[illegible]-81-1977

CONF - 811202 - -2

TITLE: COMPARATIVE PERFORMANCE EVALUATION OF TWO SUPERCOMPUTERS: CDC CYBER-205 AND CRI CRAY-1

AUTHOR(S): Ingrid Y. Bucher

James W. Moore

SUBMITTED TO: Computer Measurement Group XII meeting, New Orleans, LA, December 1-4, 1981

COMPARATIVE PERFORMANCE EVALUATION OF TWO SUPERCOMPUTERS: CDC Cyber-205 and CRI Cray-1

by

Ingrid Y. Bucher and James W. Moore

ABSTRACT

This report compares the performance of Control Data Corporation's newest supercomputer, the Cyber-205, with the Cray Research, Inc. Cray-1, currently the Laboratory's largest mainframe. The rationale of our benchmarking effort is discussed. Results are presented of tests to determine the speed of basic arithmetic operations, of runs using our standard benchmark programs, and of runs using three codes that have been optimized for both machines: a linear system solver, a model hydrodynamics code, and parts of a plasma simulation code. It is concluded that the speed of the Cyber-205 for memory-to-memory operations on vectors stored in consecutive locations is considerably faster than that of the Cray-1. However, the overall performance of the machine is not quite equal to that of the Cray for tasks of interest to the Laboratory as represented by our benchmark set.

1. INTRODUCTION

To satisfy its scientific computing needs, Los Alamos National Laboratory has traditionally required the largest and fastest computers commercially produced. We therefore have an ongoing benchmarking effort to obtain performance data on new machines as they become available. This report compares the performance of Control Data Corporation's newest supercomputer, the Cyber-205, with the Cray Research, Inc. Cray-1, currently the Laboratory's largest mainframe.

[illegible]

The rationale of our test design is discussed in the light of how new supercomputers are put to work at the Laboratory. We then describe very briefly basic hardware features of the two machines. Finally, along with a description of our test programs, timing results for both machines are presented and discussed.

2. DESIGN OF TESTS

Use of new supercomputers at the Laboratory typically begins with the transfer of FORTRAN codes already running on other computers. Extensive revisions will be required if the architecture of the new machine differs significantly from the old. Initial modifications to the codes are limited to those necessary to get useful work done on the new machine. Once codes are running, we optimize to get higher performance. Optimizations range from minor changes in structure or syntax (to help the compiler) to total redesign of the program, including the use of new and different algorithms.

Our benchmarking effort is aimed at each stage of code implementation on the new machine. Ideally we would like to get performance estimates for

(a) codes with only minimal modifications to get them running on the new equipment,

(b) codes that have been optimized for the new machine with a reasonable investment of programming time, and

(c) optimal performance.

Due to obvious limitations, we make little effort to use new algorithms in benchmarking but explore other possibilities in whatever detail time allows.

The Computing Division maintains a set of portable benchmark programs that represent characteristic tasks that a new large computer would be required to run at the Laboratory. We also maintain a data base containing results of past runs of these programs on a variety of computers [1]. The programs are coded in ANSI FORTRAN for portability, and we are typically able to run them with little or no change. Runtimes will be indicative of the potential initial usefulness of the new machine.

If time permits, we optimize some of the benchmark programs for the machines to be tested. For the comparison of the Cyber-205 to the Cray-1 we selected three programs for optimization: a linear system solver based on the LINPACK system [2]; a model hydrodynamics code, SIMPLE, that solves the equations of Lagrangian flow and the heat diffusion equation by finite difference methods [3]; and parts of a particle-in-cell plasma simulation code [4]. For the Cyber-205, optimizations ranged from inserting explicit vector syntax into the FORTRAN code, inserting "Q8" calls that compile into specific assembly language instructions, to coding sections in META (the Cyber-205 assembly language). Cray-1 optimizations included restructuring FORTRAN code and writing sections in CAL, the Cray assembly language. The three codes were optimized at different levels. For the linear system solver high gains in speed could be achieved with minimal effort for the Cyber-205. For the hydrodynamics program, we forced the CDC compiler to vectorize major portions of the FORTRAN code by explicit vector syntax and Q8 calls; basic algorithms were left unchanged. For the plasma simulation code we carried optimization to the ultimate achievable for both machines, coding major sections in assembly language as needed.

To verify advertised performance, we wrote a set of programs to measure the speed of repetitive arithmetic operations.

The Cyber-205 is a virtual memory machine. We made every effort to suppress this feature for two reasons:

1. we were concerned with the best performance of the machine, and

2. our major codes would typically remain memory resident and handle their own input/output.

Our tests used only 64-bit arithmetic because 32-bit arithmetic is too short for most calculations of interest to us.

All of our development work was done on the Cyber-203 rather than on the Cyber-205. The main difference between the two machines is the new, faster vector processor of the Cyber-205. The Cyber-203 was accessed remotely through Cybernet, which significantly reduced the development time.

3. CHARACTERISTICS OF CYBER-205 AND CRAY-1 HARDWARE

The Cyber-205 and the Cray-1 have many features in common. Both contain pipelined vector-arithmetic processors capable of performing operations on arrays at very high speeds. In addition, each has a scalar processor subdivided into several functional units capable of performing a variety of operations in parallel with each other and with the vector pipelines.

The Cray-1 vector processor is subdivided into multiple units permitting multiple results each cycle. The Cyber-205 can have one, two, or four vector pipelines, each capable of producing one result each cycle. The cycle times are 12.5 ns for the Cray-1 and 20 ns for the Cyber-205.

The access to memory by the vector units is the characteristic difference between the two machines. The Cyber-205 vector instructions reference memory directly. The Cray-1 has 8 vector registers, each 64 words long, usually making it unnecessary to store intermediate results in memory. Ease of access to the vector registers combined with concurrent execution of vector operations accounts for the high speed of the Cray-1 when each operand does not have to be fetched from memory or each result stored. The Cray-1 processes long vectors in sections of length 64, the Cyber-205 in sections of up to 65,536. If the Cyber-205 is to achieve peak result rates, the vectors must be contiguous in memory. For the Cray-1, vectors must have a constant stride (distance between memory locations), but the stride need not be 1. For both machines, data stored at irregular locations in memory must be "gathered," a process that slows down operations of both machines. The results may then have to be "scattered" back into memory. The Cyber-205 has vector instructions to perform gathers and scatters: "transmit indexed list" and "transmit list indexed," respectively [5]. The Cray-1 performs gathers and scatters in scalar mode only.

The Cyber-205 tests were run on the first production machine, Serial 502, which was equipped with two vector pipes.

4. DESCRIPTION OF TEST PROGRAMS AND TIMING RESULTS

4.1 Speed of Repetitive Arithmetic Operations

The set of programs for this test measures the speed of 64-bit floating point operations as a function of vector length in both vector and scalar modes. The unit of measurement is MFLOPS, millions of floating point operations per second. Typically, one million operations were timed, irrespective of vector length.

