Recent tests of supercomputers may have delivered a bum rap to American machines. Here's how to read the numbers to get the answers.

UNDERSTANDING SUPERCOMPUTER BENCHMARKS

by Jack Worlton

A lot of attention has recently been focused on some preliminary benchmark data from Japanese supercomputers. Some observers, failing to understand the limitations of these data, have based unwarranted interpretations on them. The press has run headlines like "Japanese Firms Build Two Fastest Computers" and "Defense Official Asserts Japan Has Two-Year Computer Lead." I hope this article will clarify the issues raised by these data. Specifically, it will provide background information on the nature of benchmarking, especially benchmarking in a scientific computing environment. It will present the benchmark data currently available, and will provide some analyses and conclusions from these data.

The reader should first be aware of the limitations of the benchmark data. No statistically valid study of the performance of these computers has yet been completed. Only anecdotal evidence is available at this time. Preliminary benchmark data are time-dependent, i.e., these results are quickly superseded by new results. Small changes in the compiler and the program optimization level often lead to significant changes in measured performance of vector processors like those being compared here.

It should also be noted that there is a fundamental difference in design between the Japanese machines and the Cray X-MP, to which they are being compared. The Japanese computers are single-processor designs, while the Cray X-MP is a dual-processor design. But so far, all of the benchmark data for the X-MP are for a machine with just one processor, and it has been necessary to extrapolate these data in order to estimate the performance of the full X-MP.

In scientific computation, real-world problems are described in terms of the laws of physics. In turn, these laws are expressed by mathematical models. The models are implemented in algorithms adapted to the architecture of the computer to be used, and the algorithms are implemented in an application program, as illustrated in Fig. 1. (The solid lines in the figure represent the normal processes of scientific computation; the dashed lines represent the additional activities pertinent to benchmarking.) A program developed for a supercomputer is often adapted to that specific computer's architecture in order to achieve very high performance. When a new computer is considered for use in this environment, it must be tested (benchmarked) to determine how it would perform. The type of testing depends on how the site plans to make use of the new computer.

There are three generic approaches to transporting programs to a new computer system. They should determine how a computer is benchmarked.

- Type A: The dusty deck approach makes no changes to the source code but relies solely on what the compiler and other system software can do to optimize the source code for the new computer. This minimizes conversion costs but penalizes the performance obtained.
- Type B: The reprogramming approach modifies the application program to assist the system software in the conversion process. This increases conversion costs but produces higher performance.
- Type C: The rethinking approach modifies not only the application programs but also the algorithms and the mathematical models on which the application programs are based. This is the most demanding approach and yields the highest performance.

Thus, how a site intends to use a new system should decide how the benchmarking is done. The Type A site should make no changes to source codes; the Type B site should make only application-program changes; and the Type C site should do radical restructuring of the mathematical models, algorithms, application programs, and system libraries.

All benchmarking efforts are constrained by practicality, of course. It may take as many as 10 person-years of effort to fully convert a large application program (perhaps 100,000 lines of source code) to a new computer architecture through rethinking the methods from scratch. Therefore, practical benchmarking is done by creating a subset of the work load that retains the essential characteristics of the set of working programs.

WORKLOAD BROKEN DOWN

Through workload characterization the key programs that characterize the work load are selected, and the fraction of the workload each represents is determined. Because the programs themselves are too large to use, subsets of these programs, called kernels, are then extracted from the full programs. These are then converted to run on the target machines. Then, using the workload fractions as weights, subsequent runs are timed and compared.

By definition, then, a computer benchmark is a program or a section of a program that is executed for timing purposes. This method of testing computer performance has several advantages.

- The kernels are tractable. Being relatively short programs, kernels are easy to use and understand.
- The kernels are real. Actual program sequences are used rather than instruction counts or hypothetical instruction sequences.
- Kernels can be used as standard tests. Certain sets of kernels have become widely available, and they therefore can be used to compare the performance of many computers.

