

Five benchmark programs were obtained and run on the NASA Lewis CRAY X-MP/24. A comparison was made between the programs codes and between the methods for calculating performance figures. Several multitasking jobs were run to gain experience in how parallel performance is measured.

E-3411

INTRODUCTION

During the past 5 yr, there has been an increased interest in benchmarking supercomputer performance. New benchmarks have been written while older benchmarks have been put in modern perspective. Even the National Bureau of Standards has begun collecting Parallel Computer Benchmark Programs as part of the effort of its Computer Measurement Research Facility (CMRF) project. This collection, maintained by the Institute for Computer Services and Technology at NBS, is open to supercomputer users so that they may borrow from it and contribute to it.

Reports and articles written on a particular benchmark usually indicate performance in MFLOPS among various machines with specific operating systems and compilers. Some list quite a comprehensive range of machines including machines of the same type (such as a CRAY X-MP/22) but with different operating systems and compilers.

Performance figures for the same benchmark program run on the same machine at two or more locations can vary (due to running the program under different operating conditions.) An example is given in (ref. 3). We thought it would be interesting to collect a set of benchmarks and run them on the same machine (our X-MP/24) to gain some appreciation for and understanding of why a machine's performance figures can vary (sometimes greatly) depending on the benchmark program.

---

*Summer Faculty Fellow.

This report summarizes the results of our effort to:

- Collect a set of different benchmark programs and run them on our CRAY X-MP/24 to gain experience in how performance data is collected and how it can vary between benchmarks and between runs of the same benchmark.
- Set up a means for running these programs. Then when changes are made to the operating system or hardware, they can be run to see what the effects are on the performance data.
- Perform some initial experiments with multitasking to determine what kind of performance measurement to look for.

This benchmark collection contains in part specific routines which are used in scientific/engineering computing and in part segments of code which are a generic mix of calculations and instructions typical of scientific/ engineering computing. However, it does not represent a model of the specific workload at NASA Lewis.

DESCRIPTION OF THE BENCHMARKS

Five benchmark programs were obtained.

The NAS Kernel Benchmark Program (ref. 1) Authors: Dave Bailey and John Barton

The Argonne Programs (ref. 5) Author: Jack Dongarra

The Sandia Benchmark (SPEED) (ref. 2) Authors: T.H. Jefferson and M.R. Scott

The Whetstone Benchmark (ref. 7) Authors: H.J. Curnow and B. A. Wichmann

The Livermore Loops Author: F.H. McMahon

We include here a brief description of each one.

The NAS Kernel Benchmark

This is one of the more recently written benchmarks. It was developed for use of the NAS (National Aerodynamics Simulation) Projects Office at NASA Ames Research Center. It consists of approximately 1000 lines of FORTRAN code organized into seven tests, which are referred to as kernels. The calculations performed typify the type of supercomputing done at Ames. Since it is a more recent benchmark the seven kernels emphasize the vector performance of a computer system. The seven kernels are:

(1) MXM - performs matrix product on two input matrices employing a four-way unrolled outer product algorithm
(2) CFFT2D - performs a complex radix 2 Fast Fourier Transform on a two-dimensional input array
(3) CHOLSKY - performs a Cholesky decomposition on a set of input matrices

(4) BTRIX - performs a block tridiagonal matrix solution along one dimension of a four-dimensional array
(5) GMTRY - sets up an array for a vortex method solution and performs Gaussian elimination on the resulting array
(6) EMIT - creates new vortices according to certain boundary conditions
(7) VPENTA - simultaneously inverts three matrix pentadiagonals in a manner conducive to vector processing

For a more detailed discussion, see references 1 and 4.

The Argonne Programs

LINPACK is a library of FORTRAN linear algebra subroutines co-authored by Jack Dongarra in 1979. Over the past several years he has been publishing results of a benchmark program which solves systems of linear equations of order 100 using routines from the LINPACK collection. The latest results are given in Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment (ref. 5).