4.1.1 Vector Operations for Vectors Stored in Consecutive Locations

Timing results of a program in which vector operations for the Cyber-205 were specified in vector syntax are given in Table I. The corresponding ANSI FORTRAN program generated identical timings. Results obtained by running the ANSI FORTRAN version on the Cray-1 are presented in Table II.

It is evident that the vector unit of the Cyber-205 is indeed as fast as advertised. With two vector pipelines, it not only exceeds the speed of the Cray-1 for long vectors by a factor of approximately 3, but it is about twice as fast as the Cray-1 for vector lengths of 100, at least for memory-to-memory operations in a FORTRAN context.

For the Cyber-205, the average time T required to perform a vector operation on a vector of length $N < 65,537$ is given by

T(Cyber) = TSTART_VU + N*T_EL (1)

where TSTART_VU is the time required to start up the vector unit and T_EL is the interval between successive result elements emerging from it. For the vector add, T is plotted as a function of the vector length N in Fig. 1 along with results obtained for the Cray-1. The linear relationship of Eq. (1) is evident. Table III presents values for TSTART_VU and T_EL derived from the measured data for the Cyber-205 vector unit with two vector pipes.

For the Cray-1, the average time T to process a vector of length N is a more complicated relationship than Eq. (1) because vectors of length N > 64 are "stripmined" in sections of length 64. This is evident from Fig. 1 by the steep rise in processing time T each time the vector length exceeds an integer multiple of 64. For N equal to multiples of 64, T is a linear function of N and can be represented by

T(CRAY) = TSTART_OUT + N*(TSTART_STRIP/64 + T_EL) (2)

where TSTART_OUT is the startup time for the outer loop, TSTART_STRIP is the startup time that could not be overlapped with the vector operation for each strip of 64, and T_EL is the time required to process one result element. In Table IV values for TSTART_OUT, TSTART_STRIP, and T_EL are listed for the Cray-1 and the CFT compiler, Version 1.09. For the operations programmed, requiring one memory reference per vector operand, T_EL was determined by the speed with which the Cray-1 can access its memory, namely, in the absence of bank conflicts one load or store operation per 12.5-ns clock cycle. There are two exceptions: for the operations V = V+S*V and V = V*V+V, the compiler generated less than optimal code with T_EL of 50 and 62.5 ns instead of 37.5 and 50 ns, respectively. TSTART_OUT and TSTART_STRIP are compiler dependent. The minimum value of TSTART_STRIP is determined by startup times of vector memory reference and arithmetic operations and is close to the value measured. However, inspection of the compiler-generated assembly code indicates that the value of TSTART_OUT, the startup time for the outer loop, could have been reduced by better compiler optimization.

In comparing the Cyber-205 with the Cray-1 for arithmetic vector operations with vectors stored in consecutive memory locations, results indicate that startup times are comparable but the time required to process an element is considerably less for the Cyber than for the Cray in memory-to-memory operations due to faster memory access.

4.1.2 Vector Operations for Vectors Stored With a Constant Stride

Arithmetic on vectors stored not in consecutive memory locations but with a constant stride is more complicated for the

Cyber-205 [5] than for the Cray-1. Basically there are two options:

1. Perform the arithmetic on all numbers fetched from contiguous locations; then store selected results under control of a bit vector. This option is optimal for small strides. For optimal speed the process has to be coded with Q8 calls that produce inline code, which is cumbersome. For this option, speeds were measured that are consistent with those obtained by dividing the results of Table I by the stride.

2. Do a periodic gather followed by the arithmetic operation and a subsequent periodic scatter. Both periodic gather and scatter are vector operations on the Cyber-205. This option yields the fastest results for large strides.

Data obtained from a version of the test program containing Q8 calls for the periodic gather and scatter operations combined with vector syntax for the arithmetic are given in Table V. The compiler generated similar code for the corresponding ANSI FORTRAN version. The results for the ANSI FORTRAN version run on the Cray-1 are given in Table VI. Clearly, for nonconsecutive vectors the Cray-1 is faster than the Cyber-205 by a factor of more than 2 for nearly all operations and vector lengths.

The results for the average time T required to process vectors containing N elements are plotted for the addition of a scalar to a vector in Fig. 2 for both the Cyber-205 and the Cray-1. Relations similar to Eqs. (1) and (2) seem to hold for both machines, respectively, but with higher times T_EL for the Cyber-205 and higher startup times for both machines. For the Cyber-205 an analysis of the data indicates that 29 ns have to be added to T_EL for each gather of an operand and 25 ns for each scatter of a result element. Startup times for gather and scatter operations are partially overlapped with other operations and could not be determined accurately from the data; they are between 1000 and 2000 ns. Surprisingly enough, startup times for the Cray-1 are somewhat higher than for vectors stored in consecutive locations also. This is due entirely to poor compiler optimization, which we believe could be improved.

In comparing both machines, it is evident that both startup times and times required per vector element are considerably higher for the Cyber-205 than for the Cray-1 when vector operations involve elements stored in memory in a periodic pattern but not in consecutive locations.

4.1.3 Random Gather and Scatter Operations

This section discusses speeds of arithmetic operations for which operands or results or both are stored at irregular locations in memory. For our tests these locations were specified by an index to arrays of the form

INDEX = J(I) + K

where J is an integer array properly initialized (for example, by a random number generator), I is the loop count, and K an arbitrary integer constant.

Results of a program that specified random gather and scatter operations by Q8 calls and arithmetic operations by explicit vector syntax are given in Table VII. The CDC compiler generated erroneous code from the corresponding ANSI FORTRAN version of the program. The fact that an index had to be computed and operands picked from or results stored into locations designated by this index was completely ignored. Picking operands from consecutive locations resulted, not surprisingly, in much higher speeds for the erroneous code than for the correct one. Results obtained by running the ANSI FORTRAN version on the Cray-1 are presented in Table VIII.

Analysis of the data for the Cyber-205 indicates that each random gather or scatter operation requires an additional startup time of approximately 2000 ns for the vector unit. In addition, an average time T_EL = 36 ns is needed to gather each operand element and T_EL = 35 ns to scatter a result element.

Random gathers and scatters on the Cray-1 proceed at much lower speed because, contrary to the Cyber-205, all operations are performed in the scalar unit.

It has to be pointed out that for some important applications vector instructions cannot be used for gather and scatter operations because they would lead to incorrect results. Rates for scalar random gathers and scatters on the Cyber-205 are about 10 to 20% lower than those for the Cray-1.

4.1.4 Scalar Arithmetic Operations

The speeds at which the scalar unit produces results for repetitive operations were measured by the ANSI FORTRAN version of the program used for measuring vector speeds with the vectorization option off for the compilers. A set of results obtained for the Cyber-203, which has an identical scalar unit as the Cyber-205, and for the Cray-1 are listed in Table IX. The results were strongly dependent on the compiler version and optimization level used in compiling the code.

