Benchmarks

DTN → RNI

June 1989 C3P 712

Benchmarking Advanced Architecture Computers

Paul Messina[1], Arnold Alagar[2], Clive Baillie[1], Edward Felten[1], Paul Hipes[1], Anke Kamrath[2], Robert Leary[2], Wayne Pfeiffer[2], Jack Rogers[2], David Walker[3], Roy Williams[1]

[1] Caltech Concurrent Supercomputing Facility,
206-49 California Institute of Technology,
Pasadena, CA 91125

[2] San Diego Supercomputing Center,
P. O. Box 85608,
San Diego, CA 92138

[3] Department of Mathematics,
University of South Carolina,
Columbia, SC 29201

Abstract: Recently a number of advanced architecture machines have become commercially available. These new machines promise better cost-performance than traditional computers, and some of them have the potential of competing with current supercomputers, such as the CRAY X-MP, in terms of maximum performance. This paper describes the methodology and results of a pilot study of the performance of a broad range of advanced architecture computers using a number of complete scientific application programs. The computers evaluated include (1) shared-memory bus architecture machines such as the Alliant FX/8, the Encore Multimax, and the Sequent Balance and Symmetry, (2) shared-memory network-connected machines such as the Butterfly, (3) distributed-memory machines such as the NCUBE, Intel and Jet Propulsion Laboratory (JPL)/Caltech hypercubes, (4) very long instruction word machines such as the Cydrome Cydra-5, (5) SIMD machines such as the Connection Machine, and (6) "traditional" supercomputers such as the CRAY X-MP, CRAY-2, and SCS-40. Seven application codes from a number of scientific disciplines have been used in the study, although not all the codes were run on every machine. The methodology and guidelines for establishing a standard set of benchmark programs for advanced architecture computers are discussed. The CRAYs offer the best performance on the benchmark suite; the shared memory multiprocessor machines generally permited some parallelism and when coupled with substantial floating point capabilities (as in the Alliant FX/8 and Sequent Symmetry), provided an order of magnitude less speed than the CRAYs. Likewise, the early generation hypercubes studied here generally ran slower than the CRAYs, but permitted substantial parallelism from each of the application codes.

1. Introduction

This paper presents the results and conclusions of the Caltech Performance Evaluation Project (CPEP), a one-year pilot project funded by the National Science Foundation, to evaluate the performance of scientific programs on advanced architecture computers. This type of study is necessary because of the steady increase in the past few years in the number of commercially-available advanced architecture computers, which promise to provide high performance at a unit cost substantially lower than "traditional" (Von Neumann) computers, and which have the potential, in some cases, of competing with current supercomputers in terms of maximum performance. As pointed out in [NAS:86a], theoretical models and performance measures have been developed to explain and predict the performance of sequential architecture machines. However, similar tools have yet to be developed for advanced architecture machines, and this task is made harder by the complexity and diversity of the different types of newly-emerging architectures [Martin:87a].

The primary objective of this study is to investigate the performance of complete scientific and engineering codes on a range of advanced architecture computers. This will allow us to:

(1) Assess the suitability of a given machine for a class of applications;
(2) Provide input for improving the design of future advanced architecture machines;
(3) Develop a methodology for future large-scale performance evaluation studies.

Our intention is not to quantify the performance of each machine by means of a single all-encompassing number, since such a measure of performance gives no indication of how the different programs are affected by factors such as architecture, compilers, and I/O

speed on the different machines. Instead we present results separately for each application on each of the target computers.

Many of the results presented in this paper to quantify performance may be of only ephemeral value since new machines are constantly appearing and often render earlier machines obsolete. In addition, the performance of some of the machines studied may change in the future as hardware or software enhancements are made. Thus, in addition to the actual performance measurements made, an important component of this work is the establishment of a well-documented set of bug-free benchmark programs and a methodology for evaluating the performance of the target machines.

In order to make certain that the measurements made are valid the project involved both experts in the applications, to ensure that algorithms were implemented correctly and that timing runs were representative of important problems, and experts in the target computing environments, to ensure that hardware features were understood and exploited, and that system software and compilers were used properly.

The project was conducted in four stages. In the first stage, two of the applications (QCD and TRACKER) were run on as many of the selected machines as possible. This allowed us to learn the simple mechanics of getting codes to run on different machines. In the second stage a selection of the application codes was run on the machines to which we have easy access. These machines were the Intel iPSC, NCUBE/10, and Mark III hypercubes at Caltech, the CRAY X-MP and SCS-40 vector computers at the San Diego Supercomputer Center (SDSC), the CRAY-2 at the Air Force Supercomputer Center-Kirtland (AFSCC-K), and the Sequent Balance and Symmetry, Alliant FX/8, Encore shared-memory computers and Cydrome Cydra 5 very-long-instruction-word computer at Argonne National Laboratory (ANL). Since there were 7 codes and 11 machines, limitations of time and personnel prevented all the codes being run on each machine; however, several of the codes were run on most machines. Wherever possible both C and FORTRAN versions of the codes were used. In the next stage the application codes were run on new machines of interest to which we gained access in the course of the project, such the ETA10-E at the Florida State University and the Connection Machine at ANL. The final stage of the project involved the analysis of the timing measurements and the publication of the results.

The source code and I/O files for the applications in the CPEP benchmarking suite have been placed in the public domain and may be accessed electronically as described in Appendix B. We encourage people to use these benchmarks to evaluate the performance of new machines as they become available, and any existing machines of interest not considered in this work.

2. Performance Measures

Before proceeding further it is necessary to clarify the meaning of "performance" in the context of this work. In a general sense the performance could be defined as the value (to some organization) of the output of the computer normalized for total cost. In this case, it is clear that the performance of a computer is dependent on the context in which it is used. A number of cost factors, such as the cost of the hardware, maintenance, software upgrades, operating personnel and so on, must be included in estimating the input cost, together with less easily quantifiable factors such as ease of programming and debugging.

From this perspective the performance is a measure of the cost of useful output, and is typically of use to computer centers and commercial organizations in determining what sort of machine most suits their needs.

An individual with access to a number of different machines would usually measure performance by other criteria. Rather than being concerned with the total cost and system throughput in deciding which machine to use for a particular application code, he/she is more likely to look at factors such as how fast his/her applications can be run (job turnaround), how much memory is available, the cost for his/her type of job, and how user-friendly the software environment is. In our performance evaluation project we view performance more from the perspective of the individual user, and regard CPU time as the main indicator of performance. To be precise, CPU time is defined to be the length of time that the CPUs are working on the user's program. A complimentary time is the elapsed time, from the activation of the executable to its termination, measured in real time. The CPU and elapsed times were very similar for some of the target machines, such as the hypercubes, which essentially run in single-user interactive mode. Since we have chosen to equate CPU time with performance we would like to emphasize that the results of this project should not be taken to be indicative of overall system performance, particularly when there are many users or when programs are run in batch mode.

For sequential computers a commonly-used measure of speed is the peak rate of the processing hardware as measured in Mflops (Millions of Floating Point Operations Per Second) or Mips (Millions of Instructions Per Second). This is not a representative measure of the performance of most application codes for a number of reasons. In many cases it is impossible to achieve the peak rate for a real program, either because the computation does not involve a mix of operations that keep all the functional units busy all the time, or because of inherent bottlenecks, typically in memory access. Unless the user is prepared to hand-code carefully the application, inefficiencies generated by the compiler will also reduce the speed below the peak rate. In addition, an application may include many integer operations and/or conditional branching, in which case a speed quoted in Mflops is not a true indicator of performance.

Kernels and isolated routines, e.g., Livermore loops and Basic Linear Algebra Subprograms (BLAS), are more useful than the peak rate as performance indicators; their performance has been shown to correlate well with that of a representative workload. However, they may still not be representative of all types of full production codes. In particular, they do not take into account the cost of performing I/O and may mask memory constraints which are only readily apparent in full codes.

The bulk of the codes used in our performance evaluation project are what might be termed "university research codes". These programs are usually up to a few thousand lines long, and typically might be produced by a graduate student for his/her thesis. Such codes are capable of producing journal-quality scientific results. These types of applications have been selected because they constitute a significant fraction of the codes currently being run on advanced architecture computers and are readily available to us. Also these types of code are of sufficient length and complexity to exercise most aspects of the hardware and software environments of the machines to be investigated but are still small enough to be understood and ported from one computer to another.

The codes in the benchmarking suite all have successful hypercube implementations and were chosen, in part, for that reason. This is a source of bias in our study and it is not our intention to obscure this bias. Nevertheless, the existence of viable hypercube implementations of a spectrum of codes allows a meaningful comparison between hypercube codes and similar codes for the same application for other architectures. It is easy to construct a sequential code for an application that has an existing hypercube implementation; it is more difficult to construct a hypercube code when the sequential code exists because the current hypercube programs require explicit message-passing procedures. In the future, distributed memory codes will be easier to build and more generic; comparisions between very different architectures will be less ambiguous. For the present, studies such as this one fill a void that currently exists between the newly emerging computer architectures/environments (hypercubes), more well developed architectures/environments (shared-memory), and the very well developed vector supercomputers.

We refer to programs of more than about 10,000 lines as "production" or "engineering" codes, e.g., NASTRAN, SPICE, weather prediction codes, etc. These types of codes are usually used in large corporate research groups and in National Laboratories. In general they are difficult for an individual to understand in their entirety, and hence cannot be easily ported to certain types of machines such as distributed memory multiprocessors. For this reason our project will not use this type of code in evaluating performance. However the performance of such codes is of great interest and is being investigated in a collaboration headed by Prof. David Kuck of the University of Illinois [Berry:88a]. Our performance evaluation project complements their work.

3. Methodology

The performance of a particular computer depends on a number of factors, including:

- the nature of the application,
- the sophistication of the compiler,
- the operating system,
- whether the code is run in a dedicated, or in a multi-user, environment.

When running an application a careful record must therefore be kept of the conditions under which the code was run. In particular, the name and version number of the operating system, the version of the compiler used, and the setting of compiler switches should be noted. When the setting of certain compiler switches has an impact upon performance, such as on the Alliant FX/8 for example, runs should be made for the differing switch settings. Where possible both C and FORTRAN versions of the codes should be run. This gives some insight into the relative efficiency of the different compilers.

When porting an application from one computer to another, the amount of modification applied to the code can have an important impact on the performance of the code. The minimum amount of modification necessary is that required to make the code run. Further changes might include the re-writing of inner loops in assembly language, the complete customization of the code, or the re-structuring of the algorithm.

In order to control and limit the changes to a code when porting it to a new machine, the following procedure was adopted:

(1) The expert for each application produces a "vanilla" sequential version of the code. As pointed out in [Dongarra:87b] in their informative discussion of the potential pitfalls

in evaluating computer performance, there is really no such thing as a vanilla code since programming styles and algorithms are usually designed to take advantage of the architectural features of the machine for which they are written. However, an effort was made to ensure that our vanilla codes did not contain any blatant machine-specific constructs or constraints. This code is distributed to the experts for each machine who then time the code, having made the minimum changes necessary to make the code run.

(2) The code is again timed after introducing minor changes, such as using native functions, random number generators, and so on. All these modifications are documented.

(3) Next the code is timed after introducing minor restructuring and customization aimed at getting higher speed through better use of the hardware and software environments. In order to limit the extent of these changes we require that they take no more than one week to implement.

It might be argued that putting a constraint of one week on the time allowed for code optimization is arbitrary, and also does not give an accurate indication of the best possible performance achievable for a given program. This is certainly a valid point; however, resource limitations precluded spending an indefinite length of time customizing each code on each machine to achieve the optimum performance. On the other hand, the extent of the changes listed in (1), (2) and (3) above are intended to reflect the type of optimization made by the typical university programmer of reasonable experience and competence, and so may represent what happens in practice. Our view is that any amount of optimization of the code is permissible, provided that the modifications made to the original sequential code are carefully documented, and that a record of the time and effort taken to make these changes is kept.

4. The Application Programs

As mentioned in the previous section, we are evaluating the performance of *scientific* programs. These are programs typically of interest to university research groups, and range in length from a few hundred to a few thousand lines. The programs selected for inclusion in the benchmark suite are real programs capable of producing scientifically interesting results. Most of the programs have been written by researchers at Caltech and JPL, and we believe they are representative of the workload typically encountered on university research computers. A wide range of algorithms and applications is covered, drawn from the fields of physics (QCD), computational fluid dynamics (vortex dynamics), chemistry (chemical reaction dynamics), signal processing (target tracking), and others. Both CPU and I/O intensive programs are included in the problem set and where resources have permitted both C and FORTRAN versions of the codes have been run. In addition, two important linear algebra programs have been included. Double precision (64-bit) real arithmetic is used in all applications unless otherwise indicated.

4.1 Linear Algebra Algorithms

As mentioned in Section 1, most of the programs used in this project are complete scientific programs of research interest. However, two important linear algebra algorithms commonly used in application codes have been included: matrix multiplication and the

solution of banded systems by LU decomposition, followed by forward reduction and back substitution. LU decomposition of full 100 by 100 matrices has been used in previous performance evaluation work [Dongarra:88b] in which the performance of this LINPACK software was evaluated on about 100 computers. Thus the inclusion of banded LU routines in the present benchmarking suite complements other work. In addition, many scientific applications are dominated by linear algebra routines so the performance of these routines is of general interest. In the future we expect additional linear algebra routines to be added to the suite.

The "vanilla" matrix multiplication benchmark program is straightforward and its implementation on distributed MIMD machines containing a toroidal interconnection topology has been discussed by [Fox:85b] and [Fox:88a].

The LU decomposition code is rather more complicated, and optionally performs partial pivoting. Furthermore the forward reduction and back substitution steps allow for multiple right-hand sides. Thus we seek the solution of n_b banded systems of linear equations, and can express the problem as,

$$AX = B \tag{1}$$

where A is a banded $M \times M$ matrix, and X and B are $M \times n_b$ matrices. Each column of B gives a separate right-hand side, and the corresponding column of X is the solution vector. The solution of Eq. (1) takes place in three stages:

(a) The banded matrix A is factorized as $A = LU$ where L is a lower triangular matix with 1's on the main diagonal, and U is an upper triangular matrix. The matrices L and U overwrite the matrix A, so that at the end of this stage the upper triangular part of A contains the matrix U, and the lower triangular part of A contains the matrix L, except for the diagonal which need not be explicitly stored (since it is all 1's).

(b) Next the forward reduction is performed. This is formally equivalent to multiplying both sides of $AX = B$ by L^{-1} to give,

$$UX = L^{-1}B = B_{FR} \tag{2}$$

This stages involves the modification of the matrix B to B_{FR}.

(c) The last stage in the solution is back substitution, in which the solution matrix, X, is obtained by multiplying Eq. (2) by U^{-1},

$$X = U^{-1}B_{FR} \tag{3}$$

As it is evaluated, the solution matrix X overwrites the storage for B_{FR}. Thus in the forward reduction and back substitution steps the matrix B is first modified to B_{FR} and then to X. Sequential and parallel implementations for distributed memory multiprocessors are discussed in [Fox:88a] and [Aldcroft:88a].