We obtained from Argonne National Laboratory a tape consisting of nine files each of which is a self contained benchmark. The following three were selected to be included in our benchmark study.

(1) A LINPACK system solver - A program and subroutines to measure timing of the LINPACK routines for solving a dense system of equations.
(2) A better LU decomposition - A program consisting of a different implementation of the solution of linear equations (ref. 9) using an algorithm based on matrix-vector operations rather than just vector operations. As reported in reference 5, it better reflects the true performance of a supercomputer than the LINPACK routines.
(3) A Vector Loop program - A program that indicates how well a compiler vectorizes some standard loops. No timing results are included.

Two of the other files were double precision versions of (1) and (2) above.
Two others (one single precision and one double precision) contained only definitions and declarations that allowed you to insert your own LU decomposition. The remaining one was a program to study indirect addressing in single precision.

The Sandia Benchmark (SPEED)

This is a program written at Sandia National Laboratory which consists of five kernels taken from programs in use at Sandia in 1978. The five kernels are:

(1) A linear equation solver with pivoting
(2) A routine which consists of part of the predictor step of an ordinary differential equation solver
(3) A routine consisting of a forward and backward substitution excerpt from a linear equation solver with pivoting
(4) A routine which consists of an excerpt from a Vortex Dynamics Code
(5) A routine which consists of an excerpt from a lattice relaxation code

Although the first three were taken from software library routines, the last two were taken from large user codes at Sandia. Thus this benchmark typifies the workload there.

The Whetstone Benchmark

This is a synthetic benchmark developed in the early 70's by Curnow and Wichmann at U.K.'s National Physical Laboratory in Whetstone, England. It's somewhat unique with respect to the above three in that it was developed to match instruction frequency statistics of language usage (originally ALGOL) collected from programs run at that laboratory. The resulting program is said to represent the execution of one million Whetstone instructions. The inverse of the measured run time indicates millions of Whetstone instructions per second. Our copy is a FORTRAN version which was obtained from the Computer Measurement Research Facility at the National Bureau of Standards.

The Livermore Loops

This program was developed at the Lawrence Livermore National Laboratory, Livermore, Ca. by F.H. McMahon, and had its initial beginnings in the late 60's and early 70's. Work was sponsored by the DOE. The version we obtained consists of 24 kernels (or loops) each consisting of a relatively small extract from a CPU - limited scientific application program. These computational structures are considered to be the most important CPU time components from the applications. They are:

Kernel	Description
1	Hydro Fragment
2	Incomplete Cholesky Conjugate Gradient Excerpt
3	Inner Product
4	Banded Linear Equations
5	Tri-diagonal Elimination - Below Diagonal
6	General Linear Recurrence Relation
7	Equation of State Fragment
8	ADI Integration
9	Integer Predictors
10	Difference Predictors
11	First Sum
12	First Difference
13	2-Dimensional Particle in Cell
14	1-Dimensional Particle in Cell
15	Casual FORTRAN. Development Version
16	Monte Carlo Search Loop
17	Implicit, Conditional Computation
18	2-Dimensional Explicit Hydro Fragment
19	General Linear recurrence Equations
20	Discrete Ordinate Transport
21	Matrix * Matrix product
22	Planckian distribution
23	2-Dimensional Implicit Hydro Fragment
24	Location of 1st minimum in array

Observations

The NAS kernels, the Argonne programs, the Sandia benchmark, and the Livermore Loops produce timing information in MFLOPS (millions of floating point operations per second) by dividing the number of floating point operations by the CPU time. However, the methods for counting the number of floating point operations differ.

In the NAS Kernel Benchmark, the number of floating point operations for each kernel is computed as follows. Each function operation (+, -, /, SQRT, SIN, etc.) has a precise number of floating point operations (weight) associated with it. For example, the additions of two real counts as one floating point operation while the division of real by a real counts as three. A complete table showing the number of floating point operations for the various functions is given in reference 1. The total number of floating point operations for each kernel is obtained by summing the products of the number of occurances of each function operation and its weight.