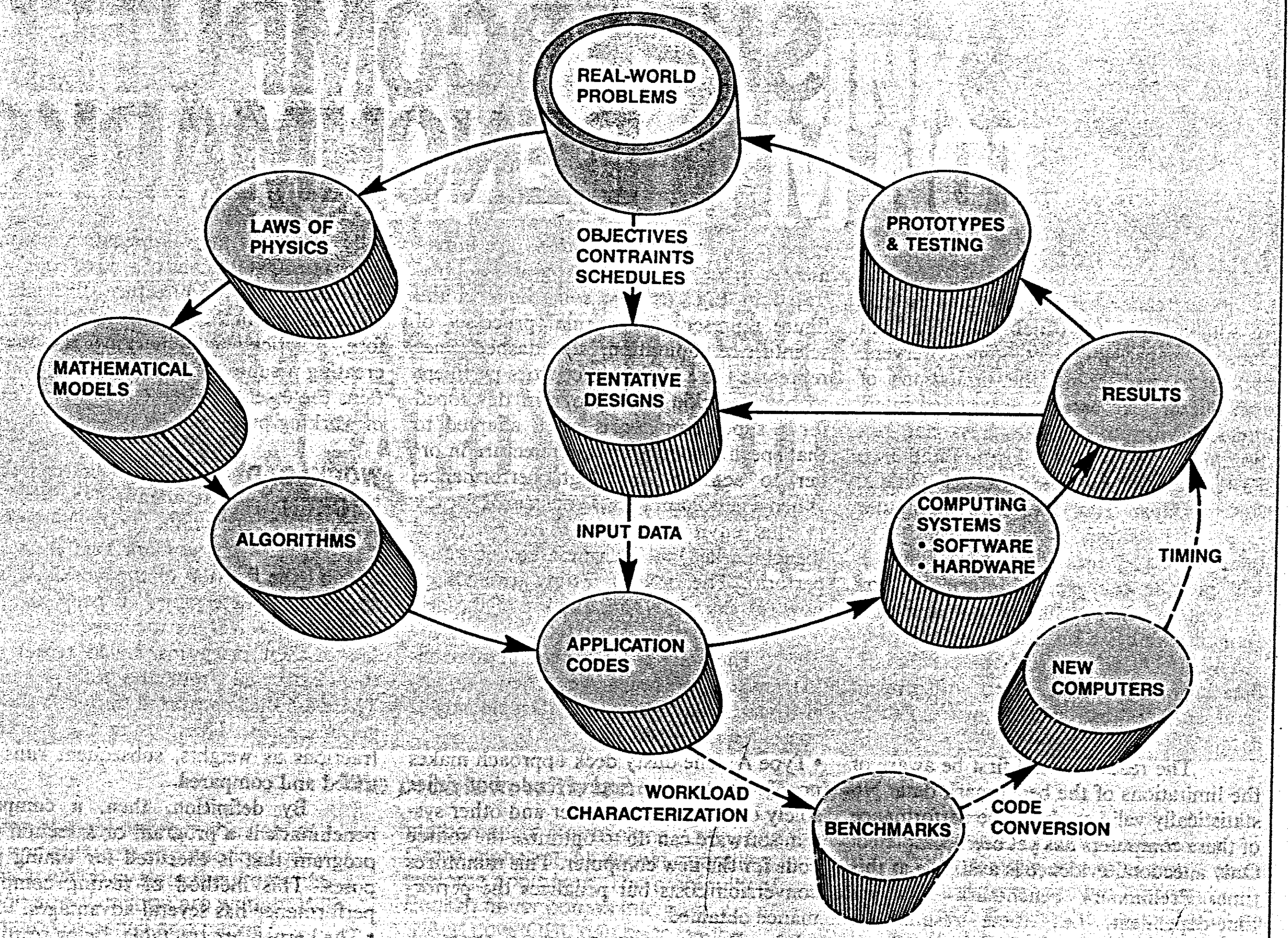
Nevertheless, the benchmark method also has its disadvantages.

- Kernels may be too "easy" in that they may miss important limitations (bottlenecks) of the target machine—limitations that the full program would expose. Memory and input-output bottlenecks are obvious examples. Benchmarks may be so small that they easily fit into main memory without calling on, say, disks or other levels in the storage hierarchy.