4.2 Quantum Chromodynamics

Quantum Chromodynamics (QCD) is the gauge theory of the strong interaction which binds quarks and gluons into hadrons, which in turn make up the constituents of nuclear

matter. Analytical perturbation methods can be applied to QCD only at small distances (or equivalently at high energies), hence computer simulations are necessary to study long-range effects in QCD theory (i.e., at lower energies). In the pure-gauge model, the only contribution to the action comes from the gauge field so the action is strictly local. The inclusion of dynamical fermions in the model gives rise to a non-local action which complicates the algorithm considerably. However, in the QCD code used in this project, the effects of dynamical fermions are ignored. The code therefore represents a pure-gauge model in the "quenched" approximation. A description of the QCD program, together with a discussion of its performance on the Mark III and NCUBE, is given in [Flower:88b], [Walker:88c].

In lattice gauge theory simulations, the quantum field is discretized onto a periodic, four-dimensional, space-time lattice. Quarks are located at the lattice sites, and the gluons that bind them are associated with the lattice links. The gluons are represented by SU(3) matrices, which are a particular type of 3×3 complex matrix. A major component of the QCD code involves updating these matrices. A number of different methods have been proposed for updating the SU(3) matrices (see [Otto:87a] for a summary). The QCD benchmark uses the Cabbibo-Marinari pseudo heat-bath algorithm [Cabbibo:82a] to update the SU(3) matrices on the lattice links. This algorithm uses a Monte Carlo technique to generate a chain of configurations (set of values for the entries of the SU(3) matrices for all links of the lattice) which are distributed with a probability proportional to $\exp(-\beta S(U))$, where $S(U)$ is the action of configuration U, and β is inversely proportional to the square of the coupling constant. Two basic computations are involved in updating the lattice; multiplication of SU(3) matrices to calculate the local contribution to the action, and the generation of pseudo-random numbers with the desired distribution. It is the matrix multiplication which dominates the execution time.

Lattice QCD problems are among the most computationally-intensive large-scale scientific computations. In fact, it was the need for a cost-effective means of seriously addressing lattice QCD computations that led to the development of the early hypercubes at Caltech in 1981-3. The QCD code has its origins in the work of Otto and Flower at Caltech. Their parallel pure gauge QCD code has been run for 1000s of hours and generated a substantial amount of science: [Otto:83b], [Brooks:84a], [Otto:84a], [Otto:85b], [Flower:85a], and [Flower:87b] In 1987-8, sequential and parallel FORTRAN versions of the code were developed and benchmarked [Baillie:88g].

4.3 Target Tracking

The TRACKER code was developed at Caltech by Gottschalk and co-workers to determine the course of a set of an unknown number of targets, such as rocket boosters, from observations of the targets taken by sensors at regular time intervals (see [Gottschalk:88a] and references therein). The targets may be launched from a number of different sites. Gottschalk's code was originally written in C for the Caltech/JPL Mark II and III hypercubes, and has also been ported to the iPSC/1 and NCUBE hypercubes [Baillie:88f] and shared-memory machines [Cao:88a]. The sequential FORTRAN version used for benchmarking was written by Gottschalk at the start of 1988.

If the target's acceleration is assumed to be known, the path of an individual object is described fully by a 4-component launch vector made up of the latitude and longitude of

the launch site, the time of launch, and the initial launch azimuth relative to due north. At each time step a simple linear Kalman filter is first used to estimate the position, velocity and acceleration of the targets from the noise-corrupted sensor data, using an underlying kinematic model with a stochastic acceleration component. The output from this phase is then passed to the precision parameter estimation module that uses Newton-Raphson iteration to solve an equation to give a more precise estimate of the launch parameter vector.

In a multi-target scenario, sensor data points are associated with tracks by means of the "track splitting" algorithm described in [Gottschalk:87c]. In general this association will not be unique, and a single sensor point may at some stage of processing be associated with more than one track. This is particularly true at the early stages of processing when the number of possible valid tracks may be large. The problem of a potential combinatoric explosion in the number of valid tracks is dealt with by a track pruning algorithm which discards the poorer of any duplicate tracks or a track which is incompatible with sensor data. The initialization of new tracks is managed by a separate module called the "batch mode initializer", which limits possible new tracks to a reasonably plausible set.

4.4 Chemical Reaction Dynamics

This application studies the quantum mechanical reactive scattering of an atom and a diatomic molecule by calculating the scattering matrix as a function of energy. In the past, lack of convergence and other numerical problems in the solution of the Schrödinger equation have hindered these types of *ab initio* quantum-chemical calculations, even for very simple systems. A recent significant advance in chemical dynamics is the use of symmetrized hyperspherical coordinates [Kuppermann:75a], coupled with generous allocations of CRAY resources, in the solution of reactive scattering problems. Symmetrized hyperspherical coordinates (SHC) are used to specify the geometrical configuration of triatomic systems in terms of the hyperspherical radius, ρ, and a set of five hyperspherical angles collectively denoted by ω. The use of SHC avoids the numerical problems inherent in the earlier methods.

As explained in [Hipes:88b], the SHC method first expands the scattering wave function in terms of a separable basis set, namely the local hyperspherical surface functions (LHSF), $\Phi_n^{J,M}(\omega;\rho)$, which are defined to be the simultaneous eigenfunctions of the angular part of the Hamiltonian. Thus, for ρ in some neighborhood of $\overline{\rho}_j$ we write

$$\Psi^{J,M}(\rho,\omega) = \rho^{-5/2} \sum_{n'=1}^{N} f_{n'}^{J}(\rho;\overline{\rho}_j)\Phi_{n'}^{J,M}(\omega;\overline{\rho}_j) \tag{4}$$

This expansion is then substituted into the Schrödinger equation to yield a set of coupled ordinary differential equations known as the coupled channel Schrödinger equation, from which the coefficients $f_{n'}^{J}(\rho;\overline{\rho}_j)$ can be derived,

$$\sum_{n'=1}^{N} \left[\delta_{n'}^{n} \frac{d^2}{d^2\rho} + V_{n,n'}^{J}(\rho,\overline{\rho}_j) \right] f_{n'}^{J}(\rho;\overline{\rho}_j) = 0 \tag{5}$$

where $\delta_{n'}^{n}$ is the Kronecker delta function, and V^J is the interaction matrix, which depends upon the total energy and integrals over pairs of LHSFs mutliplied by the potential energy. The potential energy induces couplings between surface functions.

The system of coupled ODEs in Eq. (5) is solved using the logarithmic derivative integrator of [Johnson:73a] [Johnson:77a]. This fourth-order integrator is based on the Ricatti form of the coupled channel Schrödinger equation. In addition to having good stability, the logarithmic derivative method has the additional advantage of being straightforward to implement, since the principal algorithm involved is matrix inversion.

A different set of LHSFs is needed for each neighborhood, $\overline{\rho}_j$. Thus it is necessary to be able to transform coefficients in Eq. (4) for one set of LHSFs to those corresponding to the LHSFs of an adjacent neighborhood. The main operation involved in such transformations is matrix multiplication. In summary, the implementation of the logarithmic derivative method requires matrix inversion and matrix multiplication algorithms.

The complete reactive scattering problem is performed in three stages. First the LSHFs are determined, then the interaction matrices are calculated using numerical quadrature, and lastly the coupled Schrödinger equation is solved by the logarithmic derivative method. The benchmark program used in the Caltech Performance Evaluation Project performs the last of these stages. The input to the program is the set of interaction matrices, which have previously been calculated on a CRAY X-MP. 65 terms are used in the expansion of the scattering wave function and the input set contains several hundred 67 by 67 matrices, resulting in an input file of approximately 24 MB. The matrix inversion required in the logarithmic derivative method is performed using the Gauss-Jordan algorithm; this dominates the calculation time.

The parallel implementation of the logarithmic derivative method on the hypercube is discussed in [Hipes:88b] and the matrix inversion is performed using the parallel Gauss-Jordan algorithm described in [Hipes:88a]. A local rectangular sub-block decomposition is used for both the Gauss-Jordan and matrix multiplication algorithms. On the hypercubes it was found that I/O dominated the run time due to the large size of the input data file.

The problem addressed in the benchmark program is the reactive scattering in three-dimensional space of $H + H_2 \rightarrow H_2 + H$. In the original calculation, the number of neighborhoods used is 100 and the calculation is performed for 1000 ρ steps. The CRAY X-MP/48, CRAY-2, and SCS-40 runs used this data set. Because the large input data set is cumbersome for some facilities (e.g. because of disk space limitations), the number of neighborhoods in the data set has been reduced to 30, corresponding to the 24 MB file refered to in the last paragraph. The times reported in this publication are scaled, when necessary, to the CRAY standard of 100 neighborhoods.

4.5 Fluid Dynamics Using the Vortex Method

The vortex method is used to model the flow of incompressible, inviscid fluids. The method is particularly useful for studying turbulent flows at high Reynolds number, since such flows often are characterized by regions of concentrated vorticity (eddies) embedded in an otherwise irrotational fluid. Applications of the vortex method include the simulation of the flow past bluff bodies, helicopter blades, and stalled airfoils [Spalert:83a] [Spalert:84a], and the interaction of colliding smoke rings [Leonard:88a]. Reviews of the method may be found in [Leonard:80a], [Leonard:85a] and [Aref:83a].

The vortex method rests on the fact that in an inviscid flow the fluid vorticity moves with the local velocity. If the flow is incompressible, the velocity field can be determined from the vorticity distribution. In other words, the incompressible Navier-Stokes equations can be replaced by the vorticity equation, which in two dimensions is

$$\frac{\partial \boldsymbol{\omega}}{\partial t} + (\mathbf{u} \cdot \nabla)\boldsymbol{\omega} = 0 \tag{6}$$

Since the vorticity is defined as $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ the velocity field can be determined from the Poisson equation

$$\nabla^2 \mathbf{u} = -\nabla \times \boldsymbol{\omega} \tag{7}$$

Writing the solution to Eq. (7) in terms of the Biot-Savart integral we have

$$\mathbf{u}(\mathbf{r}, t) = -\frac{1}{2\pi} \int \frac{(\mathbf{r} - \mathbf{r}') \times \mathbf{k}}{|\mathbf{r} - \mathbf{r}'|^2} \boldsymbol{\omega}(\mathbf{r}', t) d\mathbf{r}' \tag{8}$$

where $\mathbf{k}$ is the unit vector in the z direction.

The vorticity field is represented by a static set of N vortices where the vortices are spread out so that each has a finite core size [Chorin:73a]. Thus in the "vortex blob method" the vorticity field is represented by

$$\boldsymbol{\omega}(\mathbf{r}, t) = \sum_{j=1}^{N} \Gamma_j \gamma_j \Big(\mathbf{r} - \mathbf{r}_j(t) \Big) \tag{9}$$

where γ_j is the vorticity distribution of the vortex centered at $\mathbf{r}_j$ and Γ_j is the circulation of the jth vortex. In the standard blob vortex method, the velocity of the ith vortex induced by the other vortices is given by

$$\frac{d\mathbf{r}_i}{dt} \equiv \mathbf{u}_i = -\frac{1}{2\pi} \sum_{j=1}^{N} (-1)^j \frac{(\mathbf{r}_j - \mathbf{r}_i) \times \mathbf{k}}{\sigma^2 + |\mathbf{r}_j - \mathbf{r}_i|^2} \tag{10}$$

where σ represents the size of each vortex blob. This set of non-linear, first-order, differential equations is solved for the vortex positions using a third-order Runge-Kutta integrator with a fixed time step, δt.

As may be seen from Eq. (10), each vortex contributes to the velocity of all other vortices. The vortex method is, therefore, a form of long-range interaction algorithm in which each vortex interacts with all other vortices. Thus, for a system of N vortices the standard vortex method has a computational complexity of $O(N^2)$. This currently limits the number of vortices to a few thousand, whereas the simulation of flows of engineering interest requires 10^5 to 10^6 vortices.

The VORTEX code used in the Caltech Performance Evaluation Project makes use of the standard, two-dimensional, vortex blob algorithm. The problem considered is the evolution of a vortex sheet in which vortices are initially located at regular intervals, Δ, in

the x direction. Successive vortices alternate at a distance Δ above and below the x-axis, and also alternate in the sign of their circulation, i. e.,

$$\mathbf{r}_j = \Big(\Delta j, (-1)^j \Delta\Big), \qquad \Gamma_j = (-1)^j \tag{11}$$

A constant number of 5000 vortex blobs were used with the following model parameters:

$$\Delta = 0.1, \quad \sigma^2 = 0.0043, \quad \delta t = 0.025 \tag{12}$$

Since the vortex method is essentially a long-range force type of problem it can be implemented on a hypercube using an algorithm similar to that discussed in Chap. 9 of [Fox:88a]. [Catherasoo:87a] has discussed the performance of the vortex method on the AMETEK System 14 hypercube, and [Harstad:87a] has compared the performance of the Caltech/JPL Mark II hypercube with that of the Cyber 205 vector computer for a problem similar to that used in the benchmark described above. Since the concurrent overhead of the parallel long-range force algorithm on the hypercube is inversely proportional to the grain size (the number of vortices per processing node), it was found that the vortex method can be efficently implemented on the hypercube for sufficiently large problems.

4.6 Plasma Physics

Plasma simulations have important applications in many areas of plasma physics, such as electron and ion beam propagation, microwave generation by gyrotrons, and magnetic and inertial fusion. In plasma simulations we seek to determine self-consistently the orbits of up to 10^6 charged particles as they move under the influence of an electromagnetic field which is generated by the plasma particles themselves. The motion of each particle is governed by the equation

$$\frac{d^2\mathbf{x}_i}{dt^2} \equiv \frac{d\mathbf{v}_i}{dt} = \frac{q_i}{m_i}\left(\mathbf{E} + \frac{\mathbf{v}_i \times \mathbf{B}}{c}\right) \tag{13}$$

where $\mathbf{x}_i$, $\mathbf{v}_i$, q_i, and m_i are the position, velocity, charge and mass of the ith particle. $\mathbf{E}$ and $\mathbf{B}$ represent the electric and magnetic fields, and are determined by Maxwell's equations, with the charge and current densities generated by the plasma particles being given by

$$\begin{aligned} \rho(\mathbf{x}, t) &= \sum_{i=1}^{N} q_i \delta(\mathbf{x} - \mathbf{x}_i) \\ \mathbf{j}(\mathbf{x}, t) &= \sum_{i=1}^{N} q_i \mathbf{v}_i \delta(\mathbf{x} - \mathbf{x}_i) \end{aligned} \tag{14}$$

Since each particle influences the motion of all other particles through the Lorentz force, $\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}/c)$, a naive solution for a system of N plasma particles would have a computational complexity proportional to N^2. To allow simulations involving large numbers of particles a faster algorithm is required. The particle-in-cell (PIC) method

avoids the "N^2 problem" by solving for the electromagnetic field on a numerical grid, using either a fast Fourier transform or finite difference technique.