Only the first two Argonne programs listed above give a MFLOP rate. In the LINPACK system solver, the number of operations in computed using a well-known formula which approximates the number of additions and multiplications for solving a system of n equations in n unknowns. The formula is a function of the order of the matrix. The third program chosen gives no timing information since it only tests vectorization capability.

The Sandia Benchmark defines a floating point operation as an add, subtract, multiply, or divide, with each operation counting equally.

In the Livermore Loops program, floating point operations are counted according to the following weights:

+, -, *	1
/, SQRT	4
EXP, SIN, etc.	8
IF (X rel. Y)	1

The sum of the products of the number of occurances of each of these operations and its weight gives the FLOPS. This weight association is different from the one used in the NAS benchmark.

The Livermore Loops produces the most comprehensive set of statistics. Also, unique to this benchmark is a harmonic mean among the kernel rates. It can be argued that the harmonic mean (or more generally the weighted harmonic mean) is more meaningful than the arithmetic mean (refs. 3 and 8).

BENCHMARK RESULTS

The five benchmark programs provided the following results on our CRAY X-MP/24 running under COS 1.14 BF4. Each program was executed in dedicated time, meaning that this was the only job executing in the system at the time. All other jobs, including diagnostics, were suspended. These were runs made without any changes to the program codes we received (Level 0 tests as defined in (ref. 1)).

THE NAS KERNEL BENCHMARK

Program	MFLOPS
MXM	136
CFFT2D	51
CHOLSKY	53
BTRIX	80
GMTRY	70
EMIT	82
VPENTA	41
Total	65

The total MFLOPS represents the ratio of the sum of the floating point operations (FP OPS) for each kernel to the sum of the times for each kernel.

THE ARGONNE PROGRAMS

Program 1. LINPACK System Solver

MFLOPS	
22	(system of order 100)

The routines SGEFA, SGFSL from LINPACK perform standard LU decomposition with partial pivoting and back substitution. This is a FORTRAN version with simple statements and simple loops.

Program 2. A Better LU Decomposition

Array dimensions 301

Order 50

Unrolled Depth	MFLOPS
1	17
2	22
4	25
8	27
16	26

Order 200

Unrolled Depth	MFLOPS
1	53
2	70
4	81
8	94
16	96

Order 100

Unrolled Depth	MFLOPS
1	32
2	42
4	50
8	57
16	57

Order 250

Unrolled Depth	MFLOPS
1	61
2	80
4	91
8	106
16	108

Order 150

Unrolled Depth	MFLOPS
1	44
2	58
4	68
8	78
16	79

Order 300

Unrolled Depth	MFLOPS
1	68
2	88
4	99
8	115
16	117

The algorithm for this LU decomposition, whose description can be found in reference 9, is based on matrix-vector operations rather than just vector operations. Notice that timings are given for matrices of six different orders ranging from 50 to 300.

Program 3. Vector Loop Program.

The following table gives annotations of the types of loops (17 in all) and whether or not they vectorized.

	Loop	Vectorized
1	Statements in wrong order	n
2	Dependency needing a temporary	y
3	Loop with unnecessary scalar store	y
4	Loop with ambiguous scalar temporary	n
5	Loop with subscript that may seem ambiguous	y
6	Recursive loop that really isn't	y
7	Loop with possible ambiguity because of scalar store	n
8	Loop that is partially recursive	n
9	Loop with unnecessary array store	y
10	Loop with independent conditional	n
11	Loop with noninteger addressing	y
12	Simple loop with dependent conditional	y
13	Complex loop with dependent conditional	y
14	Loop with singularity handling	y
15	Loop with simple gather/scatter subscripting	n
16	Loop with multiple dimension recursion	n
17	Loop with multiple dimension ambiguous subscripts	y

THE SANDIA BENCHMARK (SPEED)

Kernel	MFLOPS
1	23
2	11
3	39
4	10
5	8
Total	13

In reference 2, a modified version of this program is reported to give a total of 36.8 MFLOPS, and an unmodified version is reported to show a total of 10 MFLOPS.