For the nonoptimized compilations, rates on the Cray-1 are higher than those on the Cyber-205 by a factor that equals the ratio of the respective cycle times within a few percent. For compilations with scalar-code optimization, the Cyber is 10% to 30% faster than the Cray. Inspection of the assembly code generated by the compilers provides the explanation of the results. The code produced by the Cray CFT compiler could easily be optimized by hand to yield an increase in speed by a factor of 2 or better for most operations. The code produced by the CDC BENCHFTN compiler was highly optimized and could not be speeded up significantly by hand optimization without extensive loop unrolling. The results for optimized code, therefore, reflect more the higher sophistication of the BENCHFTN compiler to optimize scalar loops as compared to the CFT compiler than properties of the hardware. It is noteworthy that the ratios of runtimes for optimized scalar compilations of the Los Alamos benchmark programs are not as favorable for the BENCHFTN compiler (see Sec. 4.2) as the timings of repetitive operations described above.

4.1.5 Subroutine Calls

. Times required to call a subroutine were measured. For the Cyber-205, times per call ranged from 500 to 16,000 ns, depending on compiler optimization and type of subroutine. The Cray-1 subroutine calls take 720 ns plus 60 ns for each parameter passed. These numbers do not include time losses due to possible less efficient code optimization.

4.2 Results From The Los Alamos Standard Benchmark Programs

Table X contains, along with a brief description of most of the Los Alamos benchmark programs, the CPU times required to run the codes on the Cyber-205 and the Cray-1. The programs were compiled for machines with the vectorization option turned both on and off. Scalar optimization was on in all cases. Timings of scalar versions for the Cyber were obtained from running the codes on the Cyber-203, which has a scalar unit identical to that of the Cyber-205. However, there may be some differences in the capabilities of the respective BENCHFTN compilers to optimize scalar code. Program 11 was not run because of problems with the compiler. Ratios of CPU times required to run the codes on each machine are presented in Fig. 3 for the vectorized versions of the code and in Fig. 4 for the nonvectorized (scalar) versions.

With the exception of Programs 1 and 18, running the codes in vector mode on the Cray-1 required less CPU time (50 to 90%) than on the Cyber-205. Program 1 tests the speed of integer arithmetic, which is traditionally fast on Control Data machines. Program 18 performs simple vector operations on long contiguous vectors, and is the only unmodified program that showed the strength of the Cyber-205.

Except for Program 18, the compiler did not automatically vectorize the programs well for the Cyber-205, as can be seen from the ratios of runtimes for the vectorized and nonvectorized versions. Hand vectorization of Programs 15, 16, and 22 was possible by modifying one line of code (see also Sec. 4.3, "Results of the Linear System Solver Test"). For Program 16, this reduced the runtime from 62.9 to 12.2 s. Program 4A's execution time was reduced from 8.7 to 1.3 s by substituting the Control Data FFT Library routine written by Dennis Kuba [7] for the routine contained in our benchmark.

The ratios of runtimes for the nonvectorized versions as represented in Fig. 4 give an indication of the performance ratio of both machines for scalar codes using the compilers available at present. The average ratio (excluding Program 1) is 0.69, about 10% higher than one would predict considering the ratios of cycle times only. It is interesting, however, to note that the CDC 7600, using the FTN compiler (with opt=2), outruns the Cyber-205 in scalar mode on all codes except the linear system solvers [1] in spite of its higher cycle time of 27 ns and less elaborate architecture. This indicates that there is potential for improvements in both the BENCHFTN and CFT compilers.

Results of the benchmark program tests indicate that most codes in our workload would run more slowly on the Cyber-205 than on the Cray-1 without at least some hand optimization.

4.3 Results from The Linear System Solver Test

The linear system solver test used several variations of Program 22, which contains 450 lines of FORTRAN, including subroutines, and is based on the LINPACK system [2]. It makes calls to a set of standard basic linear algebra subroutines, the BLAS. The algorithm used is Gaussian elimination with partial pivots.

From measurements of where the time was spent in the code, it became clear that any significant speed increase would have to come from improvements in SAXPY, one of the BLAS. SAXPY performs multiplication of a scalar by a vector and the addition of the result to another vector. SAXPY can be executed on the Cyber-205 as a single vector instruction (a linked triad). The program uses FORTRAN versions of the BLAS, so the aim was to encourage the compiler to compile the linked triad. In the original version, the CDC compiler compiled SAXPY as two gathers, a linked triad, and a scatter. This happened because SAXPY was written as a general routine to process either rows or columns, whereas in the LINPACK solver only columns are used. This is an ideal method for the Cyber-205 because it has to process contiguous vectors only. There were no significant differences between the versions for the Cray-1 with contiguous or apparent (to the compiler) noncontiguous vectors.

Significant differences were measured with SAXPY inline as opposed to a subroutine. The two versions of interest are

1. LSS2 - SAXPY is vectorized with contiguous vectors, and

2. LSS5 - one call to SAXPY is replaced by inline code.

The call replaced in LSS5 accounts for nearly all the CPU time attributable to SAXPY. It is clear from this result that subroutine linkage is costly on the Cyber-205 and that small heavily-used subroutines should be replaced by inline code, preferably by the compiler. This is also true for the Cray-1 but to a lesser degree. Time losses due to subroutine calls for SAXPY were calculated as 4 μs per call for the Cyber-205 and 3 μs per call for the Cray-1. Table XI compares CPU times required for 300 solutions of an NxN system using LSS2 and LSS5 for the Cyber-205 and the Cray-1.

Table XII compares CPU times required for 300 solutions of an NxN system using library routines available for both machines. The Cyber-205 routine is a FORTRAN implementation of LINPACK, which might be improved in the future as more experience is gained on the machine. The Cray-1 routine is a highly optimized and vectorized version, written in assembly language.

It can be seen that even though the Cyber-205 is substantially faster than the Cray-1 for the FORTRAN version of this test, hand optimization and recoding brings the Cray-1 up to Cyber speeds for all but very large systems.

4.4 Results from the Simple Test

SIMPLE is a model hydrodynamics code that solves the two-dimensional partial differential equations of Lagrangian compressible flow and the heat conduction equation by finite difference methods [3]. It contains 1900 lines of FORTRAN. The version used for the test vectorized well automatically on the Cray-1 except for the equation-of-state routines, but did not vectorize well on the Cyber-205. The equation-of-state routines contain table lookups and interpolations and can be vectorized with some effort. We did not chose to do so because we felt the code was more representative of actual Laboratory codes in the form used.

Control Data Corporation did vectorize the equation-of-state routines for the Cyber-205, and achieved a speedup of approximately two over the nonvectorized versions but required considerably more memory. Similar speedups by a factor of about 2 are reported for the Cray-1 [8]. The timing for the Cyber-205 vectorized versions were roughly equal to those for the Cray-1 version with scalar equation-of-state routines. In our version, we estimate that about 50% of the execution time is spent in the equation-of-state subroutines. However, a very small fraction of the computational work is performed in them because the rest of the code is vectorized.