How a site intends to use a new system should decide how the benchmarking is done.

FIG. 1

A MODEL OF SCIENTIFIC COMPUTATION



• The sets of kernels are rarely statistically weighted representations of the workloads of interest. For example, the 14 Livermore kernels used in some of the benchmarks reported below do not each represent exactly 1/14th of the Livermore workload.

• Kernels are often obsolete. Most kernels are based on the physics, mathematical models, algorithms, and application programs developed for computers of generation N; those that will be used for generation N+1 are of greater interest. A scalar benchmark run on a vector processor may not demonstrate adequately the performance of the vector processor; a vector benchmark developed for a vector processor of generation N may not demonstrate adequately the performance of a vector processor of generation N+1; and a uniprocessor benchmark may not demonstrate adequately the performance of a parallel processor.

The kernels used in the benchmarks have many of these advantages and disadvantages, so it is important to be wary of apparently simple interpretations of these, or indeed any, benchmark data. Benchmarks provide a useful first step in estimating performance and guiding further studies. But as procurement interests deepen, benchmarking must be followed up with increasingly detailed workload characterization and execution of full programs on the target machines.

The key problem in the use of benchmarks is assuring comparability. The criteria for comparability are site specific and depend on whether the site plans to use the dusty deck, reprogramming, or rethinking approach to program conversion. If a given benchmark is executed on two machines, and that is all that is known, what can be said about the relative performance of the two machines? Not much.

SOME IMPORTANT QUESTIONS

If we are to obtain comparisons that are valid in even the most elementary sense, we have to ask several questions: Were any changes made to the mathematical models or algorithms for any of the machines being compared? If so, were the changes equivalent across all the machines tested? Were any of the kernels optimized on any of the machines by modifying the source or object code? Were the optimization levels comparable? Were the configurations comparable? Also important, are the memory capacities of the test environment the same, and will these be the same in a production environment? Were the peripherals comparable? Were the compilers comparable? Were compiler direc-

The key problem in the use of benchmarks is assuring comparability.

FIG. 2

BENCHMARK COMPARISONS

KERNEL NUMBER	SYSTEM A (MFLOPS)	SYSTEM B (MFLOPS)
1 Hydro excerpt	250.0	228.0
2 MLR, inner product	300.9	239.4
3 Inner product	322.1	211.9
4 Banded linear eq.	91.9	59.2
5 Tri-diag. elim. (below)	10.8	5.4
6 Tri-diag. elim. (above)	10.8	4.6
7 Eq. of state excerpt	254.6	232.7
8 P.D.E. integration	85.0	48.8
9 Integer predictors	226.7	207.6
10 Difference predictors	62.7	49.0
11 First sum	9.8	9.8
12 First diff.	104.1	93.0
13 2-D particle pusher	4.2	4.2
14 1-D particle pusher	7.7	8.5
"Average"	**124.4**	**100.2**
Harmonic mean	**19.4**	**14.7**

FIG. 3

SOME BENCHMARKS USING THE LIVERMORE KERNELS (MFLOPS)

KERNEL	CRAY-1 CFT '78	CRAY-1 CFT '84	X-MP-1 CFT '84	FUJITSU VP-100	FUJITSU VP-200	HITACHI S810-20
1	62.5	100.0	153.0	187.0	326.4	228.0
2	15.2	41.7	76.6	104.6	178.1	239.4
3	3.3	33.3	95.8	168.0	331.1	211.9
4	2.9	24.3	41.1	73.6	88.0	59.2
5	4.0	7.7	8.7	10.0	10.0	5.4
6	4.6	7.0	8.0	9.5	9.5	4.6
7	80.0	120.0	167.9	190.0	326.1	232.7
8	8.5	55.4	95.7	86.3	90.4	48.8
9	77.3	68.0	163.0	161.5	257.4	207.6
10	3.1	36.0	59.9	50.1	84.8	49.0
11	2.0	2.9	3.1	4.8	4.8	9.8
12	21.3	25.0	76.5	58.8	114.1	93.0
13	2.1	4.0	4.8	6.1	6.2	4.2
14	3.7	5.6	6.9	12.9	13.9	8.5
H_m	4.7	11.1	13.7	18.7	19.8	14.7
HIGH	80.0	120.0	167.9	190.0	331.1	239.4
LOW	2.0	2.9	3.1	4.8	4.8	4.2
RATIO	40:1	41:1	54:1	40:1	69:1	57:1

tives inserted in some programs to aid the compiler vectorization but not in other programs?