THE WHETSTONE BENCHMARK

The total CPU time for executing the program found by subtracting two calls to the function SECOND was

Time (T)	(1/T)
.04030 secs.	25 MWIPS

MWIPS = millions of Whetstone instructions per second.

THE LIVERMORE LOOPS

Mean Vector Length = 468

Kernel	MFLOPS	Span
1	152	1001
2	26	101
3	137	1001
4	44	1001
5	6	1001
6	13	64
7	171	995
8	113	100
9	145	101
10	65	101
11	8	1001
12	71	1000
13	4	64
14	11	1001
15	5	101
16	3	75
17	9	101
18	112	100
19	7	101
20	12	1000
21	29	25
22	66	101
23	13	100
24	2	1001

MFLOPS Range = 2 to 171
Harmonic Mean = 10
Median Rate = 26
Median Dev. = 40
Average Rate = 51
Standard Dev. = 55

The interesting and probably more meaningful indicator of overall performance is the harmonic mean computed with equal weights attached, i.e.,

$$\frac{1}{\frac{1}{24}\sum_{i=1}^{24}\frac{1}{R_i}}$$

where R_i = the MFLOPS of Kernel i, i=1, ..., 24. Each kernel above was assigned a weight of 1. As explained in references 3 and 8, the harmonic mean is a more meaningful representation of the actual performance and measure of the workload. This can be particularly true if there is a significant difference between the best and worst rate among all kernels or measured performances in a benchmark.

In addition, two other sets of MFLOP rates are output in the same form as above, one with a mean vector length of 89 and one with a mean vector length of 18. In each case, most of the spans or vector lengths were reduced. In case of the MVL of 89, the spans of 1001, for example, were dropped to 101, while in case of the MVL of 18, spans of 1001 were dropped to 27. A weight of 2 was attached to each kernel in the MVL = 89 case, while a weight of 1 was attached to each kernel in the MVL = 18 case. In general, for each particular kernel, the MFLOP rate decreased if the span decreased. However, loops were repeated enough times to keep the order of magnitude of the FLOPS about the same. In some cases the MFLOP decrease was significant. For example, kernel 1 rates were

Span	MFLOPS
1001	152
101	114
27	64

Overall harmonic means did not vary much.

MVL	MFLOPS (Harmonic Mean)
468	10
89	9
18	6.5

MULTITASKING RESULTS

This section reports on the results of two experiments. Experiment 1 involved running two copies of the same program, one on each of the two processors of our XMP/24 simultaneously. Experiment 2 involved actually multitasking one of the benchmarks.

EXPERIMENT 1

Bailey and Barton (ref. 1) suggested that it would be possible to get an idea of the amount of interprocessor resource contention which would occur when a particular program was multitasked by executing that program simultaneously on each of the individual processors. We did this for two of the benchmarks, the NAS Kernel Benchmark and the Vector Loop Program from Argonne. No explicit multitasking was done. The procedure was simply to suspend all jobs in the system and execute two identical, independent copies of the benchmark simultaneously on two (2) processors. The results of these simultaneous runs are given below, along with a single dedicated run for comparison.

NAS KERNEL BENCHMARK

	Wall clock	CPU	MFLOPS
2 programs run simultaneously	35.34 35.36	34.6813 34.6797	62.64 62.64
Single Run	33.84	33.4193	65.01

VECTOR LOOP PROGRAM

	Total wall Clock time
2 programs run simultaneously	91.58 91.52
Single Run	90.80

Both the NAS kernels and Vector Loop programs were modified slightly by inserting calls to the function TIMEF (which gives wall clock time). This was done so that an approximate maximum speedup could be calculated.