The versions of interest for the Cyber-205 are

1. SIMP - the original version, and

2. SIMP4 - some vector syntax and Q8 calls added.

SIMP is actually a slight modification of the Cray-1 version because the compiler generated incorrect code that caused the program to abort. Because the Cyber-205 BENCHFTN compiler did not vectorize as well as the Cray CFT compiler, there were numerous places where either Q8 calls or vector syntax were used to achieve more vectorization in SIMP4. This optimization effort required 1.5 man-months. Part of this time was due to difficulties with the compilers (there were several) and the Cyber utilities. Since all the development work was done on the Cyber-203, we are not sure that we have an optimally efficient code for the Cyber-205.

Several approaches to vectorization could be explored for the Cyber-205. The approach we tried vectorizes the column computations (on the Cray-1 row computations vectorize also). On the Cyber-205, the row computations may compile as gathers and scatters with vector computations in between. This is at least seven times slower than arithmetic on contiguous vectors. Another approach would be to vectorize matrix operations over the entire mesh wherever possible into one single vector operation and subsequently clean up the boundaries. There is vector syntax in Cyber-205 FORTRAN that would make this task easier [7]. This approach would lead to additional speedup on the Cyber-205 because of the longer vector lengths, but the boundary might slow down. It is, however, impossible to vectorize some areas of the code in this way because of dependencies within the mesh. Different computational algorithms are needed to avoid row computations, which are slow on the Cyber-205.

Table XIII presents the execution times for SIMP and SIMP4 for 100 time steps on an NxN mesh. The actual mesh is (N+2)x(N+2) because of the boundary around it. The ratio of speeds of the two machines appears almost linear with mesh size. If it were linear, the crossover point would be on about a 350x350 mesh, which would require 2.7 million words of storage. Run times for this size problem would be about 330 s for 100 time steps. Since the number of time steps for real problems of this type is typically two to three orders of magnitude higher and real codes are much more complex, it seems infeasible to run problems of this size on present machines, including the Cyber-205.

4.5 Results From The Optimized Plasma Simulation Code

Benchmark Program 11 contains the "particle pushers" from a particle-in-cell (PIC) plasma code that simulates the motion of charged particles in an electromagnetic field [4]. Although the particle-push subroutines (PARMVE, PARMOV, and PARMVR) constitute only a small fraction of the code, about 90% of the computation time is spent in them. Generally, particle pushers are not fully vectorizable. Considerable effort was spent to optimize the code for each machine. We believe that very little additional speedup of these routines can be achieved for either machine and that this test demonstrates the maximum performance that can be achieved for this type of code.

For the Cyber-205, PARMVE, PARMOV, and PARMVR were completely rewritten with vector syntax except for the subroutine CHARGE, which was written in assembly language (META). PARMVE calls CHARGE once; PARMOV and PARMVR each call it four times. CHARGE contains the major part of the nonvectorizable computation; it is entirely scalar. CHARGE executed for approximately 0.7 μs per particle; its FORTRAN version required about 2 μs per particle. Gather instructions cause the major slowdown in the rest of the code. There are no FORTRAN DO loops in the code. Within the three routines there are statements that can be replaced with direct calls to linked triads controlled by a bit vector. Because of limited Cyber-205 machine time and the instability of the Cyber-205 compiler, these modifications were not incorporated into the code. We believe these modifications would have only minor effects on the execution time. Cyber-205 timings for the PIC kernels are listed in Table XIV for several vector lengths.

Cray-1 timings are essentially independent of vector length. The versions of interest are

- CAL - a best effort Cray Assembly Language (CAL) version,
- Vector FORTRAN - a best effort vectorization in FORTRAN, and
- Scalar FORTRAN - the original version.

Cray-1 timings for the PIC kernels are given in Table XV.

The most meaningful comparison for this test can be made between the CAL version and the Cyber-205 version with vector length 2560, which gives the best possible timings in both cases. Table XVI shows the corresponding speed ratios.

Although the Cray-1 clearly outpaces the Cyber-205 on this highly optimized code, it is important to note that optimization for the Cray was achieved at high cost. Approximately 5 man-months were required for the Cray-1 version compared to 1.5 man-months for the Cyber-205 optimization, primarily because most of the Cyber-205 version was written in FORTRAN (with vector syntax). About 300 lines of META had to be written compared to 2500 lines of CAL. This is an important consideration for programmers who wish to achieve optimal performance for their codes.

5. CONCLUSIONS

The Cyber-205 is a very high-speed computing machine. It works especially well on problems that have long contiguous vectors. The raw speed of its vector box is considerably higher than that of the Cray-1. However, major drawbacks compared to the Cray-1 are its inability to access noncontiguous vectors efficiently and the longer cycle time of its scalar unit.

The Cyber-205 compiler needs additional work on automatic vectorization. The vector syntax is cumbersome and unique to this machine. It is possible, however, to use vector syntax and gain maximum performance at lower programming cost than with assembly language, the only option on the Cray-1. Although the Cray compiler is better at vectorization, the Cyber-205 compiler optimizes scalar code more effectively, not as well, however, as the FTN compiler marketed by Control Data Corporation for the CDC 7600.

Subroutine calls are expensive on both high-speed computers, which runs against current trends toward better modularity and structure. It would be highly desirable, therefore, to have compiler directives that place subroutines inline.

Our tests point up the problems with using raw maximum-result rates for high-speed vector computers as an indication of how real codes will perform. Many of our optimized codes ran 10 times slower than peak MFLOP rates for 64-bit arithmetic on the Cyber-205. Since portions of some codes cannot be vectorized, the speed of the scalar unit is an important factor for the overall performance of a machine. There is no substitute for actual experimentation and measurement.

Based on our test results, we conclude that in spite of the high speed of the Cyber-205 vector unit for contiguous vectors, the Cyber-205 does not appear to operate as rapidly as the Cray-1 on most codes that are representative of the Los Alamos Class VI Computer workload.

ACKNOWLEDGMENTS

The authors appreciate the assistance of many persons at the Control Data Benchmark Facility, Minneapolis, Minnesota. Special thanks to Dennis Kuba, George Vandegrift, and Robert Ayars of Control Data Corporation for their support in getting programs run on the Cyber-205. The efforts of Lawrence Rudsinski in optimizing the plasma code for the Cyber-205 were invaluable.

REFERENCES

1. A. H. Hayes and I. Y. Bucher, "Los Alamos Scientific Laboratory Computer Benchmark Performance 1979," Los Alamos Scientific Laboratory report LA-8689-MS (February 1981).

2. J. J. Dongarra, C. B. Moler, J. R. Bunch, G. W. Stewart, "LINPACK Users Guide," SIAM (1979).

3. W. P. Crowley, C. P. Hendrickson, T. E. Rudy, "The SIMPLE Code," Lawrence Livermore National Laboratory report UCID-17715 (February 1, 1978).

4. D. W. Forslund, C. W. Nielsen, and L. Rudsinski, "Vectorized PIC Simulation Codes on the Cray," Scientific Information Exchange Meeting, Lawrence Livermore National Laboratory, (September 12-13, 1979).

5. Control Data Corporation, Cyber 200 Model 205 Reference Manual, Publication number 60256020, (September 1980). Control Data Corporation, CDC Cyber 200 Fortran Language Reference Manual, Publication number 60457040, (August 1980).