The particles themselves are not constrained to lie at the grid points, thus before the electromagnetic field can be calculated the charge and current densities are interpolated onto the grid. After Maxwell's equations have been solved for the electromagnetic field on the grid, the force on a particle at a particular location is found by interpolation of the values of $\mathbf{E}$ and $\mathbf{B}$ at nearby grid points. The particle position and velocity can then be updated according to Eq. (13).

The BEPS1 code benchmarked in the Caltech Performance Evaluation Project is a one-dimensional, electrostatic, PIC simulation model. The magnetic field is neglected, and the code self-consistently calculates the motion of the plasma particles in the electric field generated by their charge densities.

Equation (13) is integrated using a simple leap-frog scheme to give the velocity and position of each particle.

$$\begin{aligned} \mathbf{v}_i(t + \Delta t/2) &= \mathbf{v}_i(t - \Delta t/2) + \frac{q_i}{m_i}\mathbf{F}_i\Delta t \\ \mathbf{x}_i(t + \Delta t) &= \mathbf{x}_i(t) + \mathbf{v}_i(t + \Delta t/2)\Delta t \end{aligned} \tag{15}$$

The domain of the problem is divided into intervals of equal length, and the net charge density at each point on the resulting one-dimensional grid is obtained by interpolating the charge of each particle onto the nearest grid points. The electric field, $\mathbf{E} = (E, 0, 0)$, is then determined on the grid by solving

$$\frac{dE}{dx} = 4\pi\rho \tag{16}$$

by a fast Fourier transform technique. The force, $q_i\mathbf{E}_i$, on each particle is then found by quadrupole interpolation of the electric field values at the nearest grid points.

The parameters of the benchmark program were taken to be the same as those used by [Decyk:87a] in his extensive evaluation of plasma simulation codes on predominantly shared memory and vector supercomputers. The total number of plasma particles (electrons and ions) was 11264, 128 spatial grid points were used, and the benchmark code was run for 2500 time steps. Initially the particles are spaced at regular intervals, and have velocities distributed randomly between $-v_0$ and $+v_0$.

The implementation of plasma PIC simulation codes on hypercube concurrent computers has been described in [Liewer:88b] [Liewer:89a].

5. The Machines

The computers on which we ran our programs spanned a broad range of the commercially available advanced architecture computers and are shown in Table 1. Our emphasis was decidedly on parallel architectures, consequently we did not evaluate some very powerful or cost-effective computers. For example, we did not make any runs on certain high performance vector architectures like the Convex series of mini-supercomputers or on any of the Japanese supercomputers.

We had one year in which to do this pilot study, so practical considerations had some influence on our choice of systems. Machines at Caltech or San Diego Supercomputer Center were easily accessible. Almost as convenient to use were the computers at Argonne National Laboratory's Advanced Computing Research Facility (ACRF). Fortunately, those three sites contained at least one representative of most of the interesting commercial parallel computers.

The CRAY X-MP and CRAY-2 computers were chosen as the reference point for conventional supercomputers. They are the most strongly identified with supercomputing and probably have the largest user community of any family of supercomputers. The SCS-40 superminicomputer was included in our set of target machines because it was accessible (San Diego Supercomputer Center has one) and so compatible with the CRAY X-MP that little effort was required to run the same programs on it. Runs were also made on an ETA10-E computer since it is a very high performance system and 8-processor configurations were to become available, thus providing the opportunity for modest parallelism.

The hardware market is in constant flux. On the one hand, some interesting new computers, such as the Symult 2010 and Intel iPSC/2, and Meiko Computing Surface were close to being ready in time for this study but did not quite make it. We intend to continue this line of research and will have opportunities to evaluate additional systems in future phases of this project. On the other hand, in the course of this study, Cydra, ETA, Symult, and SCS companies went out of business.

5.1 Bus-connected shared memory multicomputers

A short summary of hardware characteristics of some shared-memory machines used in this study can be found in Table 2.

5.1.1 Alliant FX/8

Characteristic	Alliant FX/8	Multimax 320	Symmetry
CEs	1-8	2-20	2-30
IPs	3-12	0	0
Peak speed(Mflops/CPU)	5.9	0.5 or 2	0.6 or 2
Vector registers (number x words/CPU)	8x32	0	0
Cache KB/CPUs	256/4 and 32/3	64/1	64/1
Total Memory MB	64	4-128	120

Table 2: Some hardware characteristics of the shared-memory machines.

The Alliant FX/8 has two types of processors: up to eight custom vector processors, known as Computational Elements (CEs), and up to twelve Motorola 68020 microprocessors, referred to as Interactive Processors (IPs), all with global access to memory, whose maximum size is 64 megabytes. The CEs and IPs have caches that are connected to the global memory via two busses. Each set of 4 CEs shares a single 256 Kbyte cache. Each set of 3 IPs shares a 32 Kbyte cache. The vector processors are based on Weitek 1064/1065

Machine	Description
Alliant FX/8	Shared memory vector multiprocessor
Encore Multimax	NS32332-based shared memory multiprocessor
Sequent Balance	NS32032/32081-based shared memory multiprocessor
Sequent Symmetry	Intel 80386-based shared memory multiprocessor with optional scalar Weitek chips
BBN Butterfly	MIMD network of MC68020/68881-based processors
INTEL iPSC/1	Intel 80286/80287-based hypercube
Mark III	Hypercube with MC68020/68882 processors
Mark IIIfp	Mark III hypercube with XL Weitek chip set
NCUBE	Hypercube with custom scalar processors
Cydrome Cydra 5	Very Long Instruction Word machine
Connection Machine 2	Massively parallel SIMD machine with 16K nodes and Weitek chips
CRAY X-MP/48	4-node vector supercomputer
CRAY-2	4-node vector supercomputer with large memory
SCS-40	Vector mini-supercomputer, Cray X-MP compatible
ETA10-E	4 vector processors with shared memory

Table 1: Advanced architecture computers studied in the Caltech Performance Evaluation Project.

floating-point chips. Each CE has 8 vector registers, 8 64-bit scalar floating-point registers, 8 32-bit integer registers, and 8 32-bit address registers. Each vector register holds 32 64-bit floating-point numbers. The 68020 processors are normally used for non-floating-point tasks such as compilation, editing, and the operating system but can also be used for user program execution. IEEE 754 floating-point standard arithmetic is supported by both CEs and IPs.

The peak speed of the FX/8 with 8 vector processors is 47.2 Mflops for 64-bit data and 94.4 Mflops for 32-bit data. Basic cycle time is 170 ns; a floating-point 64-bit multiply requires two cycles to execute, as does a multiply-add triad operation.

The operating system, called Concentrix, is based on 4.2BSD UNIX and enhanced to support many features of the FX/8 architecture. The Fortran compiler is able to both vectorize and parallelize automatically or with user-supplied directives.

The system we used in this project was the Alliant FX/8 at Argonne's ACRF. It has 8 CEs, 6 IPs, and 32 megabytes of global memory. Alliant has recently introduced faster versions of this system models FX/80 and FX/82.

5.1.2 Encore Multimax

The two models of the Encore Multimax family use a fast bus to connect two or more microprocessors to a global shared memory. The Multimax 120 was the first model introduced; it has 2 to 20 National Semiconductor 32032 microprocessors with 32081 floating

point units that support IEEE 754 standard 32 and 64 bit floating point arithmetic; each has a peak speed of 0.75 MIPS integer and about 0.3 Mflops floating-point. Each pair of CPUs share a 32 Kbyte cache. Memory sizes are 4 to 128 megabytes.

The Multimax 320 has 2 to 20 National Semiconductor 32332 microprocessors with 32082 floating point units that support IEEE 754 standard 32 and 64 bit floating point arithmetic; each has a peak speed of 2 MIPS integer and about 0.5 Mflops floating-point. Each processor has its own 64 Kbyte cache. Memory size is the same as for the Multimax 120. An option for the 320 is a floating-point accelerator for each node, based on the Weitek 1164/1165 chips. This boosts peak performance to about 2 Mflops.

The operating systems available for both models are UMAX 4.2 and UMAX V. As their name implies, they are compatible with 4.2BSD and System V UNIX, respectively. Languages supported are Fortran, C, Pascal, Basic, Lisp, Cobol, and Ada. The Fortran compiler has a precompiler that provides automatic parallelization. User specified parallelization is also provided through language extensions and calls to system routines.

We used the Argonne ACRF Multimax 320 for our runs. It is a 20 processor system with 64 megabytes of memory. It does not have the Weitek floating point accelerator option.

5.1.3 Sequent Balance and Symmetry

Like the Encore Multimax, the Sequent computers use a fast bus to connect 2 to 30 microprocessors and globally shared memory, though the Sequent machines have a slower bus. The Balance series uses National Semiconductor 32032 microprocessors with 32081 floating point units that support IEEE 754 standard 32 and 64 bit floating point arithmetic; each has a peak speed of 0.75 MIPS integer and about 0.3 Mflops floating-point. Each CPU has an 8 Kbyte cache.

The Symmetry series uses Intel 80386 microprocessors with 80387 floating-point coprocessors that support IEEE 754 standard 32 and 64 bit floating point arithmetic; each has a peak speed of 3 MIPS integer and about 0.6 Mflops floating-point. Each processor has a 64 Kbyte cache. An optional floating-point accelerator uses Weitek 1167 chips at each node; their peak speed is 2 Mflops. The maximum number of processors per system is 30.

The operating system is called Dynix. It is compatible with both 4.2BSD and System V UNIX. Fortran, C, Pascal, and Ada are supported. The Fortran compiler has automatic and user-directed parallelization capabilities.

The Sequent Balance we used for this study was Argonne's 24 processor system with 24 megabytes of memory.

The Sequent Symmetry we used for this study was Argonne's 24 processor system with the Weitek 1167 floating-point accelerators and 120 megabytes of memory.

5.2 Network-connected shared memory - *BBN Butterfly*

BBN Advanced Computers Inc. produces the Butterfly family of computers. These systems consist of microprocessors and memory units that are connected to each other via a specially-designed switch. Although all memory is local to the processor to which it is attached, each processor has access every other processor's memory through a multistage switch. This general type of connection scheme is found in several research computers,

such as the NYU Ultracomputer and IBM's RP3, but at present is unique to the Butterfly among commercial systems. Butterfly processor nodes consist of Motorola 68000 or 68020 microprocessors with one to four megabytes of memory. The largest configuration built to date has 256 processors. The peak performance of each node is 2.5 MIPS and 0.3 Mflops for systems with the 68882 floating-point coprocessor.

Two operating systems are available, the proprietary Chrysalis and a version of MACH UNIX. Fortran, C, and LISP are supported.

The machine we used for our runs was a 16-processor system with Motorola 68020 processors and 68881 floating-point coprocessors at the Rockwell Science Center, Thousand Oaks, California.

Higher performance models of the Butterfly family have been announced with first deliveries expected in 1989.

5.3 Hypercubes

Characteristic	iPSC/1	NCUBE/10	JPL Mark IIIfp
Max Nodes	128	1024	256
Host	Intel 310AP	Intel 80286	Counterpoint
Node processor	Intel 80286	proprietary	68020/882 or Weitek
Peak speed(Mflops/CPU)	0.05	0.5	0.3 or 16
Cache KB/CPU	0	0	128
Memory MB/CPU	0.5	0.5	4

Table 3: Some hardware characteristics of hypercube computers.

Several of the systems we studied use the hypercube connection scheme. A short summary of their hardware characteristics can be found in Table 3. With this architecture, each processor has private memory and is connected to other memory/processor combinations (known as "nodes" in the parlance of message passing architectures) via high speed channels. This architecture scales well to large numbers of processors, because in a hypercube with N nodes each node is connected to only $\log_2(N)$ other nodes, where N must be a power of 2 ($N = 2^m$ for some m).

The message-passing software used for the hypercube applications was CrOS/Cubix. CrOS (Crystalline Operating System) [Fox:88a] is a loosely sychronous set of functions which facilitate a number of data passing operations. Loosely sychronous means that all nodes must issue the same CrOS call before any node can continue execution; however, references to CrOS routines need not occur at the same instant of real time. Cubix [Fox:88a] is a host or control processor program which provides an application running CrOS in the nodes access to most of the UNIX functionality and the C standard I/O library (e.g., scanf() and printf()). All runs in this study were done with CrOS/CUBIX.

5.3.1 Intel iPSC/1

The Intel hypercube we used in this project was the first generation model, the standard version of which has up to 128 nodes; at each node is an Intel 80286 microprocessor with the Intel 80287 floating point coprocessor and 0.5 Mbytes of memory. Each node is capable of about 0.05 Mflops. IEEE-754 floating-point standard arithmetic is supported.

We made our runs on a 64-node configuration of the standard system located at Caltech. A large memory version has the same processors but with 4.5 Mbytes of memory. Its maximum configuration is 64 nodes. Finally, there is a version with vector processors at each node. This model is offered in configurations with up to 64 nodes each with 1.5 Mbytes of memory. The peak speed of the 64-node configuration is 424 Mflops (64-bit).

The iPSC uses an Intel 310AP microcomputer as a program development host. Xenix, an enhanced version of UNIX System III with some 4.2BSD extensions, is the operating system for the 310. Languages supported include Fortran, C, and Lisp. NX is the Intel proprietary operating system that runs on the nodes; it provides message-passing services, a form of multitasking within each node, and other system services. Alternatives to NX include the CUBIX message-passing environment.

There is a newer Intel hypercube, the iPSC/2, that uses Intel 80386/80387 processors in the standard configuration, optional additional Weitek 1167 scalar floating-point processors, or the same vector option as the iPSC/1. Memory sizes of 4, 8 or 16 megabytes per node, faster communications channels, and routing chips are additional differences. We will include the iPSC/2 in the next phase of this project.

5.3.2 NCUBE/10

The NCUBE/10 hypercube has 64 to 1024 nodes, in increments of 64, since 64 nodes fit on a single board. Each node consists of a 32-bit microprocessor and 512 Kbytes of memory. The processors are a custom design and fit on a single chip that contains a general purpose 32-bit microprocessor, memory interface, and 22 DMA communication channels, 11 inbound and 11 outbound. The processor has 16 32-bit general registers and conforms to the IEEE 754 floating-point standard with operations on 32 and 64 bit real data. The peak speed per processor is 0.5 Mflops or 2 MIPS. It also has potential for high-speed I/O: eight I/O channels can transmit data at 90 Mbytes per second bidirectionally. A Host Board containing a microprocessor is provided for program development tasks. Alternatively, one may purchase an optional Sun 3/4 workstation interface and program development tools that run on the Sun.