The Vector Loop Program was further modified so that instead of every tenth array element being printed out, as is done in the original benchmark, every array element is printed out (the parameter PRTINC was changed from 10 to 1). This was done in order to see what effect a large amount of I/O would have on interprocessor resource contention.

The speedup is calculated as suggested in the CRAY Multitasking Users Guide:

$$\text{speedup} = \frac{\text{execution time of uniprocessor run}}{\text{execution time of multiprocessor run}}$$

where the uniprocessor run refers to the run of the original unmodified program executing on one CPU in dedicated time, and the multiprocessor run represents the run of the multitasked program executing on two CPU's in dedicated time.

In our example the execution time of the multiprocessor run is approximated by taking the average of the wall clock times of the two simultaneous runs. The execution time of the uniprocessor run is approximated by doubling the wall clock time of the unmodified program run. The results are shown below:

NAS KERNEL BENCHMARK

$$\text{Speedup} = \frac{3.84 \times 2}{35.35} = 1.914$$

VECTOR LOOP PROGRAM

$$\text{Speedup} = \frac{90.80 \times 2}{91.55} = 1.983$$

As Bailey pointed out, this is a relatively easy way of making some estimates about multitasking. It is not true multitasking.

EXPERIMENT 2

In this effort, we decided to gain some experience with multitasking. To save time from developing an algorithm and program ourselves, we looked for a benchmark program from the collection that could be quickly and easily multitasked. The Vector Loop Program was chosen for a number of reasons. First of all, it has the largest execution time of any of the benchmarks (approximately 90 seconds). Second, the structure of the program is amenable to multitasking. It is structured as one major loop which consists of 17 well defined, independent minor loops. One execution of the major loop causes all 17 of the minor loops to be executed once. Because these minor loops are independent, any combination of them could be executed concurrently.

The first step in multitasking was to acquire more information about the execution times of the various parts of the program and to determine the use and scope of the data. Two utilities are available to give more information about the execution time of various parts of the program. Flowtrace summarizes the number of calls to subroutines and what portion of the program's time is spent in those subroutines. SPY samples the program while it is executing and reports on the number of times it found the program working in certain label groupings as it samples. It can be used to identify frequently executed portions of the program. Neither of these utilities provided the necessary information for this particular program and therefore the TIMEF function was inserted in the code at appropriate places in order to determine the time used by each of the 17 individual loops.

Another utility, FTREF, was found to be extremely useful in analyzing the use and scope of the data. Using this information, the program was split by placing some of the minor loops in a subroutine and having the main program call this subroutine as a task with a call to TSKSTART. The strategy for determining the split of the two parts was twofold. First and foremost, we want the time spent in the subroutine to be approximately equal to the time

spent in the main program (between the call to TSKSTART and TSKWAIT). If possible, we would like the subroutine to finish slightly ahead of the main program because the overhead for executing a TSKWAIT is considerably less if the program making the call to TSKWAIT (in our case the main program), doesn't have to wait. The second thing that was considered when splitting the program was memory contention. Even though the 17 loops were independent, some of them were still accessing the same memory (shared, read only memory). We did not want 2 parts trying to read the same piece of data at the same time. Keeping this information in mind as well as the timing requirement, several different versions were created and tested. The results for the most successful version are given below. The time for a dedicated run of a single execution of the original version of Vector Loop Program is given for comparison.

	Total wall clock time	Speedup of multitasked version
Single Run	89.57	
Multitasked Run	48.05	1.86

It must be kept in mind that this was a rather simplistic exercise in multitasking. It was a relatively small piece of code which had few dependencies and which required very little synchronization. It did, however, allow us to gain some experience in, and appreciation for, the intricacies of multitasking.