We can summarize the minimal steps that should be taken in benchmarking when kernels are to be used.

- Step 1: Conduct a workload characterization study. The output of this study should be the rates, R_i, that are characteristic of the programs in the workload, together with their workload fractions, f_i, where the sum of the fractions equals 1.
- Step 2: Select a subset of the programs to represent the whole workload; these should include the programs having relatively large f_i; renormalize the fractions to represent the whole workload.
- Step 3: Select portions (kernels) of these programs to represent the whole programs. This is the crucial step in successful benchmarking.
- Step 4: Time the kernels to obtain the kernel execution rates.
- Step 5: Compute the weighted harmonic mean.

Researchers conducting actual benchmarking studies often fail to conduct a workload characterization study, depending instead on intuitive ideas of the programs that are most important and of what their weights should be. They can also select kernels that are too easy and hence do not adequately represent the complete programs. And they use the arithmetic mean rather than the harmonic mean as a workload measure.

As a concrete example of the difficulty of comparing benchmarks, consider the actual benchmarks in Fig. 2. What can we conclude from this comparison of benchmarks? It would appear that System A is some 24% faster than System B, right? Dead wrong. These benchmarks were executed on the same computer, the Hitachi S810-20. System A data were generated at the University of Tokyo (see Fig. 5) and System B data were obtained by personnel from the Magnetic Fusion Energy Computer Center (see Fig. 3). Clearly, the benchmarks were run under different conditions, so the results cannot be compared without an understanding of these conditions. In this example, the conditions caused the performance ratios to vary by a factor of more than 2.5.

A further point about Fig. 2 concerns the average of these results. The average was computed using an unweighted arithmetic mean, and the arithmetic mean often gives distorted perspectives of average rates.

Consider the following example. Suppose I want to travel 100 miles, and part of the journey I travel at 5 mph and part at 55 mph. What is my average rate? The average of these two rates appears to be $(5 + 55)/2 = 30$ mph, but in fact we can't even answer the question without further information: how much distance was covered at each of these rates? If I travel 45 miles at 5 mph and 55 miles at 55 mph, then my average rate would be just 10 mph ($= 100/(45/5 + 55/55)$), not 30 mph. Just so with averaging computer execution rates: we must use the harmonic mean or run the danger of being wrong by hundreds of percentage points in estimating computer performance.

The weighted harmonic mean, H_m, is computed as follows:

The arithmetic mean often gives distorted perspectives of average rates.

$$H_m = \frac{1}{\sum_i f_i/R_i},$$

where the f_i are the fractions of the workload corresponding to the execution rates, R_i. If the fractions, f_i, are unknown, the unweighted harmonic mean can be computed using the assumption that all the weights are equal to $1/I$, where I = the number of programs we are using:

$$H_m = \frac{1}{(1/I) \sum_i (1/R_i)},$$

In contrast, the unweighted arithmetic mean is computed as:

$$A_m = (1/I) \sum_i R_i.$$

MEANS CAN DIFFER WIDELY The arithmetic mean and the harmonic mean can often differ by large factors. For example, the arithmetic mean of the rates for System A in Fig. 2 for the S810-20 is 124.4 MFLOPS, but the harmonic mean is only 19.4 MFLOPS—the two means differ by a factor of 6.4. The harmonic and arithmetic means are equal only when (a) the rates are all equal, or (b) the time spent executing at each rate is equal.

With these factors in mind, look at some benchmark data for some recently announced supercomputers.