The operating systems for the NCUBE are proprietary. AXIS, the OS for program development, runs on the Host Board and is reminiscent of 4.2 BSD UNIX. Among the capabilities unique to AXIS are the ability to allocate subcubes to different users, thus providing a means for space-sharing. Fortran and C are supported. The OS that runs on the nodes and provides message passing and other system services is called VERTEX. Alternatives to VERTEX include the CUBIX message-passing environment.

The system we used for this project is Caltech's 512-node NCUBE/10.

5.3.3 JPL/Caltech Mark III Hypercube

The Mark III is a member of the family of hypercubes designed and built as a collaboration between Caltech campus and the Jet Propulsion Laboratory. The Mark III can be configured with up to 256 nodes. Each node has two Motorola 68020 microprocessors, one for computation and one for communications, and four megabytes of memory. The processor used for computation has a Motorola 68882 co-processor with a peak speed of about 0.3 Mflops. IEEE 754 standard floating-point arithmetic is supported. Program development is done on Counterpoint workstations; these are Motorola 68020-based systems

that run UNIX System 5.3. Fortran and C are supported. One Counterpoint is used for up to 32-node cubes. Bigger configurations use multiple Counterpoints.

The CUBIX and CrOS message-passing environment is the most frequently used software. Other environments include Mercury, an asynchronous multitasking system, and Time Warp, an environment designed particularly for discrete-event simulations.

Most of the runs for this project were made on a 32-node Mark III.

The Mark IIIfp hypercube that Jet Propulsion Laboratory completed the spring of 1989 is the next model of this family of hypercubes. A 128-node system will be shared by JPL and Caltech campus. The Mark IIIfp differs from the Mark III in that each node has an additional daughter board with a pipelined floating point unit based on the Weitek XL series of chips. The daughter board has two 128 Kbyte caches of static RAM, one for instructions and one for data. The peak speed of the 32-bit version of the unit is 16 Mflops at the 8 MHz clock rate used in the Mark IIIfp. The 64-bit version of the chips will be installed in mid 1989. IEEE 754 standard floating-point arithmetic is supported. Due to the newness of this system, only one program was run on the Mark IIIfp during the first phase of the project.

5.4 Very long instruction word - *Cydrome Cydra 5*

The Cydra 5 is a heterogeneous multiprocessor system whose numerical computation performance depends on a custom Numerical Processor (NP) with a Very Long Instruction Word (VLIW) architecture. VLIW systems achieve high performance by incorporating more than one functional unit in the CPU and finding ways to utilize the functional units concurrently. This approach takes advantage of parallelism at a very fine level, that of individual arithmetic, logical, or memory operations. Array processors of the 1970's exploited this approach, the Floating Point Systems product line being the most successful. The modern VLIW machines, of which the Multiflow Trace is another example, have considerably more complex hardware and use new compiler techniques that permit the use of high level programming languages like Fortran instead of assembler or microcode.

The VLIW architectures typically have lower peak speeds than vector processors but have the potential for high performance on real user programs. Constructs that inhibit vectorization, such as recursion, conditional branches in loops, can often be handled effectively in VLIW systems.

The Cydra 5 has three types of processors: the NP, a VLIW processor that will be described in more detail shortly, Interactive Processors (IPs) for program development, and I/O processors, all sharing a common bus and up to 512 megabytes of memory. There can be one to six IPs (based on Motorola 68020 processors) and one or two I/O processors, each with up to three VME buses. Both the NP and the IPs support IEEE 754 standard arithmetic for 32 and 64 bit operands.

The NP has the following pipelined functional units:

(1) Combination floating-point adder and integer ALU
(2) Floating-point multiplier
(3) Floating-point divide and square root
(4) Memory read and write ports (2)
(5) Address arithmetic units

The cycle time is 40 ns. Seven operations can start on one machine cycle; up to 56 operations can be executing simultaneously. The NP can execute two different types of instructions. The MultiOp is 256 bits wide and controls all NP functional units in one cycle. The UniOp is 40 bits wide and delivers one operation to a single functional unit per cycle; it is used for code outside inner loops.

The NP has a 32 Kbyte instruction cache and a Context Register Matrix, a set of 2496 registers that are connected so that output from previous operations can be used as input to subsequent operations.

The operating system, called Cydrix 5.3, is an enhanced implementation of UNIX System V.3. Extensions include sockets similar to UNIX 4.2BSD's and asynchronous I/O under user control. Fortran and C are supported, but only the Fortran compiler compiles code for the NP's VLIW architecture; C programs run only on the IPs.

The Cydra 5 we used in this project was loaned by Cydrome to Argonne's ACRF.

5.5 SIMD architecture - *Connection Machine*

The Connection Machine, built by Thinking Machines Corporation, is a massively parallel Single Instruction stream Multiple Data stream (SIMD) system; its maximum configuration has 65,536 processors, each is a one-bit computer. In the original model, the CM-1, each processor has only 512 bytes of memory and floating point operations are performed by groups of processors (microcode is provided to do this). The Connection Machine Model 2 (CM-2) has up to 65,536 bit-serial nodes and 2048 floating-point units (based on the Weitek XL series floating-point chips), one for each 32 of the bit-processors. Only 32-bit IEEE 754 standard arithmetic is supported by this hardware. A future option will provide 64-bit floating point chips. The minimum configuration is 16,384 single-bit processors with 512 floating-point processors. In the CM-2, each processor has 64 Kbits of memory. The system is built from chips that contain 16 bit-serial processors each. Peak speed for the full size machine is 2,500 MIPS for 32-bit integer and logical operations and 20 Gigaflops (32-bits) with the Weitek option.

An external host (VAX running Ultrix, Symbolics 3600 Lisp machine, or Sun 4 running UNIX) serves as a program development front-end and controls an instruction sequencer. There are several communications mechanisms. Broadcast communications send data from the front-end computer to all the processors simultaneously. Within a chip all 16 processors are connected. Communications between the chips are in the hypercube topology. All processors receive the same instruction each cycle, which they may ignore depending on the setting of a flag bit.

The languages currently available on the Connection Machine are special versions of LISP and C, with Fortran soon to become available.

The machine we used for our runs is a CM-2 with 16,384 bit-processors and 512 Weitek units. It is jointly owned by Caltech and the Argonne ACRF and is located in the ACRF.

5.6 Vector supercomputers

The CRAY X-MP/48, CRAY-2, and SCS-40/XM have many similarities. All are vector computers that operate on 64-bit words. Moreover, the X-MP and SCS run essentially the same set of instructions. As will be seen, there are also some important differences.

The salient CPU and memory characteristics for these machines are listed in Tables 4 and 5, respectively. New X-MPs have a slightly faster clock. They may also have a larger albeit slightly slower memory. Conversely, new CRAY-2s can be had with a faster, but smaller memory.

5.6.1 CPU Characteristics

Both the X-MP and CRAY-2 have four CPUs, but each code considered here was run on only one CPU at a time. Hence all speeds are quoted for a single CPU.

The clock period of the three machines varies by more than an order of magnitude: from 4.1 ns on the CRAY-2 to 9.5 ns on the X-MP to 45 ns on the SCS. The peak speed exhibits a similar variation, being inversely proportional to the clock period. This is because one result can be generated every clock period from each vector pipeline that is filled. All three machines have three floating point units (FPUs) per CPU, but only two (one add and one multiply) can generally be used simultaneously to fill vector pipelines. The peak speed of a single CPU is then just the number of add and multiply FPUs divided by the clock period: that is, 488 Mflops for the CRAY-2, 210 Mflops for the X-MP, and 44 Mflops for the SCS.

For well vectorized codes the ratio of the actual speeds between the three machines should, in principle, be close to the ratio of the clock speeds. This is, in fact, the case when comparing the X-MP and SCS. By contrast, there are several hardware features of the CRAY-2 that degrade its relative performance. The most important of these are its lack of chaining, its slower scalar speed, its single port connecting each CPU to memory, and its relatively slow memory. On the other hand, because of its very large memory, the CRAY-2 will excel at problems requiring out-of-core solution on the other machines.

Chaining is the overlapping of two dependent vector pipelines by coupling the output from one to the input of the other. By chaining the multiply and add FPUs the floating point execution rate can be doubled. Chaining is available on both the X-MP and SCS, but not on the CRAY-2. On the latter, it is still possible to overlap the multiply and add FPUs when the operands and results are independent.

For floating point operations the CRAY-2 requires roughly three times as many clock periods (clocks) as the X-MP, whereas the SCS requires just half as many. Relative to the X-MP, then, the CRAY-2 scalar performance is degraded, while the SCS performance is enhanced. In addition, the CRAY-2 can only issue one instruction every second clock period. This primarily slows logical operations, which require only a few clocks.

All three machines have special hardware to perform gather/scatter operations. Such hardware, however, is not invoked by CFT 1.13, the standard compiler on the SCS. On that machine one should compile with CIVIC if gather/scatter operations are expected to be significant.

One other hardware feature that is especially useful for performance evaluation is available on the X-MP, but not on the other two machines. This is the performance monitor, which consists of eight performance counters for each CPU. These may be configured by the user with a FORTRAN- callable subroutine to monitor any one of four groups of events. Generally the events in the first group are of most interest; they summarize the floating-point operations as well as the instructions issued, hold issue cycles, and memory references.

Characteristic	CRAY X-MP/48	CRAY-2	SCS-40/XM
CPUs	4	4	1
FPUs (+,*)/CPU	2	2	2
Clock period (ns)	9.5	4.1	45
Peak speed(Mflops/CPU)	210	488	44
Vector registers (number x words/CPU)	8x64	8x64	8x64
Time (clocks) for FP add	6	19	3 or 4[a]
Time (clocks) for FP multiply	7	19	3 or 4[a]
Chaining	Yes	No	Yes
Gather/scatter	Yes	Yes	Yes[b]

[a] The latter value applies when the FP operation is followed by a store to memory.
[b] The gather/scatter hardware is not invoked by the CFT 1.13 compiler.

Table 4: CPU Characteristics.

5.6.2 Memory

Both CRAYs are shared memory machines, although the CRAY-2 has a modest amount of distributed memory as well. (See Table 5) The most distinctive feature of the CRAY-2 is its very large memory, some 256 Mwords compared to the 8 Mwords on the X-MP and 16 Mwords on the SCS (where each word has 64 bits, i.e., 8 bytes, in all cases). This makes the CRAY-2 best suited for problems that would require out-of-core solution on the other machines.

Characteristic	CRAY X-MP/48	CRAY-2	SCS-40/XM
Shared Memory			
Memory size (Mwords)	8	256	16[a]
CPU to memory ports/CPU	2 load; 1 store	1 load or store	2 load; 1 store
Bidirectional memory	Yes	No	Yes
Banks	64	128;256[b]	16
Bank grouping	4 sections	4 quadrants	4 tiers
Chip type	ECL	Dynamic MOS	MOS
Access time (clocks)	14 scalar; 17 vector	48	8
Cycle time (clocks)	4	45; 22[b]	5
Distributed Memory			
Memory size (words/CPU)	64 T and 64 B registers	16k	64 T and 64 B registers
Access time (clocks)	1	4 scalar; 5 vector	1

[a] The machine at SDSC has another 16 Mwords configured as extended memory.
[b] The latter value results from pseudobanking.

Table 5: Memory Characteristics.

Both the X-MP and SCS have three ports between memory and each CPU: two for loads and one for stores. This nicely complements the two operands and single result of

each FPU and avoids the memory bottleneck of the CRAY-2, which has only a single CPU to memory port.

Bidirectional memory is a hardware feature on the X-MP and SCS that allows simultaneous loads and stores. The CFT compiler checks whether this feature can be safely used. Unfortunately, some assembly-coded library routines fail to make the necessary test, so bidirectional memory is defaulted off at SDSC. However, it is almost always OK to enable bidirectional memory, and doing so usually produces faster code.

On the X-MP the memory access time is 14 clocks for a scalar load and 17 clocks for the first element in a vector load. This compares with an access time for both scalar and vector loads of 48 clocks on the CRAY-2 and 8 clocks on the SCS. Thus the SCS has an advantage over the X-MP in getting data from memory, while the CRAY-2 has a disadvantage.

To facilitate vector processing, memory on all three machines is organized into banks with consecutive vector elements stored in adjacent banks. The number of banks varies from 16 on the SCS to 64 and 128 on the X-MP and CRAY-2. Following access to a given bank, there is a dead time, or memory cycle time, during which a second access to that bank cannot be made. This cycle time is 4 clocks on the X-MP, 5 clocks on the SCS, and 45 clocks on the CRAY-2. To mitigate the effect of the long cycle time on the CRAY-2, each bank is logically divided into two pseudobanks, resulting in an effective cycle time of 22 clocks.

Attempting a second access to a bank before its cycle time has elapsed causes a delay in processing due to what is called a memory bank conflict. Vector operations typically access vector elements separated by a fixed increment or stride. Memory bank conflicts occur for strides that are multiples of 32 on the X-MP, 16 on the CRAY-2, and 4 on the SCS.

To reduce the number of interconnects between the CPUs and memory the banks are combined into four groups called sections on the X-MP, quadrants on the CRAY-2, and tiers on the SCS. This produces additional conflicts for even shorter strides, namely section conflicts for multiples of 8 and quadrant conflicts and tier conflicts for multiples of 2. Such strides should generally be avoided.

Because the four CPUs on the CRAYs share the same memory, conflicts also arise from codes running on the other CPUs. These conflicts are unavoidable. When the X-MP is busy, such conflicts reduce performance by a few percent and, because of the conflict resolution rules, make vector accesses with even strides slightly worse than those with odd strides. On the AFSCC-K CRAY-2 such conflicts degrade performance considerably more, typically by about 20is also due to the dynamic memory, which must be periodically refreshed. The newest model CRAY-2 has a static memory with a 17 clock cycle time, and degradation due to memory conflicts is only a few percent.

Each CPU on the X-MP and SCS has a very small distributed memory consisting of 64 T registers for 64-bit data words and 64 B registers for 24-bit address words. The CRAY-2 has a much larger distributed (or local) memory of 16 kwords/CPU, but compilers have not made effective use of this to date.

5.6.3 Software tools

The software tools available on each machine impact performance and can facilitate

its measurement. Of special note are the FORTRAN compilers, the math libraries, and the timing tools. The operating system was not an issue in this study, since all three computers ran the same system, namely CTSS.

Three compilers were used: CFT 1.13, CFT 1.14, and CFT77 2.0. All are available on the X-MP at SDSC, but only CFT 1.13 runs on the SCS, and only CFT77 is on the CRAY-2 at AFSCC-K. The default compiler on the X-MP is CFT 1.14.

CFT 1.13 and 1.14 are similar compilers, with the latter generating code that is typically a few percent faster. Both are very fast in terms of compilation time, typically processing about 30,000 FORTRAN statements per minute.