6. P. N. Swartztrauber, National Center for Atmospheric Research, personal communication (June 1980).

7. D. Kuba, Control Data Corporation, personal communication (February 1981).

8. T. E. Rudy, Lawrence Livermore National Laboratory, personal communication (April 1981).

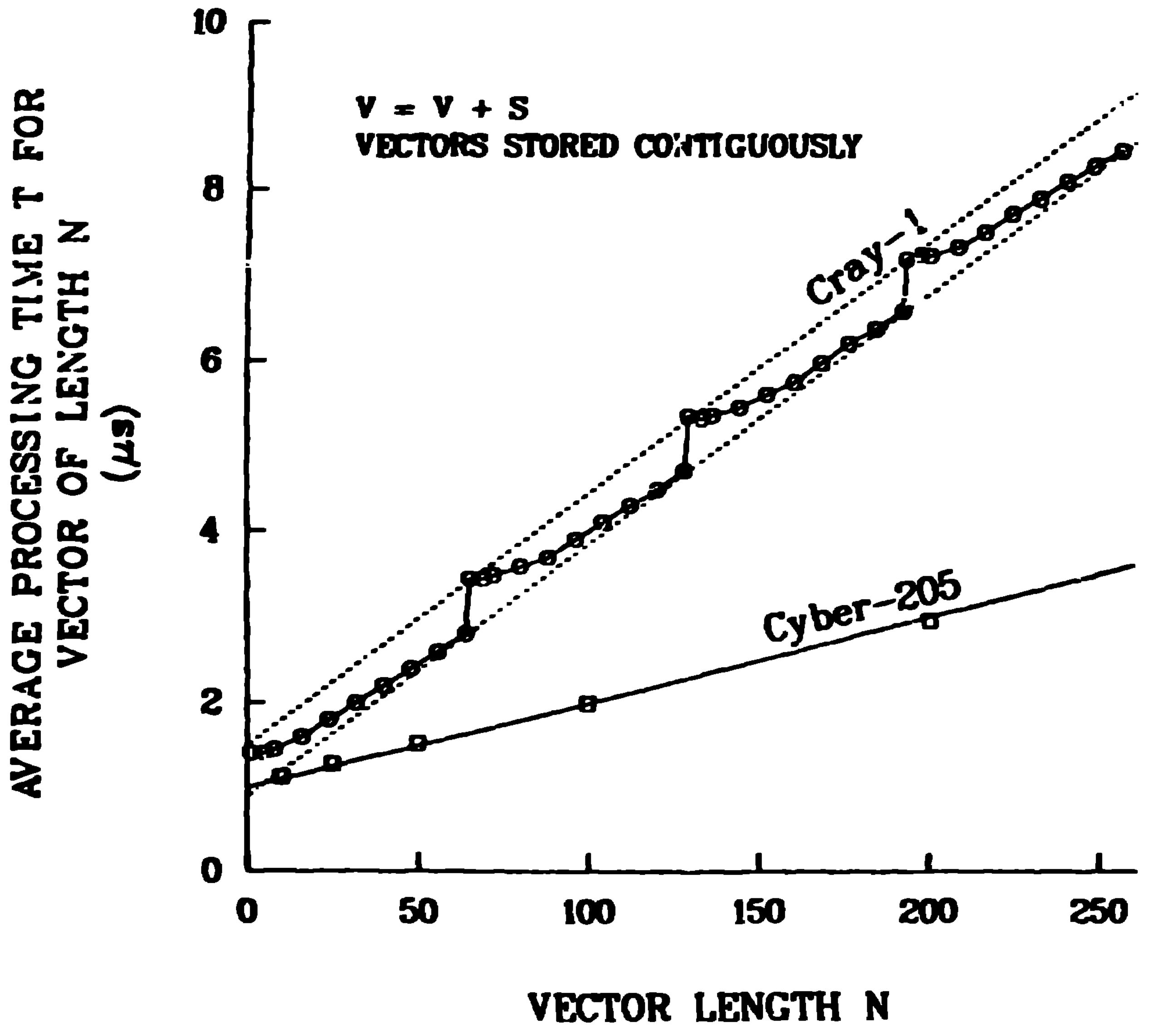

Fig. 1. Average Processing Time for Vector of Length N as a Function of Vector Length N.

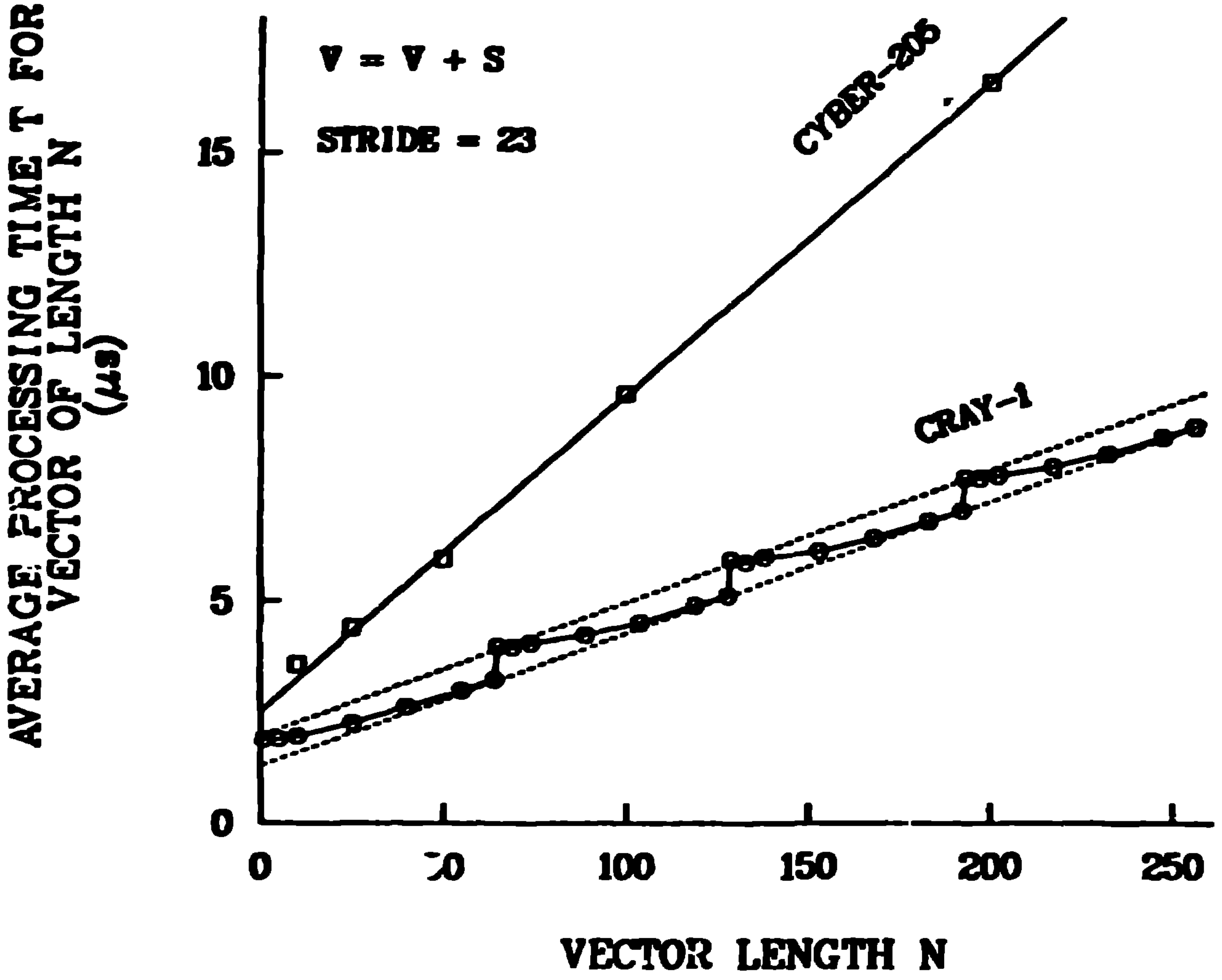

Fig. 2. Average Processing Time for Vector of Length N as a Function of Vector Length N.