Personnel from the Department of Energy's Magnetic Fusion Energy (MFE) Computer Center at Lawrence Livermore National Laboratory, Livermore, Calif., visited Japan and obtained data for the Livermore kernels on the Fujitsu VP-100 as well as the Hitachi S810-20. (The Livermore kernels are short FORTRAN kernels, a few lines each, that have been abstracted from actual programs used at the Lawrence Livermore National Laboratory. No statistical weighting of these kernels exists as a workload characterization, and no claim is made that they represent the current Livermore workload.) Data for the VP-200 have since been released by Amdahl Corporation. These results were compared to earlier benchmarks on other computers, generating the data in Fig. 3. The study by MFE was carefully controlled: no changes in source code or insertion of compiler directives were allowed.

FIG. 4

CRAY RESEARCH BENCHMARK DATA (MFLOPS)

	CRAY-1 CFT '78	CRAY-1S W/X.14	X-MP-1 CFT '82	X-MP-1 W/X.14	X-MP-2* W/X.14
H_m	4.7	13.3	11.0	19.4	35.0
High	80.0	91.8	150.7	164.1	295.4
Low	2.0	4.4	3.1	5.6	10.1
Ratio	40:1	21:1	49:1	29:1	29:1

*The performance of two PEs is projected using an assumed speedup of 1.8.

FIG. 5

BENCHMARK DATA FROM THE UNIVERSITY OF TOKYO (MFLOPS)

KERNEL	HITAC M-280H (UNI-CPU)	HITAC S810-20
1	25.7	250.0
2	12.6	300.9
3	32.4	322.1
4	24.6	91.9
5	5.3	10.8
6	4.5	10.8
7	20.3	254.6
8	10.3	85.0
9	14.6	226.7
10	5.0	62.7
11	6.6	9.8
12	25.9	104.1
13	1.8	4.2
14	2.6	7.7
H_m	6.5	19.4
High	32.4	322.1
Low	1.8	4.2
Ratio	18:1	77:1

Based on these data, the workload represented by these kernels would run 1.68 times as fast on the VP-100, and 1.78 times as fast on the VP-200, as on the Cray-1S. For the Cray X-MP-1* these ratios would be 1.36 and 1.45, respectively; no data are available for the X-MP-2.

Cray Research has recently run these same Livermore kernels on a Cray X-MP-1 using the newest version of their compiler, X.14. This is not yet a production compiler, and the reason for including data describing its performance is to illustrate the large variance in benchmark data. The results are given in Fig. 4 for this machine and several others, including some earlier data for the Cray-1 and the X-MP-1 for reference. Here we project the performance of the Cray X-MP-2 as S2 = 1.8. Readers, however, can adjust this column depending on their preferred value of S2, the speedup that full two-processor operation will produce. Because this compiler is not yet a released product, the details of the kernel results are still proprietary and only a summary is available for publication.

It appears from these data that the Cray X-MP-1 will run about 1.78 times as fast as a Cray-1S when both use the X.14 compiler. How much faster the X-MP-2 will

*Cray X-MP-1 and Cray X-MP-2 refer to the Cray X-MP with one and two processing elements, respectively.

The best we can conclude is that the VP-200 and the X-MP-1 would run with roughly the same execution rate.

run compared to the Cray-1S depends on the speedup achieved with two processors, and this is application dependent. The harmonic mean for the X-MP-1 using the X.14 compiler in Fig. 4 is about the same as for the VP-200 in Fig. 3, indicating that these computer/compiler systems would give about the same performance on this work load, assuming equal weights for the kernels. The workload fractions corresponding to each kernel would be needed to make a more precise comparison, however.

The University of Tokyo Newsletter for October 1983 published benchmark results from runs of the Livermore kernels made on the Hitachi S810-20, as shown in Fig. 5, along with some data for the Hitachi M-280H with an integrated array processor (IAP). Notice that the data in Fig. 5 are higher than the comparable data for the Hitachi S810-20 in Fig. 3, even though the same kernels and computer were used. According to the newsletter, the loops are the original kernels except for the correction on the clock overhead and accuracy.