CFT77 is the newest compiler of those considered. It consistently generates code that is faster than CFT 1.14, but takes several times longer to compile. Thus CFT 1.14 is preferred for software development, while CFT77 is better for generating production codes. Indeed, as CFT77 only became available on the X-MP at SDSC after this study was underway, the optimization of each code was done with CFT 1.14, whereas the final X-MP benchmarks were done with CFT77.

One further consideration when comparing code compiled at AFSCC-K and SDSC is that the two sites use different linkage options in CFT77, because multitasking is allowed at AFSCC-K, but not at SDSC. The only option available at AFSCC-K is the stack-based linkage, which generates the reentrant code needed for multitasking. This slows the run time of the generated code by a few percent.

Other software tools, well-developed on these machines, are the math libraries. Routines from three general purpose math libraries - IMSL, NAG, and SLATEC (which enlarges LINPACK) - were considered. However, only the linear algebra routines were relevant for the codes considered here. Since these routines were coded in standard FORTRAN, they were generally less efficient than their counterparts in the more specialized OMNILIB library (which includes SCILIB from CRAY Research). For example, OMNILIB contains well-vectorized versions of the LINPACK routines as well as some even faster assembly-coded alternatives. OMNILIB and MATHLIB, another specialized library of efficient routines, were available on all three machines.

Three software timing tools were used to evaluate performance. For arbitrary sections of code, timing information was obtained on all three machines by calling the SECOND subroutine.

Two additional tools were used on the X-MP. By compiling with the FLOWTRACE option on, the relative time spent in each subroutine was easily determined. The most detailed information was obtained from the hardware-performance monitor using the PERFMON and PERFPRT subroutines. By calling these routines, which are in the BENCHLIB library, the speed in Mflops was obtained for arbitrary sections of code.

5.7 ETA10-E

The ETA10-E family of supercomputers is produced by ETA Systems, a subsidiary of CDC, gets its name from "ETAOINSHRDLU", the first row of type on a linotype machine, the width of a newspaper column, and a code phrase used in the Enigma project. ETA produced four supercomputer family members: P, Q, E and G. P, the "piper", and Q are air cooled machines with 24 and 19 ns clock cycles respectively and 1 or 2 processors with 64-512 MBytes of shared memory. The E and G are cooled by liquid nitrogen. The E, the

first of which was installed at FSU, has a 10.5 ns clock cycle, between 1 and 4 processors and 256-1000 MBytes of shared memory; the G, which was installed at FSU (Florida State University) early in 1989, has a 7 ns clock, 2-8 processors and 512-2000 MBytes. In all the models each processor also has 32 MBytes of local memory.

The base operating system for the ETA10-E is called EOS. On top of it runs the familiar-to-Cyber-users VSOS operating system. ETA has also developed AT&T System V version 3 Unix with Berkeley 4.3 extensions. VSOS is similiar to COS, the Cray operating system; in both, commands are primitive with awkward syntax. VSOS's concept of *local* and *permanent* files was useful, for example, when running the same program with different data, different *local* files (with the same name) are used so no conflicts occur. Finally, the ETA10-E supports ETHERNET with TCP/IP protocols so FTP and TELNET are available.

Unfortunately, FSU did not have the multi-tasking library in place, so only one "head" of the four processor machine was benchmarked. A VAX cluster comprising a 780 and a 8700 provided a front-end to the ETA10-E. Batch jobs were submitted from the VAX cluster. Both the 2-pipe Cyber 205 and the ETA10-E were benchmarked on all the codes. Both supercomputers (under VSOS) use the same Fortran compiler: FTN200 version 678. The Cyber runs version 2.2 of VSOS and the ETA runs 1.2. VAST version 2.24L7 is also available but time constraints didn't permit us to make much use of it (with VSOS). The FTN200 Fortran compiler, of course, vectorizes (a little) and optimizes the code; this being controlled by a compiler switch. All the benchmarks were run with both "OPT=0" and "OPT=1". On the ETA10-E with Unix, the ftn77 Fortran compiler was used which invokes VAST and so vectorizes very well; vectorization is invoked via the compiler switch "-v d" and optimization by "-O dprs".

6. Results and Discussion

The data which were accumulated during this study can be found in a series of tables. A key to the notation in these tables is located in Appendix A. In addition to the tables, selected results from the tables are plotted in performance figures for easy visualization. For details about the results in the figures, refer to the corresponding tables. Superimposed on the figures are straight lines which represent perfect parallel performance. They are included for discerning multiprocessor performance from the ideal which for any parallel machine is a speedup curve offset from the ideal performance lines and parallel to it.

6.1 Matrix Multiplication

Matrix multiplication of full matrices is one of the simplest algorithms in linear algebra and typically reflects the optimum performance of computers designed for fast floating point computations. Three flavors of matrix multiplication exist: DOT, SAXPY, and GAXPY [Dongarra:82a]. The DOT formulation is common and consists of a sequence of dot or inner products. The SAXPY form ($Y = Y + a X$, where X and Y are vectors and a is a scalar) is a column-oriented (Fortran) algorithm where the innermost loop contains a scalar times vector and accumulate instruction. GAXPY (generalized SAXPY) is a special implementation of SAXPY where one column vector (Fortran) is accumulated at a time, thus permitting chaining on the CRAY X-MP and SCS-40. Accordingly, for the X-MP and SCS-40, SAXPY is much better than DOT, because it avoids bank conflicts

and accesses memory with unit strides. GAXPY is better still because chaining reduces the number of stores of intermediate results to memory. On sequential processors (lacking vector registers), the SAXPY formulation is still faster than DOT, because of simple memory access patterns (unit strides). Furthermore, for machines with a cache (e.g. Alliant FX/8), unit strides reduce the number of page faults and generally take advantage of the caching mechanism which pulls contiguous blocks of memory. An extensive discussion of the characteristics of matrix multiplication on the CRAY X-MP, CRAY-2, and SCS-40 is available [Pfeiffer:88a] and [Dongarra:82a]. The original Fortran program for the sequential, vector, and shared memory machines is a DOT formulation of matrix multiply.

A special program for multiplying matrices on hypercube multicomputers is used; it is written in C and uses CrOS [Fox:88a] communication routines. The hypercube matrix multiplier is based on a block decomposition of the matrices with one subblock from each matrix in each processor; the subblocks are multiplied in the dot product manner. Blocks are passed to other processors as they are needed [Fox:88a]. For illustration, if each processor contains one n by n block of each matrix, then for every $2n^2$ words communicated, n^3 floating point multiplications occur; for large n the communication times are a small fraction of the total time and multiple processors are effectively utilized. Small matrices show poor efficiencies on the hypercubes and, therefore, disappointing performance because there is a poor ratio of computation to communication.

For the case of the matrix multiplication on the JPL/Caltech Mark IIIfp, there exists a highly optimized version which has microcode for the Weitek fast floating point processor and explicit cache management. The Weitek chip on this machine was capable of only 32-bit precision. The program required around two months to write and debug, so it represents an optimization effort atypical for the present performance evaluation project. Nevertheless, since such a code was already in existence, the corresponding times have been included. In the same spirit, the CRAYs and SCS-40 have assembly matrix multipliers generally available and the performance of these is also included. The construction of a microcoded routine for the benchmark itself violates the intention of the authors and was not done.

The results for matrix multiplication are shown in Table 6 and Figures 1 though 4. The small configurations of the hypercubes do not have enough memory for the N=1024 case shown in Figure 4. Similarly, the small configurations of shared memory machines required excessive execution time for this size matrix.

Beginning with a surprising observation: the performance of the multiplier written in C* and Paris on the CM2 is very disappointing. For N=512, the observed speed is 1.30 Mflops. This is a reflection of C* and not the CM2 which is capable of 3.5 Gflops for the multiplication of 4K by 4K matrices (Connection Machine Technical Report HA87-4, April 1987, p. 60); see also [Johnsson:88a], where a speed of 5.2 Gflops is asserted for unspecified matrix dimensions and single precision arithmetic. The three culprits in the poor performance of our benchmark are needlessly slow data transfer by general routers, a slow front end (VAX 8250) and an inefficient compiler for C*. A significant amount of additional coding in PARIS and specialized data movement would be well worth investing for this code on the CM2. The precision used was 32-bit.

The single processor performance observed on the machines is not surprising. Ex-

Matrix Multiplication Benchmark Times/Sec					
Machine	Comments	N=128	N=256	N=512	N=1024
CRAY X-MP	opt. assembler, CFT77 2.0	0.021	0.171	1.36	10.9
CRAY X-MP	opt. Fortran, CFT77 2.0	0.026	0.196	1.52	12.0
CRAY X-MP	orig. Fortran, CFT77 2.0	0.144	0.936	6.68	50.0
CRAY-2	opt. assembler, CFT77 2.0	0.011	0.084	0.669	5.32
CRAY-2	opt. Fortran, CFT77 2.0	0.029	0.208	1.62	12.3
CRAY-2	orig. Fortran, CFT77 2.0	0.346	3.99	30.8	242.
SCS-40	opt. assembler, CFT 1.13	0.103	0.811	6.45	51.4
SCS-40	opt. Fortran, CFT 1.13	0.136	1.02	7.91	62.6
SCS-40	orig. Fortran, CFT 1.13	0.817	5.42	38.8	290.
ETA10-E	orig. Fortran,inner	0.025	0.149	0.918	6.67
ETA10-E	sdsc Fortran,outer,unroll	0.019	0.122	0.848	6.34
Cyber 205	orig. Fortran,inner	0.064	0.354	2.21	-
Cyber 205	sdsc Fortran,outer,unroll	0.075	0.384	2.26	-
Alliant FX/8	Fortran, 8P, -Ogvc, inner	0.206	2.89	104.	833.
Alliant FX/8	Fortran, 8P, -Ogvc, outer	0.104	1.14	7.98	62.4
Alliant FX/8	Fortran, 8P, -Ogc, outer	0.856	6.73	54.1	433
Alliant FX/8	Fortran, 4P, -Ogvc, outer	0.150	1.78	14.0	109
Alliant FX/8	Fortran, 4P, -Ogc, outer	1.68	13.4	107	856
Alliant FX/8	Fortran, 1P, -Ogv, outer	0.605	6.70	53.1	431
Alliant FX/8	Fortran, 1P, -Og, outer	6.71	53.8	430	3450
Symmetry	Fortran, 1P, outer	27.6	222.	1770.	too long
Symmetry	Fortran, 1P, Weitek, outer	16.8	134.	1070	too long
Symmetry	Fortran, 12P, -K, outer	2.35	18.9	147.	1130
Symmetry	Fortran, 12P, -K, Weitek, outer	1.45	11.6	90.3	736
Symmetry	Fortran, 23P, -K, outer	1.33	10.3	79.3	595
Symmetry	Fortran, 23P, -K, Weitek, outer	0.800	6.32	48.1	387
Balance	Fortran, 1 proc., -K, outer	75.4	602.	4850	MEM
Balance	Fortran, 12P, -K, outer	6.52	51.9	411.	MEM
Balance	Fortran, 23P, -K, outer	3.92	28.6	226.	MEM
Multimax	Fortran, 1P	65.8	271	2270	too long
Multimax	Fortran, 10P, epf -fpa, inner	13.4	89.8	631	4260
Multimax	Fortran, 10P, epf -fpa, outer	6.38	50.2	402	3200
Multimax	Fortran, 19P, epf -fpa, outer	3.43	27.0	210	1680
Mark III	1P, C, CrOS	41.8	332	MEM	MEM
Mark IIIfp	1P, C, Weitek assembly, 32b	0.75	5.69	MEM	MEM
Mark III	16P, C, CrOS	2.94	22.5	176	1390
Mark IIIfp	16P, C, Weitek assembly, 32b	0.13	0.63	3.72	25.6
Mark III	32P, C, CrOS	1.63	12.4	97.5	772.
Mark IIIfp	32P, C, Weitek assembly, 32b	0.10	0.43	2.17	14.4
NCUBE	1P, C, CrOS	MEM	MEM	MEM	MEM
NCUBE	16P, C, CrOS	4.24	31.5	MEM	MEM
NCUBE	32P, C, CrOS	2.31	16.3	124.	MEM
NCUBE	64P, C, CrOS	1.32	8.50	63.1	MEM
NCUBE	128P, C, CrOS	0.83	4.68	32.6	248.
NCUBE	256P, C, CrOS	0.67	2.76	17.1	126.2
NCUBE	512P, C, CrOS	0.58	1.75	9.59	65.5
CM2	C*/PARIS	11.8	45.6	206	-
Cydra		0.250	1.76	13.6	-

Table 6: Execution times for full square matrix multiplication.

amination of the $N = 128$ case shows the ranking per processor, slowest to fastest, is: Balance, Multimax, NCUBE (estimated), Mark III, Symmetry without Weitek, Symme-

try with Weitek, Mark III with Weitek (assembly), Alliant, Cydra, SCS-40, Cyber-205, X-MP, ETA10-E, and CRAY-2. Other matrix sizes generally corroborate this ordering; however, the performance of the Weitek processors is significantly better for the larger matrix sizes because of its pipeline architecture.

The high end performance of the single processor machines is tantalizing. For most machines, the best performance occurs for N=1024, the largest matrices in the benchmark. The best performance (for all matrix dimensions) is the assembly coded multiplier on the CRAY-2. The CRAY-2 has a peak speed of 488 Mflops/CPU and the observed speed varied from 381 Mflops for N=128 to 404 Mflops for N=1024. The peak speed for the CRAY X-MP is 210 Mflops/CPU while the observed speed for the assembler matrix multiplier varied from 196 Mflops for N=128 to 198 Mflops for N=1024. ETA10-E has a peak speed of 381 Mflops/CPU for 64-bit arithmetic and the observed speeds varied from 221 Mflops for N=128 to 339 Mflops for N=1024. The large variation in ETA10-E speed is expected because of its large vector start up overhead. Cydra does not compare as favorably to the X-MP as the SCS-40 and Alliant FX/8. The SCS-40 was about 5 times slower than the X-MP, as expected based on the difference in clock rates.

The speed of shared-memory multiprocessors and distributed-memory hypercubes can be increased by using more processors. The Mark IIIfp and the Alliant FX/8 show similar single processor performance (e.g. see $N = 256$), and by using 8 processors, these two machines exhibit nearly the same speed as the SCS-40. On the $N = 1024$ case, the 32 processor Mark IIIfp approaches the X-MP performance; In spite of its modest processor, the 512 processor NCUBE is also close in speed to the Alliant FX/8 and SCS-40. The 128 processor Mark IIIfp multiplied N=1024 matrices at 480 Mflops. Contrast the near-constant speed of the X-MP over matrix sizes with the widely varying performance of the hypercubes: $N = 256$ ran at 197 Mflops on the X-MP and 78 Mflops on the Mark IIIfp. The shared-memory machines suffer from this effect also, but to a lesser degree. The symptom of this parallel efficiency loss is seen clearly in Figure 1; the NCUBE, Mark IIIfp and Alliant curves are not linear; rather, they tend towards a minimum with increasing numbers of processors. Decreasing grain size is responsible for this behaviour due to the increased relative cost of data transfers. For the Alliant and Mark IIIfp, which have strong floating point units, the inefficiency occurs at smaller numbers of processors. In fact, because of their strong CPUs, neither the Alliant nor the Mark IIIfp has an ideal speedup curve for any matrix size; increasing the matrix size does move their performance curves towards ideal, of course. The irregular shape of the N=1024 Mark IIIfp curve is due to the alternation between square and rectangular tori decompositions. Notice that for $N = 128$, the 16 processor NCUBE performance is very similar to the Balance with the same number of processors and the larger NCUBE configurations effectively extend the Balance performance. If larger Balance configurations were available, the relative merits of the shared-memory and the distributed memory NCUBE for more processors would be apparent.