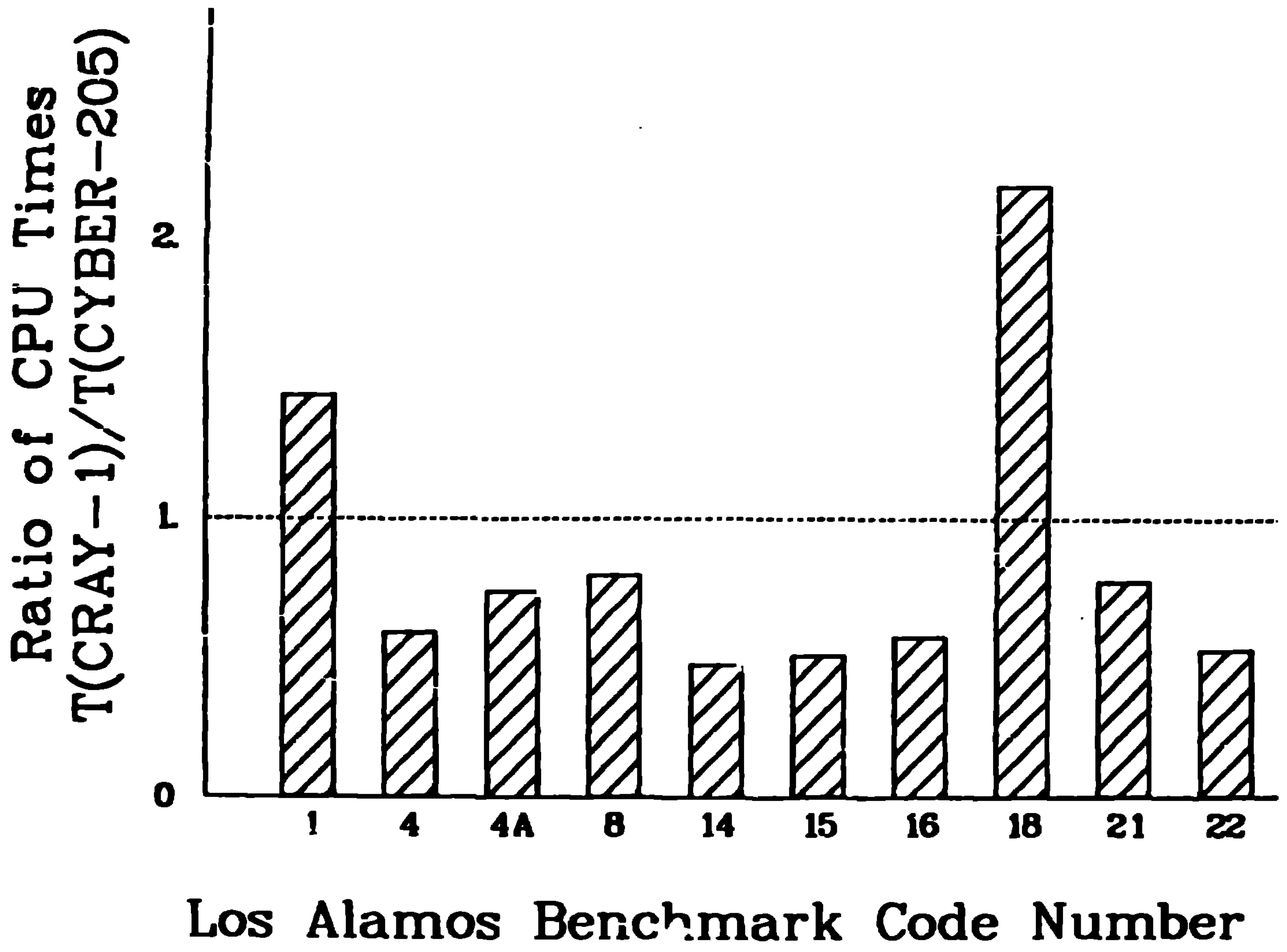

Fig. 3. Ratio of CPU Times T(Cray-1)/T(Cyber-205) for Los Alamos Benchmark Programs with Compiler Vectorization On.