In a production environment, programs are usually optimized by identifying and correcting those sections with low performance. To simulate that environment from these preliminary data, the best we can do is to compare (and it is a highly uncertain comparison) the best benchmark data for each computer. This is done in Fig. 6. The figures for the Fujitsu VP-100 and VP-200 come from Fig. 3; the figures for the Hitachi S810-20 from Fig. 5; and the figures for the Cray-1S, Cray X-MP-1 and X-MP-2 from Fig. 4.

FIG. 6

COMPARISON OF BEST PERFORMANCE DATA

	VP-100	VP-200	S810-20	CRAY-1S	X-MP-1	X-MP-2*
H_m	18.7	19.8	19.4	13.3	19.4	35.0
H_m-Norm**	1.41	1.49	1.46	1.00	1.46	2.63
High	190.0	331.1	322.1	91.8	164.1	295.4
Low	4.8	4.8	4.2	4.4	5.6	10.1
Ratio	40:1	69:1	77:1	21:1	29:1	29:1

*Simulated by assuming a speedup of S_2 = 1.8
**Normalized to the Cray-1S with the X.14 compiler

FIG. 7

BENCHMARK DATA FROM PROFESSOR MENDEZ (TIME IN SECONDS)

	VP-200		1-PE CRAY X-MP	
TEST	SCALAR	VECTOR	SCALAR	VECTOR
VORTEX				
I = 500	217.2	34.4	233.6	37.8
EULER				
I = 1000	6.3	4.8	9.0	3.1
I = 8608	NA	41.1	NA	27.9
2DMHD	43.4	2.6	39.2	4.3
SHEAR3	164.4	83.6	190.3	72.7
BARO	1107.8	41.1	756.9	76.3

NA = not available

FIVE BENCHMARK TESTS RUN

Professor Raul Mendez of the Naval Post-Graduate School in Monterey, Calif., visited Japan in late 1983 and ran benchmarks on the VP-200. These tests included five benchmarks run in both scalar and vector mode. Compiler directives were inserted into two of the Fujitsu programs (SHEAR3 and 2DMHD). These directives essentially caused vectorization of some loops that would otherwise not have been vectorized, but the quantitative effects of this action are uncertain. The benchmarks were subsequently run by Professor Mendez on the Cray X-MP and the results were published in the SIAM News in January and March. Subsequently, Cray Research assisted Professor Mendez with the vector versions (but not the scalar versions) of these benchmarks, and those data, which update the earlier publication, are included in Fig. 7. Fujitsu published a paper at the 1983 IFIP Congress describing its work on vectorizing techniques. The advantages for the VP-200 compiler in these and other data are probably due in part to the vectorization methodology described by Fujitsu in its IFIP paper, and in part to the hardware features included in the Fujitsu VP-200 that permit ease of vectorization.

The automatic vectorization techniques used by Fujitsu are based in part on work done in the United States by Professor David Kuck and his associates at the University of Illinois. The special hardware features include a constant stride in vector addressing and vector indexing operations like gather-scatter and compress-expand. The Cray-1 design includes a constant stride and the CDC Cyber 205 includes vector indexing operations, but neither American design offers both features. Professor Mendez's results are shown in Fig. 7. Note that these data are times, not MFLOPS.

Some of these results indicate an advantage to the Cray X-MP-1 and some indicate an advantage to the VP-200. Because so little is known about the optimization of these programs for these computers, the best we can conclude is that the VP-200 and the X-MP-1 would run with roughly the same execution rate. Any advantage of one over the other would be due to the application used in the test.

Based on the data available now, one can make the following tentative conclusions:

- Optimized programs for the Cray X-MP-1, the Fujitsu VP-200, and the Hitachi S810-20 should achieve roughly comparable performance in general purpose computing environments.
- The relative performance of the Cray X-MP-2 will be higher by the speedup achieved with its two processors.
- The vectorization methods used in the Japanese compilers appear to be better than the methods used in the production versions of American compilers. This is due in part to the software vectorization methods used and in part to the data-handling features of the hardware.
- Only site-dependent benchmarking can determine which of these computers will perform better for a given workload. ◉

Jack Worlton is a laboratory fellow at Los Alamos National Laboratory.