The remaining (parallel) machines display performances that are significantly below those for the previous machines (CRAYs, SCS-40, Alliant, 512 processor NCUBE). For this slower group of processors, the multiprocessor speedup are near optimal for all the matrix dimensions. The one exception is the case $N = 128$ on a 512 processor NCUBE.

Near optimal speedups is due to the favorable ratio of computation time to data transfer time. The unoptimized Mark III multiplier performed about 1.3 times faster than the NCUBE with the same number of processors.

The machines with special floating point units were timed with those units turned off. The "vector" mode of the Alliant was 7 to 11 times the speed of the "nonvector" mode. This ratio is much larger than the 2 to 4 reported in the Alliant literature; the discrepancy is due to a poor choice of a concurrent loop by the optimizer when vector processing is off; compiler directives would remedy this poor choice. This emphasizes the need to parallelize the outer loops in a nested set instead of the inner loop; unfortunately, the Alliant compiler may not do this without intervention by the user. On the Symmetry, the use of Weitek chips increased the speed only by a factor of 1.7. This is a typical performance factor observed for the Weitek chips on the Symmetry, yet it represents only 10% of the peak speed of the Weitek chips.

For the shared and distributed memory machines, it is important to measure the efficiency of utilization of the multiple CPUs. For the largest matrices, the Alliant's efficiency was 86%, the Symmetry's was 97%, the Mark III showed 84% and the Mark IIIfp showed 41% efficiency. The difference between the Mark III and IIIfp efficiencies is due to the constant communication costs with increased floating point performance. Clearly high efficiency will come only with very large matrices on the Mark IIIfp. The Encore Multimax (19 processors) was 57% efficient for N=512.

The performance improvement due to optimization depends on the experience of the programmer, but it is nevertheless an interesting quantity, giving the potential user some indication of the potential performance increase he can expect for a modest investment in time. Often the greatest increases in performance occur after only a short investement in optimization. Further performance improvements tend to require increasing levels of effort (e.g. new choice of algorithm better suited to particular architecture). The full optimization time allotted was used for each each shared memory machine. The hypercube codes were already well optimized and were run as is.

Performance Improvement-Fortran Only

Matrix Size	X-MP	SCS-40	ETA10-E	Alliant(8)	Multimax(10)
N=128	5.5	6.0	1.3	2.0	2.1
N=256	4.8	5.3	1.2	2.5	1.8
N=512	4.4	4.9	1.1	13.	1.6
N=1024	4.2	4.6	1.1	13.	1.3

The large performance increase from optimization observed for the Alliant FX/8 on N=512 and N=1024 is probably a symptom of the DOT formulation. The performance improvements on the other machines range from less than 2 to 5-6. This is shown above.

6.2 Banded LU Decomposition

The LU decomposition of a nonsymmetric, banded matrix is a typical time-consuming part of many application programs for solving partial differential equations. It represents a challenge for distributed memory computers because the data distribution is highly nontrivial, the grain size grows relatively slowly with matrix dimension, and the algorithm

has an inherent sequential component - the progressive movement down the band from top to bottom. The benchmark code *did not* employ partial pivoting. The program for the sequential and shared-memory machines was written in Fortran and that used on the hypercubes was in C.

The distributed memory code is based on a scattered data decomposition. At any step, the active part of the matrix is the square block within the band. The active square moves down the band as the computation proceeds. The data is distributed so that the processors all share the active block of the matrix at all steps. The basic computation is a rank-one update of the active block. Processors communicate the necessary part of the pivot row and multipliers to the other processors and the computation proceeds in parallel. For details see [Fox:88a].

The original *sequential* Fortran code is overly general for a matrix with constant bandwidth. Elements of the matrix are accessed indirectly - meaning that an index array is employed to point to specific elements. This one feature inhibits every Fortran compiler from optimizing the code, since indirect addressing prevents the compiler from assessing possible data dependencies.

Optimization for the vector and shared-memory machines was the same. The indirect indexing was replaced by direct indexing. The compilers for the CRAY X-MP, CRAY-2, and SCS-40 could then vectorize the loops directly. CFT does permit vectorization of loops with indirect indexing through the use of directives, but the performance of direct indexing is better. Both the Alliant FX/8 and the Sequent machines required explicit compiler directives to instruct the compiler about possible data dependencies. See the Alliant section of Table 7. With directives present in the source, the Alliant compiler produced an executable exploiting both concurrency and vector processors. Likewise, the Kuck and Associates preprocessor of Sequent and the epf preprocessor of Encore could locate concurrency. Without the replacement of indirect indexing with direct, the compilers for shared-memory machines could not locate concurrency in the code. The CRAY X-MP and SCS-40 optimization also included the use of bidirectional memory and assembler level SAXPY. The hypercube codes were not further optimized and the indirect indexing was left intact.

Results are shown in Table 7 and selected results are displayed in Figures 5 through 7.

The relative performance of all the machines is similar to that observed on the matrix multiplication with some exceptions: CRAY-2 is slower than the X-MP; the SCS-40 is significantly faster than the Alliant, especially for a band width 511; the ETA10-E performance is worse than that of the SCS-40. The CRAY-2 is slower for LU than for matrix multiplication because of the loss of an ability to chain. The Alliant is hurt by the smaller grain size available for parallelization (two nested loops). The ETA10-E suffered because the compiler failed to expand in-line the SAXPY subroutine. The CRAYs and SCS-40 presented the best performance for all bandwidths. Cydra performance did not improve much with increasing bandwidths as it did in the case of matrix multiplication.

The absolute speeds of all machines are significantly lower than for matrix multiplication. This is not surprising given the simplicity of matrix multiplication. Only the X-MP surpassed 10 Mflops on the w=31 case with the CRAY-2 close behind. The Alliant is the

Banded LU Decomposition Benchmark Times/Sec N=1024						
Machine	Comments	w=31	w=63	w=127	w=255	w=511
X-MP	opt assembler, CFT77 2.0	0.035	0.076	0.176	0.462	1.28
X-MP	opt Fortran, CFT77 2.0	0.060	0.130	0.280	0.668	1.69
X-MP	orig Fortran, CFT77 2.0	0.126	0.490	1.89	7.26	26.5
CRAY-2	opt assembler, CFT77 2.0	0.050	0.110	0.264	0.732	2.00
CRAY-2	opt Fortran, CFT77 2.0	0.089	0.187	0.409	1.05	2.87
CRAY-2	orig Fortran, CFT77 2.0	0.170	0.633	2.36	8.67	31.1
SCS-40	opt assembler, CFT 1.13	0.148	0.320	0.790	1.98	5.20
SCS-40	opt Fortran, CFT 1.13	0.267	0.566	1.22	2.93	7.25
SCS-40	orig Fortran, CFT 1.13	0.446	1.72	6.68	25.4	92.2
ETA10-E	orig Fortran	0.154	0.587	2.37	10.4	40.1
ETA10-E	SDSC-Fortran	0.195	0.407	1.25	5.89	22.2
Cyber-205	orig Fortran	0.247	1.91	8.93	41.6	-
Cyber-205	SDSC-Fortran	0.452	2.99	12.8	57.3	-
Alliant	Fortran, 8P, -Ogvc, CD, DI	0.166	0.413	1.54	15.2	96.4
Alliant	Fortran, 8P, -Ogc, CD, DI	0.186	0.604	2.34	11.4	47.6
Alliant	Fortran, 4P, -Ogvc, CD, DI	0.240	0.615	2.31	27.6	189
Alliant	Fortran, 4P, -Ogc, CD, DI	0.323	1.14	4.47	22.7	92.2
Alliant	Fortran, 1P, -Ogv, CD, DI	0.675	1.93	7.86	108	774
Alliant	Fortran, 1P, -Og, CD, DI	0.985	3.86	17.1	90.5	389
Alliant	Fortran, 8P, -Ogvc, orig.	1.12	4.84	24.5	110	450
Alliant	Fortran, 1P, -Ogv, orig.	1.12	4.39	19.1	97.6	408
Alliant	Fortran, 1P, -Og, orig.	1.13	4.42	18.5	99.0	414
Symmetry	Fortran, 1P	4.85	22.9	91.7	356	1300
Symmetry	Fortran, 1P,Weitek	4.43	19.5	79.3	308	1120
Symmetry	Fortran, 12P, DI	1.08	2.55	8.65	29.6	104
Symmetry	Fortran, 12P, DI, Weitek	0.950	2.20	7.35	25.0	85.6
Symmetry	Fortran, 23P, DI	0.850	2.02	5.08	17.6	61.3
Symmetry	Fortran, 23P, DI, Weitek	0.817	1.90	4.77	16.9	59.5
Balance	Fortran, 1P	12.1	51.0	206	800	2950
Balance	Fortran, 12P, -K	3.57	7.05	21.2	70.0	292
Balance	Fortran, 23P, -K	3.57	7.00	14.7	45.5	260??
Multimax	Fortran, 1P -fpa -O, orig	5.98	26.1	107	417	1560
Multimax	Fortran, 1P -fpa -O, DI	5.00	21.5	83.8	326	1182
Multimax	Fortran, 10P, epf -fpa -O, orig	5.92	25.6	105	405	1490
Multimax	Fortran, 10P, epf -fpa -O, DI	1.55	3.43	10.3	34.4	121
Multimax	Fortran, 18P, epf -fpa -O, DI	1.83	2.98	7.42	24.0	77.4
MarkIII	1P, C, CrOS	8.83	32.5	124	470	MEM
MarkIII	4P, C, CrOS	3.51	10.1	34.7	126	449
MarkIII	16P, C, CrOS	1.18	3.28	9.57	32.8	114
NCUBE	1P, C, CrOS	12.7	MEM	MEM	MEM	MEM
NCUBE	4P, C, CrOS	7.64	17.4	52.7	MEM	MEM
NCUBE	16P, C, CrOS	5.28	7.80	17.0	50.2	165
NCUBE	64P, C, CrOS	4.45	5.19	7.65	16.4	46.2
NCUBE	256P, C, CrOS	4.24	4.47	5.20	7.46	15.4
Cydra	-O-M2-NP-save	0.64	2.51	9.81	37.5	136

Table 7: LU decomposition times for banded matrices with 1024 equations and various bandwidths.

only parallel machine to exceed 1 Mflop performance on w=31.

For the smallest bandwith $w = 31$, shown in Figure 5, the multiprocessor speedups were far from ideal with the exception of the Alliant which showed reasonable speedups

(e.g. 8 processor efficiency of 50% and 4 processor efficiency of 70%). As in other cases, the 8 processor Alliant compared very favorably to the SCS-40 on the small bandwidth cases. For w=31, the 256 processor NCUBE was showing no further speedups upon increasing processors and a far from ideal speedup curve over the entire range of configurations; it is over a factor of 10 slower than the 8 processor Alliant. With as few as 16 processors, the NCUBE shows 15% efficiency; for more processors, the efficiency is even less. In their largest configurations, the efficiencies for the Balance, Mark III, and Symmetry are 15%, 37%, and 25%, respectively. From Figure 5 it is apparent that the Weitek on the Symmetry buys little performance for this small grain problem. Comparing the NCUBE with the Balance, we see that the shared-memory Balance displays a more nearly ideal speedup curve than the NCUBE for this bandwidth.

The largest bandwidth case w=511, shown in Figure 7, is quite different from the smallest bandwidth case for both vector and parallel machines. The performance of the uniprocessor vector machines spans a wider spectrum than the w=31 case. The CRAY X-MP, CRAY-2, and SCS-40 are running at 87, 56, and 21 Mflops, respectively and outperform the remaining machines by a significant margin. The Mark III and NCUBE could not run this case with the smaller configurations because of memory constraints. The multiprocessor speedups are much closer to ideal than for the w=31 case. For w=511, the NCUBE presents much better speedup curves (67% efficiency and 5-10 Mflops) and performs the decomposition 3 times faster than the Alliant (100% efficiency and a 3.6 Mflops, nonvectorized). Notice in Table 7 that for w=255 and 511, vectorization reduced the speed of the Alliant by 30% to 50 %. This may well represent a memory or cache conflict. Because of the poor vectorization performance of the Alliant on the large bandwidth benchmarks, the SCS-40 was a factor of 10 times faster than it. There is a pronounced degradation of the Multimax's performance from 18 processors to 19 which is most severe in the small bandwidth cases, but is almost absent for w=511. The source of the dramatic loss in efficiency has not been investigated. For this reason, no 19 processor results are given. For w=511, the largest configurations of Balance, Mark III, Multimax, and Symmetry display efficiencies of 49%, 90-98%, 85%, and 92%. The Mark III efficiencies are not precise because single processor times are unavaiable.

The Balance processor is nominally much slower than that of the NCUBE. Here, the unusually low speed of the NCUBE processors is probably due to the use of indirect indexing of arrays which was not changed because of the complexity of the hypercube program.

6.3 Quantum Chromodynamics (QCD)

Our QCD benchmark program was originally written for a parallel, non-pipelined machine, since it offers excellent parallelism by domain decomposition. The pure gauge QCD algorithm can be parallelized and/or vectorized because the SU(3) updates can be done entirely independently for links that are not part of the same plaquette (elementary 1x1 square). The space-time lattice can be decomposed and different domains updated in parallel. Furthermore, the independent links can be gathered into a vector and updated in vector fashion. The Caltech QCD code is parallelized but not vectorized. The bulk of floating point operations are in a single procedure, which multiplies two 3x3 complex matrices. Fortran QCD codes by others are vectorized for the CRAY X-MP and run on

one processor of that machine with a sustained performance of 60 Mflops (in Fortran, 120 Mflops with inner loops written in CAL) [Gupta:88c]. This vectorized code was not included in our benchmark suite.

We should also note that the lattices chosen for the benchmark are much coarser than those used in modern scientific simulations. This penalizes highly parallel machines, since the grainsize rapidly becomes small when the number of processors is increased. However, we can extrapolate the performance on more realistic lattices to many processors because of the simple scaling of the performance; we predict that configurations like a 512-processor NCUBE should outperform the vector sequential machines.