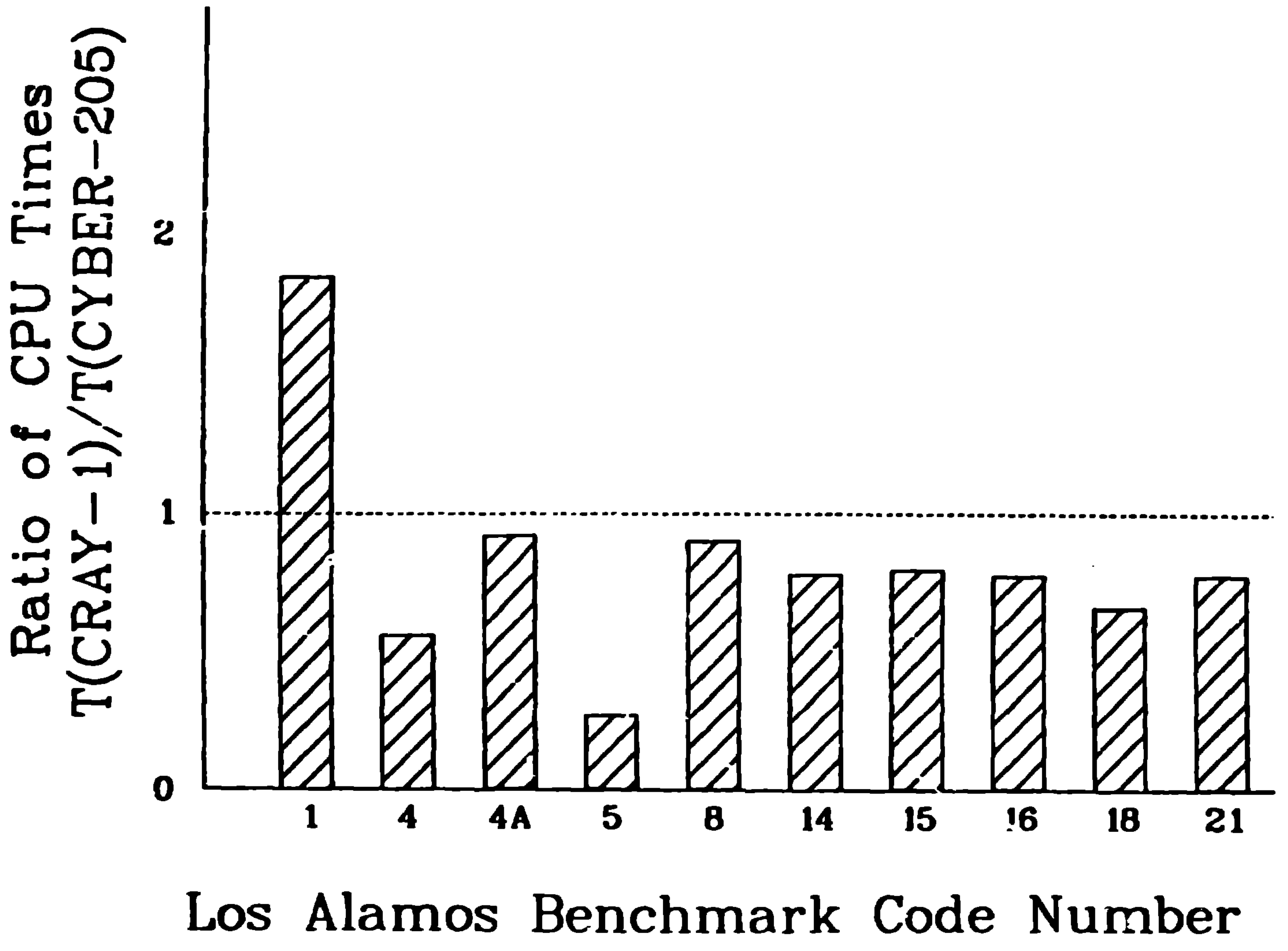

Fig. 4. Ratio of CPU Times T(Cray-1)/T(Cyber-205) for Los Alamos Benchmark Programs with Compiler Vectorization Off.

CYBER-205 VECTOR OPERATIONS
RATES FOR VECTORS STORED IN CONSECUTIVE LOCATIONS
(in MFLOPS)

Operation	Vector Length 10	25	50	100	200	500	1000	5000	10000	50000
V=V+S	8.93	19.53	32.89	50.00	67.57	83.33	91.24	98.04	99.01	99.80
V=S*V	9.26	20.16	33.78	50.00	68.49	83.33	91.58	98.14	99.01	99.80
V=V+V	8.93	19.53	32.89	50.00	67.57	83.26	91.24	98.04	99.01	99.80
V=V*V	9.26	20.16	33.78	50.00	68.49	83.33	91.58	98.14	99.01	99.80
V=V+S*V	11.90	27.17	48.07	78.09	112.30	152.32	172.86	193.80	196.85	199.20
V=V*V+S	11.90	27.17	48.07	78.09	112.30	152.32	172.86	193.80	196.85	199.20
V=V*V+V	9.26	20.16	33.78	49.99	67.54	83.30	91.09	98.03	99.01	99.80
V=S*V+S*V	10.42	22.86	41.66	63.55	91.44	119.38	132.92	146.20	148.00	148.66
V=V*V+V*V	9.15	19.95	33.48	50.67	66.95	83.68	91.44	98.04	99.04	99.73

V = vector
S = scalar

CRAY-1 VECTOR OPERATIONS
RATES FOR VECTORS STORED IN CONSECUTIVE LOCATIONS
(in MFLOPS)

	Vector Length									
Operation	10	25	50	100	200	500	1000	5000	10000	50000
V=V+S	6.72	13.51	20.20	25.00	27.63	31.95	32.95	33.81	33.99	34.11
V=S*V	6.78	13.61	20.30	25.00	27.54	31.79	32.76	33.60	33.76	33.88
V=V+V	6.11	11.43	16.00	18.78	20.18	22.45	22.92	23.33	23.42	23.47
V=V*V	6.11	11.43	16.00	18.73	20.10	22.36	22.82	23.23	23.31	23.36
V=V+S*V	11.11	19.70	26.40	30.25	32.06	35.01	35.61	36.12	36.23	36.29
V=V*V+S	11.19	21.39	30.53	35.79	38.46	43.08	43.99	44.80	44.97	45.08
V=V*V+V	10.19	17.32	22.47	25.16	26.38	28.41	28.80	29.13	29.20	29.24
V=S*V+S*V	16.67	31.91	45.63	53.57	57.48	64.58	65.97	67.18	67.45	67.62
V=V*V+V*V	14.20	25.00	32.88	36.47	38.06	41.15	41.67	42.12	42.23	42.30

V = vector
S = scalar

TABLE III

TIME REQUIRED TO PERFORM A VECTOR OPERATION ON THE CYBER-205

Operation	TSTART_VU (in ns)	T_EL (in ns)
add v:=v+s or v:=v+v	1020	10
mult v:=v*s or v:=v*V	980	10
triad v:=v+s*v or v:=s+v*v	1580	10

TABLE IV

CRAY-1, CFT COMPILER VERSION 1.09
TIME PARAMETERS FOR VECTOR OPERATIONS[a]
(in nanoseconds)

Operation	TSTART_OUT	TSTART_STRIP	T_EL
V=V+S	900	275	25.0
V=S*V	900	288	25.0
V=V+V	900	326	37.5
V=V*V	900	339	37.5
V=V	900	326	50.0 (min 37.5)
V=V*V+S	900	442	37.5
V=V*V+V	900	378	62.5 (min 50.0)
V=S*V+S*V	900	442	37.5
V=V*V+V*V	900	531	62.5

[a]See Eq. (2).

TABLE V

CYBER-205 VECTOR OPERATIONS
RATES FOR VECTORS STORED WITH A CONSTANT STRIDE[a]
(in MFLOPS)

Operation	Vector Length 10	25	50	100	200	500	1000
V=V+S	2.78	5.68	8.45	10.42	12.08	13.05	13.47
V=S*V	2.78	5.68	8.45	10.42	12.08	13.05	13.47
V=V+V	2.16	4.40	6.25	7.72	8.87	9.48	9.77
V=V*V	2.16	4.40	6.25	7.72	8.87	9.48	9.77
V=V+S*V	3.73	7.91	11.68	14.79	17.12	18.77	19.42
V=V*V+S	3.73	7.91	11.68	14.79	17.12	18.77	19.39
V=V*V+V	2.98	6.07	8.74	10.68	12.17	12.95	13.33
V=S*V+S*V	4.63	9.97	14.76	19.33	22.94	25.29	26.30
V=V*V+V*V	3.41	6.94	10.03	12.33	14.18	15.30	15.79

[a]Stride = 49

TABLE VI

CRAY-1 VECTOR OPERATIONS
RATES FOR VECTORS STORED WITH A CONSTANT STRIDE[a]
(in MFLOPS)