FINAL REMARKS

It is difficult to draw conclusions from a comparison of these benchmarks. Part of the reason is due to the inherent difficulties involved in performance testing itself, as explained by Worlton (ref. 3) and reinforced by Bailey (ref. 1). That is, sometimes runs are made with tuned versions of benchmarks involving minor or major changes which exploit the best features of a compiler or architecture. Sometimes compiler versions are not noted, or differences in operating system conditions are not exposed.

We ran all of these benchmark programs as received. No changes were made. An interesting and important point to make is the following. All five of them are different, but parts of some of them do similar things. Suppose we observe a single figure (rate) from each of the benchmarks and compare them, e.g.,

NAS Kernels Total MFLOPS	65
LINPACK System Solver	22
Sandia SPEED Total MFLOPS	13
Whetstone MWIPS	25
Livermore Loops Harmonic Mean	10

The Sandia SPEED program gives the lowest performance figure while the NAS kernels program gives the highest figure. However, this comparison is not even meaningful because as we explained in section 2, the meaning of MFLOPS differs among all five due to the fact that each one counts floating point operations in a different way. Furthermore, one can increase these figures by modifying (or tuning) the programs in various ways.

These benchmarks might be used in a relative sense to see how rates are affected when changes are made to the compiler or operating system. Also, one could choose a particular routine or a particular segment of code from this benchmark collection and study its timing information. However, it's clear that before making comparisons or trying to draw conclusions, one should understand how rates (e.g. MFLOPS) are calculated, exactly what kind of algorithms or calculations the code is doing, and whether or not the algorithms and code are written to exploit the features of a compiler or architecture.

REFERENCES

1. Bailey, D.H.; and Barton, J.T.: The NAS Kernel Benchmark Program., NASA TM-86711, 1985.

2. Jefferson, T.H.; and Scott, M.R.: Megaflop Comparisons of Various Computers., Sand 80-2205, Oct. 1985.

3. Worlton, Jack: Understanding Supercomputer Benchmarks., Datamation, vol. 30, no. 14, Sept. 1, 1984, pp. 121-130.

4. Bailey, Dave H.: NAS Kernel Benchmark Results. First International Conference on Supercomputing Systems, IEEE, 1985, pp. 341-345.

5. Dongarra, J.J.: Performance of Various Computers Using Standard Linear Equations Software in a FORTRAN Environment., Argonne National Laboratory, TM No. 23, May 1986.

6. Serlin, O.: MIPS, Dhrystones, and Other Tales., Datamation, vol, 32, no. 11, June 1, 1986, pp. 112-118.

7. Curnow, H.J.; and Wichmann, B.A.: A Synthetic Benchmark., The Computer Journal, vol. 19, no. 1, 1975, pp. 43-49.

8. Rau, B.R.: Don't Measure Only Peak Performance., Information Week, Oct. 13, 1986, p. 56.

9. Dongarra, J.J; and Eisenstat, S.C.: Squeezing the Most Out of an Algorithm in CRAY FORTRAN. ACM Trans. Math. Software, vol. 10, no. 3, Sept. 1984, pp. 219-230.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-88956		

4. Title and Subtitle	5. Report Date
A Comparison of Five Benchmarks	February 1987
	6. Performing Organization Code: None

7. Author(s)	8. Performing Organization Report No.
Janice E. Huss and James A. Pennline	E-3411
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135	
	13. Type of Report and Period Covered: Technical Memorandum
12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546	

15. Supplementary Notes

Janice E. Huss, Summer Faculty Fellow; present address: Utica College, Utica, New York. James A. Pennline, NASA Lewis Research Center.

16. Abstract

Five benchmark programs were obtained and run on the NASA Lewis CRAY X-MP/24. A comparison was made between the programs codes and between the methods for calculating performance figures. Several multitasking jobs were run to gain experience in how parallel performance is measured.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Benchmark programs Cray X-MP/24 MLOPS Multitasking	Unclassified - unlimited STAR Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified		

*For sale by the National Technical Information Service, Springfield, Virginia 22161