C-QCD (Distributed Memory) Benchmark Results				
Machine	Comments	2x2x2x2	4x4x4x4	8x8x8x8
NCUBE	Cubix; P=1	41.8	641.22	10187.57
NCUBE	Cubix; P=16		48.66	702.00
NCUBE	Cubix; P=32			361.9
NCUBE	Cubix; P=64			187.4
NCUBE	Cubix; P=128			96.9
NCUBE	Cubix; P=256			51.56
NCUBE	Fast Cubix; P=16		43.87	664.12
NCUBE	Fast Cubix; P=256			46.67
NCUBE	Express; P=1	36.17	570.91	
NCUBE	Express; P=2		307.75	
NCUBE	Express; P=4		165.01	2456.76
NCUBE	Express; P=8		88.03	1273.70
NCUBE	Express; P=16		46.76	659.35
NCUBE	Express; P=32			344.63
NCUBE	Express; P=64			179.66
Mark III	P=1	40.6	648.7	10378.6
Mark III	P=16		43.2	671.6
Mark III	P=32			350.7
Intel iPSC/1	P=1	103.7	1648.1	26403.4
Intel iPSC/1	P=16		126.6	1855.9
Intel iPSC/1	P=32			982.7
Intel iPSC/1	P=64			516.7
Butterfly	CrOS; P=1	30.0	469.9	
Butterfly	CrOS; P=2	15.2	236.7	
Butterfly	CrOS; P=4	7.7	119.3	
Butterfly	CrOS; P=8	3.8	59.9	
Butterfly	CrOS; P=16	1.9	32.3	

Table 8a: Distributed memory C lattice QCD times.

The QCD results are given in Tables 8a through 8e and selected data are displayed in Figures 8 and 9. As in the LU benchmark, the observed speeds are well below those reported for matrix multiplication. Only the CRAY X-MP and CRAY-2 exceeded 10 Mflops for the 4^4 lattice. For the 8^4 lattice, only the NCUBE performs above 10 Mflops (we did not run this on the CRAYs).

For the 4^4 lattices, CRAYs are the fastest, turning in speeds of 21 Mflops for the best optimized (SDSC) code. The original FORTRAN code runs at 8 Mflops. The optimization for the CRAYs consists of using an assembly language random number generator (instead

Fortran-QCD (Distributed Memory) Benchmark Results				
Machine	Comments	2x2x2x2	4x4x4x4	8x8x8x8
NCUBE	Cubix; P=1	30.6	488.9	7820.5
NCUBE	Cubix; P=16		41.2	580.0
NCUBE	Cubix; P=32			306.3
NCUBE	Cubix; P=64			161.0
NCUBE	Cubix; P=128			84.4
NCUBE	Cubix; P=256			54.2
Mark III	P=1	28.3	452.7	7237.3
Mark III	P=16		34.7	526.9
Mark III	P=32			273.6
Intel iPSC/1	P=1	108.1	1721.0	27550.1
Intel iPSC/1	P=16		138.3	1997.1
Intel iPSC/1	P=32			1059.0
Intel iPSC/1	P=64			565.0

Table 8b: Distributed memory Fortran lattice QCD times.

Fortran-QCD (ORIGINAL) Benchmark Results				
Machine	Comments	2x2x2x2	4x4x4x4	8x8x8x8
CRAY X-MP	CFT 1.14	0.901	14.39	
CRAY X-MP	CFT77 2.0	0.571	9.11	
CRAY-2	CFT77 2.0	0.699	11.09	
SCS-40	CFT 1.13	4.74	75.40	
ETA10-E	OPT=1			
Cyber 205	OPT=1	1.74	27.77	
Alliant FX/8	no -Ogvc	9.82		
Alliant FX/8	-Ogvc; P=1	3.46	54.27	
Alliant FX/8	-Ogvc; P=2	3.37	53.99	
Alliant FX/8	-Ogvc; P=8	3.31	52.81	
Encore	P=1	22.93	367.13	
Cydra	P=1	0.92	14.43	

Table 8c: Sequential Fortran lattice QCD times.

Fortran-QCD (SDSC-Fortran) Benchmark Results				
Machine	Comments	2x2x2x2	4x4x4x4	8x8x8x8
CRAY X-MP	CFT 1.14	0.262	4.16	
CRAY X-MP	CFT77 2.0	0.236	3.74	
CRAY-2	CFT77 2.0	0.367	5.83	
SCS-40	CFT 1.13	1.162	18.53	
ETA10-E	OPT=1			
Cyber 205	OPT=1	0.85	13.46	
Alliant FX/8	no O	7.87		
Alliant FX/8	-Ogvc; P=1	3.03	48.22	772.05
Alliant FX/8	-Ogvc; P=2	2.78	44.28	712.10
Alliant FX/8	-Ogvc; P=8	2.60	41.15	661.23
Cydra	P=1	0.80	12.62	201.84

Table 8d: SDSC optimized sequential Fortran lattice QCD times.

QCD (Shared Memory) Benchmark Results				
Machine	Comments	2x2x2x2	4x4x4x4	8x8x8x8
Butterfly	US; C; P=1	36.4	569.1	
Butterfly	US; C; P=2	18.4	262.8	
Butterfly	US; C; P=4	9.3	127.5	
Butterfly	US; C; P=8	4.8	64.3	
Butterfly	US; C; P=16	4.5	33.5	
Alliant FX/8	-Ogvc; P=1	3.37	52.1	834.8
Alliant FX/8	-Ogvc; P=2	2.28	35.5	568.0
Alliant FX/8	-Ogvc; P=4	1.92	29.4	465.1
Alliant FX/8	-Ogvc; P=8	1.58	25.2	403.9
Balance	P=1	52.1	1297	20870
Balance	P=2		653	10462
Balance	P=4		326	5227
Balance	P=8	7.2	164	2607
Balance	P=16		84	1330
Balance	P=22		74	977
Balance	P=23		78	987
Symmetry	P=1	16.8	289	4630
Symmetry	P=2		146	2322
Symmetry	P=4		73	1167
Symmetry	P=8	2.5	37	588
Symmetry	P=10		50	473
Symmetry	P=14		24	396
Symmetry	P=15		36	317
Symmetry	Weitek; P=1	7.0	110.3	1790
Symmetry	Weitek; P=2		55.8	894
Symmetry	Weitek; P=4		28.2	453
Symmetry	Weitek; P=8	1.1	14.5	226
Symmetry	Weitek; P=10		40.3	263
Symmetry	Weitek; P=14		89.3	333
Symmetry	Weitek; P=15		100.1	343

QCD code was Fortran unless otherwise indicated.

Table 8e: Shared memory Fortran (C on Butterfly) lattice QCD times.

of the one in the original code which takes about 50% of the time), removing references to subroutines by inline expansion, changing character variables to integer in logical operations. Vectorization of the SU(3) matrix multiplier and enabling of bidirectional memory provided further, but less significant improvements.

The hypercubes and the BBN Butterfly used domain decomposition and message-passing. (The Butterfly emulated message-passing by using areas of memory as pseudo-communication channels [Baillie:88j].) This approach gives excellent parallelism, as evidenced by the nearly ideal speedup curves in Figures 8 and 9 for many of the machines. Even the iPSC/1, which has notoriously slow communications, displays a nearly perfect speedup curve. For QCD the NCUBE processors are significantly faster than the Balance processors, in accord with the discussion in section 5 and in contrast with the results for banded LU decomposition in section 6.2. The Mark III runs the original code at around 0.1 Mflops per processor, however the Mark IIIfp runs an optimized code (which has crucial parts in Weitek assembly language) at 4.3 Mflops per processor, giving 550 Mflops for a

full 128-node machine and easily outperforming a 4-processor CRAY X-MP [Ding:88c].

Most of the shared memory machines also do well on this benchmark yielding linear speedups. However, there are two anomalies: the Alliant, and the Sequent Symmetry with Weitek floating-point on more than 8 processors. The Alliant showed only a factor of two improvement when going from one to eight processors. This disappointing result is a combination of two factors. Firstly, one factor of two loss is due to synchronization inherent in the Alliant. The lattice update proceeds via a red-black checkerboard: reds are all updated followed by blacks. Synchronization on the Alliant prevented the processors with black links from moving on to red links. (The Sequent Symmetry did not suffer from this effect since its processors which pick up a black link discard it and carry on trying till they get a red one.) Secondly, since updating proceeds through the lattice with very little re-use of data, the other factor of two is probably due to inefficient use of the cache. The Symmetry with Weitek floating-point is faster than the Symmetry without on small numbers of processors. However, for the 4^4 lattice, the 15 processor Symmetry with Weitek took almost at long as the 1 processor machine. This is partly because, at about 8 processors, the multi-tasking overhead becomes larger than the grain size of the tasks, so the processors finish updating their link faster than they can get another. Unfortunately, this is not the whole story - it explains why the Symmetry and Symmetry with Weitek curves come together in Figure 9 but it does not explain why these curves cross in Figure 8. We hypothesize that there is some sort of "dead time" with the memory such that if processors try to access it too frequently, they are slowed down somehow. Of course, in order to get around this larger parts of the lattice should be given out to each processor thus increasing the grain size.

Finally, we ran a C* version of our QCD code on the Connection Machine using the generalized communications provided by the router. The performance was disappointing - 10 Mflops. This should be contrasted with the 900 Mflops performance obtained by [Brickner:89a] for a similar QCD code written in *Lisp using the NEWS communications and virtual processors.

We have developed performance models for the QCD benchmark on most parallel computers: [Baillie:88p] and [Baillie:88g]. Further, [Walker:88c] has analyzed the NCUBE in detail.

6.4 Tracker

This code is unique among our benchmarks in that it offers almost no opportunity for vectorization. It is also very memory-intensive, which means that caches are not as useful as they might be for some other codes.

There are a number of algorithmic issues which impact the performance of this code on advanced architecture machines. A Newton-Raphson iteration involving small matrices is done for every track; straightforward vectorization is ineffective for this, however, parallelization is obvious. Track splitting and merging give low efficiencies on vector processor architectures. The hypercube code requires two steps which are unnecessary in the sequential code: aggregation of a global table of track data and redistribution of the tracks to ensure load balance and minimal data transfer. Global table construction and track redistribution are accomplished with general routing software built on CrOS communication software. Also on the message-passing machines, a time-consuming global communication

phase is necessary in order to sort the tracks. This, coupled with the unusual memory requirements of this code, leads to a rapid loss in performance when the grain size (number of tracks per processor) becomes small. More details about running the tracker on hypercubes are given in [Baillie:88f]. On shared memory machines, the hypercube code performs badly due to the two extra local-memory specific steps. Hence we spent a full month rewriting the C code for shared memory machines, in particular the Sequent Balance and Symmetry [Cao:88a].

C/Fortran-TRACKER Benchmark Results			
Machine	Comments	85 targets	480 targets
CRAY X-MP	Fortran, CFT 1.14	1.99	18.18
CRAY X-MP	Fortran, CFT77 2.0	2.49	16.3
CRAY-2	Fortran, CFT77 2.0	3.65	24.8
SCS-40	Fortran, CFT 1.13	10.2	59.8
ETA10-E	Fortran, OPT=1	3.26	27.67
Cyber 205	Fortran, OPT=1	4.86	40.05
Cydra 5		9.84	
Alliant FX/8	Fortran, -Ogvc; P=1	14.78	106.40
Alliant FX/8	Fortran, -Ogvc; P=8	13.55	95.90
Balance	C, P=1	148.81	881.60
Balance	C, P=2	76.38	461.54
Balance	C, P=4	40.32	254.85
Balance	C, P=8	22.28	150.75
Balance	C, P=16	13.65	98.64
Symmetry	C, P=1	43.82	240.24
Symmetry	C, P=2	22.65	126.44
Symmetry	C, P=4	12.12	69.26
Symmetry	C, P=8	6.83	40.85
Symmetry	C, P=16	4.32	27.06
Encore	P=1	84.85	537.45
NCUBE	C, P=1	193.79	
NCUBE	C, P=2	109.37	
NCUBE	C, P=4	75.21	
NCUBE	C, P=8	65.18	
NCUBE	C, P=16	55.27	177.64
NCUBE	C, P=32		167.09
NCUBE	C, P=64		147.24
NCUBE	C, P=128		147.88
NCUBE	C, P=256		150.96
NCUBE	C, P=512		162.71
Mark III	C, P=1	114.66	590.39
Mark III	C, P=2	67.99	323.29
Mark III	C, P=4	41.78	188.65
Mark III	C, P=8	30.88	119.09
Mark III	C, P=16		87.85
Mark III	C, P=32		71.79
Intel iPSC/1	C, P=4	491.02	
Intel iPSC/1	C, P=8	427.03	
Intel iPSC/1	C, P=16	399.13	
Intel iPSC/1	C, P=32	386.06	
Intel iPSC/1	C, P=64	379.90	

Table 9: Tracker times.

Benchmark results are shown in Table 9, and in Figures 10 and 11. This tracker data shows that the CRAYs perform well, which is due to the fact that they are the fastest scalar machine we tested. Long-vector specialists like the Cyber 205 did poorly. The Alliant did poorly because its compiler could not find much parallelism. Optimization for the Alliant consisted of inserting compiler directives. On the other hand, the Cydra's compiler did extract parallelism from this non-vectorized code.

The hypercubes were unable to bring many processors into play because of the low grainsize and heavy communication requirements. The Intel iPSC/1 performs very badly due to its large communication latency. The Mark III with fewer processors (and more memory per processor) outperforms the NCUBE. The NCUBE's performance has been increased by a factor of 1.3 using a library of communication routines which are implemented directly on the hardware ("fast CrOS") [Baillie:88f].

The Sequent Balance and Symmetry shared memory machines performed very well exhibiting almost linear speedup, showing that it was worthwhile rewriting the code. Similarly, if we ported this code to the Encore Multimax it would also show good speedups.

At the last minute, TRACKER was run on the IBM 3090/600 and the optimized version running on 6 processors out-performed the X-MP. The results are shown in Figures 10 and 11. The benchmarks on the IBM machine will be discussed in future work.

6.5 Chemical Dynamics

LOGD is a relatively small production code used in chemical dynamics simulations of reactive scattering processes; its essence is the integration of a special system of coupled ordinary differential equations as an initial value problem. The program is dominated by Gauss-Jordan matrix inversion with partial pivoting and, on some machines, I/O. In addition, matrix multiplication is necessary. The matrices on disk are 67 by 67; the number of coupled ODEs integrated is 65, so the last two rows and columns of the matrices on disk are not used. One matrix is read from disk for each of 300 integration steps. The benchmark uses an unformatted version of the data set so that the I/O is as fast as possible. Note the the input data file which is available to the public for running this benchmark is formatted for portability. It must be converted to unformatted data for the target machine in order that the I/O times are meaningful. Conversion software is a part of the distribution.

The distributed memory matrix inverted is based on a distributed rank-one update. The matrices are partitioned among processors in blocks. The pivot rows and multipliers are passed at each step to the appropriate processors. Each processor carries out a rank-one update on its local data set. Details can be found in [Hipes:88a].