Operation	Vector Length 10	25	50	100	200	500	1000
V=V+S	5.10	11.11	17.39	22.73	25.93	31.15	32.52
V=S*V	5.03	11.05	17.32	22.60	25.72	30.96	32.31
V=V+V	4.73	9.66	14.18	17.47	19.25	22.05	22.71
V=V*V	4.65	9.52	14.03	17.32	19.11	21.93	22.60
V=V+S*V	8.84	17.09	23.95	28.57	30.92	34.54	35.37
V=V*V+S	8.87	18.35	27.30	33.47	36.82	42.37	43.29
V=V*V+V	8.29	15.33	20.72	24.02	25.94	28.11	29.24
V=S*V+S*V	13.04	27.03	40.40	49.24	54.19	63.42	65.36
V=V*V+V*V	11.54	21.74	29.92	34.58	36.60	40.65	41.41

[a]Stride = 49

CYBER-205 VECTOR OPERATIONS INVOLVING RANDOM GATHERS AND SCATTERS
(in MFLOPS)

	Vector Length									
Operation	10	25	50	100	200	500	1000	5000	10000	50000
V=V(IND)+S	2.19	4.73	7.44	10.50	13.23	15.28	16.62	17.71	17.81	17.87
V(IND)=V*V	2.50	5.30	8.56	11.90	14.37	16.49	17.27	17.81	18.12	18.29
V(IND)=V(IND)+V*V	2.94	6.19	9.69	13.30	15.82	17.88	18.81	19.50	19.74	19.90
V=V+V*V(IND)	3.52	7.62	11.90	17.24	22.03	25.72	27.65	29.80	30.24	30.33

V = vector
S = scalar
IND = index vector of random integers

TABLE VIII

CRAY-1 VECTOR OPERATIONS INVOLVING RANDOM GATHERS AND SCATTERS
(in MFLOPS)

Operation	Vector Length									
	10	25	50	100	200	500	1000	5000	10000	50000
V=V(IND)+S	1.79	1.84	1.90	1.92	1.94	1.95	1.95	1.95	1.95	1.95
V(IND)=V*V	2.19	2.41	2.49	2.53	2.56	2.57	2.58	2.58	2.58	2.58
V(IND)=V(IND)+V*V	3.43	3.70	3.80	3.85	3.88	3.89	3.90	3.90	3.90	3.90
V=V+V*V(IND)	2.93	3.12	3.19	3.23	3.25	3.25	3.25	3.26	3.26	3.26

V = vector
S = scalar
IND = index vector of random integers

TABLE IX

CYBER-203, CRAY-1 SCALAR OPERATIONS
RATES FOR OPERANDS STORED IN CONSECUTIVE LOCATIONS
(in MFLOPS, Vector Length N = 50 000)

Operation	Cyber-203 BENCHFTN Compiler Not Optimized	Cyber-203 BENCHFTN Compiler Optimized	Cray-1 CFT, Version 1.08 Not Optimized	Cray-1 CFT, Version 1.08 Optimized
V=V+S	1.47	3.12	2.29	2.86
V=V*S	1.47	3.12	2.22	2.76
V=V+V	1.43	2.94	2.16	2.67
V=V*V	1.43	2.94	2.10	2.57
V=V+S*V	2.00	5.55	3.40	4.57
V=V*V+S	2.50	5.88	3.64	4.32
V=V*V+V	1.96	5.26	3.27	4.32
V=S*V+S*V	2.73	7.89	4.44	6.48
V=V*V+V*V	2.63	7.89	4.14	5.85

TABLE X

EXECUTION TIMES FOR STANDARD BENCHMARK PROGRAMS
CRAY-1 AND CYBER-205
(CPU time in seconds)

Program Number	Description	Cray-1 Nonvectorized	Cray-1 Vectorized	Cyber-203 Nonvectorized	Cyber-205 Vectorized
1	Integer Monte Carlo	75.1	58.2	40.9	40.7
4	Fast Fourier Transform (old)	30.5	21.8	54.1	37.2
4a	Fast Fourier Transform [6]	11.3	6.4	12.4	8.7
5	Equation of State Kernel	28.2	29.2	103.2	--
8	Vector Operations	36.9	19.0	41.3	23.9
11	Plasma Simulation Kernels [4]	177.3	104.2	a	a
14	Matrix Operations	15.9	3.0	20.5	6.2
15	Linear System Solver (small)	65.3	16.3	82.3	31.7
16	Linear System Solver (800x800)	298.5	36.3	386.9	62.9
18	Vector Operations	20.3	2.4	30.6	1.1
21	Radiation Transport Monte Carlo	4.8	4.8	6.2	6.2
22	Linear System Solver (new)	70.2	16.4	--	31.2

[a]Program 11 was not run due to compiler problems.

TABLE XI

LINEAR SYSTEM SOLVER FORTRAN VERSIONS
EXECUTION TIMES FOR 300 SOLUTIONS
(in seconds)

System Size	Cyber-205		Cray-1		Cyber/Cray Speed Ratio	
N X N	LSS2	LSS5	LSS2	LSS5	LSS2	LSS5
100 X 100	13.0	7.3	15.8	11.0	1.2	1.5
200 X 200	56.3	31.6	82.8	65.0	1.5	2.1
400 X 400	255.0	156.0	511.0	434.0	2.0	2.8
800 X 800	1270.0	882.0	3449.0	3075.0	2.7	3.5

TABLE XII

LINEAR SYSTEM SOLVER LIBRARY ROUTINES
EXECUTION TIMES FOR 300 SOLUTIONS
(in seconds)

System Size N X N	Cyber-205	Cray-1	Cyber-205/Cray-1 Speed Ratio
100 X 100	5.9	2.9	0.48
200 X 200	26.5	16.8	0.66
400 X 400	135.7	118.5	0.87
800 X 800	784.5	899.1	1.14

TABLE XIII

SIMPLE EXECUTION TIMES
(in seconds)

Mesh Size	Cyber-205		Cray-1	Cyber-205/Cray-1 Speed Ratio
N X N	SIMP	SIMP4	SIMP	SIMP4/SIMP
32 X 32	8.2	6.3	3.2	0.50
64 X 64	31.8	22.7	12.0	0.53
96 X 96	70.6	49.0	27.1	0.56

TABLE XIV

CYBER-205 TIMINGS FOR PIC KERNELS
(in microseconds/particle)

Vector Length	PARMVE	PARMOV	PARMVR
2560	2.33	7.53	9.02
640	2.49	7.89	9.46
160	2.83	8.36	10.02
64	4.06	11.54	13.95

TABLE XV

CRAY-1 TIMINGS FOR PARTICLE-IN-CELL KERNELS
(in microseconds/particle)

	PARMVE	PARMOV	PARMVR
CAL	1.74	4.35	4.77
Vector Fortran	5.5	--	11.00
Scalar	8.0	--	26.00

TABLE XVI

RATIO OF CYBER-205 TO CRAY-1 SPEED
FOR OPTIMIZED PIC KERNELS

PARMVE	PARMOV	PARMVR
0.75	0.58	0.53