Results for LOGD are shown in Table 10 and Figure 12. The vector supercomputers show the best performance, followed by the SCS-40 and shared-memory machines, with the hypercubes coming up last. The supercomputers vectorized the Fortran SAXPY - the heart of the matrix inverter - easily and well, and coupled with very fast I/O, the CRAY X-MP ran the original Fortran at 49 Mflops (with bidirectional memory enabled) and the assembly inverter version yielded 81 Mflops. The SCS-40 ran about 6 times slower than the CRAY X-MP, which is slightly slower than the ratio of clock speeds. The CRAY-2 times were around twice the X-MP times; I/O on the CRAY-2 was 7 times slower than on the X-MP and accounted for about 10% of the total time.

LOGD Benchmark Results			
Machine	Comments	Time/sec	Mflops*
CRAY X-MP	opt. assembler, CFT77 2.0	12.7	80.7
CRAY X-MP	opt. Fortran, CFT77 2.0	16.0	65.5
CRAY X-MP	orig. Fortran, CFT77 2.0	21.3	49.2
CRAY-2	opt. assembler, CFT77 2.0	25.6	40.0
CRAY-2	opt. Fortran, CFT77 2.0	30.7	34.1
CRAY-2	orig. Fortran, CFT77 2.0	32.5	32.2
SCS-40	opt. assembler, CFT 1.13	53.6	19.1
SCS-40	opt. Fortran, CFT 1.13	98.6	10.6
SCS-40	orig. Fortran, CFT 1.13	124.2	8.44
Alliant FX/8	Fortran, 8P, CD, -Ogvc	160	6.55
Alliant FX/8	Fortran, 8P, CD, -Ogc	347	3.02
Alliant FX/8	Fortran, 4P, CD, -Ogvc	218	4.81
Alliant FX/8	Fortran, 4P, CD, -Ogc	569	1.84
Alliant FX/8	Fortran, 1P, CD, -Ogv	535	1.96
Alliant FX/8	Fortran, 1P, -Og	2305	0.46
Symmetry	Fortran, 23P, CD, -K, Weitek	538	1.95
Symmetry	Fortran, 23P, CD, -K	685	1.53
Symmetry	Fortran, 12P,CD, -K, Weitek	693	1.51
Symmetry	Fortran, 12P,CD, -K	955	1.10
Symmetry	Fortran, 1P, Weitek	4630	0.226
Symmetry	Fortran, 1P	7198	0.146
Balance	Fortran, 23P, CD, -K	2370	0.44
Balance	Fortran, 12P, CD, -K	2894	0.362
Balance	Fortran, 1P	20600	0.051
Multimax	Fortran, 19P, epf -O -fpa	1560	0.672
Multimax	Fortran, 10P, epf -O -fpa	2120	0.494
Multimax	Fortran, 1P, epf -O -fpa	14500	0.072
Mark III	16P, C, CrOS	2490	0.420
Mark III	8P, C, CrOS	3240	0.323
Mark III	4P, C, CrOS	4650	0.225
Mark III	2P, C, CrOS	8120	0.129
Mark III	1P, C, CrOS	14400	0.073
NCUBE	16P, C, Cubix	4590	0.228
NCUBE	8P, C, Cubix	5200	0.201
NCUBE	4P, C, Cubix	7210	0.145
NCUBE	2P, C, Cubix	11800	0.089
NCUBE	1P, C, Cubix	MEM	MEM

* Based on CRAY X-MP performance monitor.

Table 10: LOGD times.

The multiprocessor speedups are far from ideal; the matrices are smaller (65 by 65) than any used in the matrix multiplication benchmark and unlike multiplication, the outermost loop in inversion is inherently sequential because of the pivot search. The shared memory machines had relatively small concurrent tasks in the rank-one update (two nested

loops) and are less efficient than on matrix multiplication (three nested loops). The Alliant and Sequent machines ran with the middle loop concurrent. The concurrent task is therefore approximately (65/number of processors) SAXPY instructions. The efficiency of the 8 processor Alliant is 42% with vectorization and 83% without vectorization; its observed speed is nearly 10 Mflops. For the 12 and 23 processor Sequent Symmetry, efficiency was 56% and 37%, respectively when using the Weitek floating point processor. Without using the Weitek, efficiencies of 63% and 46% were observed on 12 and 23 processors. Both Sequent machines show performance in the range of 2 to 3 Mflops with 23 processors. Sequent Balance displayed efficiencies very similar to those of the Symmetry/Weitek machines and somewhat less than a Mflop speed because of the slower processor. Figure 12 shows that the 23 processor Balance has a higher parallel efficiency than that corresponding to NCUBE.

A note about the Sequent machines; the maximum record length is 512 bytes for a formatted file, a fact which interferes with the use of the input data set. Neither the CRAYs nor the Alliant suffered from this inconvenience.

The best performance of a multiprocessor machine was that of the Alliant FX/8; its Fortran code again showing a performance very close to that of optimized Fortran on the SCS-40. Vectorization increased the speed by 2.2 on 8 processors, by 2.6 on 4 processors, and by 4.3 on one processor.

For the 12 and 23 processor Sequent Symmetry, efficiency was 56% and 37%, respectively when using the Weitek floating point processor. Without using the Weitek, efficiencies of 63% and 46% were observed on 12 and 23 processors. Sequent Balance displayed efficiencies very similar to those of the Symmetry/Weitek machines.

The hypercubes suffered from small grain sizes so that only a small number of processors were effective and again from I/O via the Host. To reach good hypercube performance, the number of unknowns or equivalently the matrix dimensions must be be larger. Future work in chemical dynamics calculations of this sort will require larger matrices. In general, parallel inefficiency due to small matrix sizes is inherent in the computation and can only be addressed by decreasing the ratio of communication to computation time. I/O is a different story. Propagating many independent systems of ODEs simultaneously reduces the ratio of I/O to computation for any size problem. Typical chemical dynamics calculations do many independent propagations representing different collision energies, so this is a viable tool to reduce the impact of I/O. Use of the energy parameter to reduce the impact of I/O has been implemented in the production version of LOGD (not the benchmark).

The times reported are total times including all data loading from disk to hypercube processors. Cubes with more than 16 processors showed no further speedup for the benchmark; a calculation with larger matrices will show speedup on more processors. In fact the 16 processor Mark III shows an effective efficiency of 36%, while the 8 processor Mark III shows 56% efficiency. The corresponding NCUBE efficiencies are somewhat lower because of slower communication channels. Specifically, 42% of the total 16 node Mark III time consists of I/O. For the 16 node NCUBE, I/O took 50% of the total time. Clearly, I/O via the host is not a strength of the current hypercubes. The NCUBE has a parallel disk farm which was not used because of time constraints.

The NCUBE presented a serious difficulty in the LOGD benchmarking. The host

permits host-programs no larger than 64 Kbytes. It was impossible to store two copies of a single 67 by 67 double precision matrix needed for a propagation step. The first copy is the matrix as it exists on disk and the second is a decomposed version for transmission to the cube. This strategy for loading the cube is optimum for the host to nodes part of the I/O. Obviously, a great sacrifice in performance would accompany the use of small data packets in the host to node communication. There is clearly a mismatch between the 64 Kbytes memory model of the host and the 512 Mbyte memory of the 1024 processor configuration of the NCUBE/10. The Sun front end which recently became available for the NCUBE remedies this problem.

As for optimization of LOGD, only the vector machines showed significant improvement. The vector machines optimized the Fortran code by activating bidirectional memory, by unrolling one loop in the matrix multiplier, and by using assembly coded inverters and multipliers. None of these features were relevant for the shared memory machines or hypercubes.

6.6 Vortex

Vortex is a fluid dynamics simulation code written in Fortran. It consists of essentially two independent loops: one initialization loop and a doubly nested loop; the latter taking up most of the time. This code originated on the Cyber-205 and one version has Cyber-205 compiler directives in the source. This version is vectorized. Another version was produced for runs on a SUN workstation; Newton's third law could be exploited to reduce the number of iterations by a factor of 2. This alteration does not help the vector machines but is useful on scalar machines. Clearly the vector computers will run the original code fastest but it is interesting to see which version is most efficient on the multiprocessor machines.

The distributed memory algorithm is a based on distributing the vortices among processors. Vortices interact via long range forces so all processors must exchange data.

The results for VORTEX are shown in Table 11 and Figure 13.

The compilers for the CRAY X-MP, CRAY-2, and SCS-40 vectorized the original Fortran and showed speeds of 162, 175, and 29 Mflops. Bidirectional memory gave a slight boost to the X-MP performance; otherwise, the code permitted no room for optimization. This is reasonable for the X-MP and SCS-40 given the high level of performance observed. The CRAY-2 is well below its peak speed of 488 Mflops and the X-MP is running at 75% of its peak speed.

The ETA10-E ran the version with Cyber-205 compiler directives at an impressive 227 Mflops. The Cyber-205 yielded 122 Mflops. Without the in-source compiler directives, speeds of 148 and 79 Mflops were observed for the ETA10-E and Cyber-205, respectively. Clearly, compiler directives are necessary for effective use of these two machines. Interestingly, the code written for the SUN and with fewer loop iterations ran the slowest of all the versions of this code on the ETA10-E, running at 28 Mflops. ETA10-E performance was hurt by the decreasing length of the inner (vectorized) loop.

One shared memory machine was benchmarked with Vortex. The 8 processor configuration of the Alliant FX/8 is again about a factor of 10 slower than the traditional vector machines. A speed of 12 Mflops was obtained on the original code (ignoring Newton's third law) with a multiprocessor efficiency of 54%. The vectorization on 8 processors increased

Fortran-Vortex Benchmark Results (5000 vortices and 10 time steps)			
Machine	Comments	Time/sec	Mflops*
CRAY X-MP	opt code2, CFT77 2.0	75.5	169
CRAY X-MP	orig code2, CFT77 2.0	79.0	162
CRAY-2	orig code2, CFT77 2.0	73.0	175
SCS-40	orig code2, CFT 1.13	436	29.2
ETA10-E	orig code1	56.1	227
ETA10-E	orig code2	86.3	148
ETA10-E	orig code3	912	28.0
Cyber 205	orig code1	105	122
Cyber 205	orig code2	162	78.9
Alliant FX/8	1P, -AS, code2	27140	0.477
Alliant FX/8	1P, -Ogv -AS, code2	4510	2.83
Alliant FX/8	8P, -Ogc -AS, code2	2380	5.37
Alliant FX/8	8P, -Ogvc -AS, code2	1050	12.2
Alliant FX/8	1P, -AS, code3	17100	
Alliant FX/8	1P, -Ogv -AS, code3	2820	
Alliant FX/8	8P, -Ogc -AS, code3	1110	
Alliant FX/8	8P, -Ogvc -AS, code3	383	
NCUBE	256P, CrOS	358	
NCUBE	128P, CrOS	673	
NCUBE	64P, CrOS	1280	
NCUBE	32P, CrOS	2500	
NCUBE	16P, CrOS	4950	
NCUBE	8P, CrOS	9880	
NCUBE	4P, CrOS	20210	
NCUBE	2P, CrOS	40590	
NCUBE	1P, CrOS		

* Based on CRAY performance monitor.
code1 has Cyber compiler directive
code2 without Newton's law N^2
code3 using Newton's law $N^2/2$

Table 11: VORTEX times.

the speed by 2.3 over the nonvectorized version. The code with reduced loop iterations required a factor of 2.7 less time on the Alliant. On the original code, the Alliant was 2.4 times slower than the SCS-40. This is a somewhat less favorable comparison for the Alliant than for some of the other benchmarks. The best Alliant execution times are 3.5, 6.8, 5.2, and 5.1 times longer than the those corresponding to the Cyber-205, ETA10-E, CRAY-2, CRAY X-MP, respectively.

The only distributed memory machine on which Vortex was run is the NCUBE. With 256 processors, the NCUBE required nearly the same time as the Alliant FX/8 on the SUN version of the code. The multiprocessor efficiency of the NCUBE code is 44%. Increasing the number of vortices would favor the NCUBE by increasing its efficiency. In contrast, the X-MP performance has little room for improvement. Peak hypercube performance for this algorithm requires increased problem size, not further code tuning.

6.7 1D Plasma simulation - BEPS1

The BEPS1 code has been benchmarked at Caltech on two machines, the NCUBE and Mark IIIfp, and we present the results of Decyk [Decyk:87a] in the cases where he

benchmarked machines in our standard set, these being an 8-node Alliant, the CRAY XMP, CRAY-2 and SCS-40.

The implementation of plasma PIC simulation codes on hypercube concurrent computers has been described in [Liewer:88b]. Two different decompositions are used; one to update the particle positions and velocities, and another to solve the electromagnetic field equations. In the first of these decompositions, the spatial domain of the computation is divided so that initially each processing node is responsible for approximately equal numbers of plasma particles. In the second decomposition, equal numbers of grid points are placed in each processor. As the system evolves, particles will move from one processing node's sub-domain to another, requiring the data for that particle to be communicated between the two nodes. After a time the number of particles in each node will differ, and hence the load balance will become uneven. When the load imbalance becomes sufficiently large the computational domain is sub-divided anew, so that each node again contains approximately the same number of particles. This technique is called dynamic load balancing. In the version of the code that was benchmarked, dynamic load balancing was not used, since for the problem considered statistical load balance is maintained. Each processing node contains all the data for the particles which lie within its sub-domain. In addition, it contains the electromagnetic field values for the grid points that lie within its sub-domain, and also the values for the grid points lying along the boundaries of the neighboring sub-domains. Thus, each node is able to update all the particles for which it is responsible. When changing from the first decomposition to the second in order to solve for the electromagnetic field, each processor distributes the charge and current densities associated with its particles onto the grid points that it contains. The nodes then communicate to distribute the regular grid over the nodes. After solving for the electric and magnetic fields on this grid, the nodes again communicate so that each node stores the field values appropriate to the first decomposition.

The BEPS1 benchmark deals with a one-dimensional grid of 128 mesh points and a collection of 11264 particles. Inefficiency in a parallel environment arises because load balancing is only statistical, so there are fluctuations in the number of particles per processor. Furthermore, in a distributed-memory environment there is communication overhead as particles need to be moved from processor to processor, and also overhead resulting from the parallel Fourier transform used to calculate the electric field.

BEPS1 results are given in Table 12 and Figure 14.

The maximum number of processors we used in the benchmark was 32; in this case there are about 350 particles per processor, with a standard deviation of about 19, so the inefficiency from the statistical load balancing should be two or three standard deviations at most, or $10 - 15\%$. The real inefficiency, for distributed memory machines, comes from moving the particles from processor to processor, which of course depends on how the simulated plasma actually behaves.

Referring to Figure 14, there are three curves for the Mark III, the slowest being Decyk's result, which was done with the old 68881 floating-point unit. The intermediate Mark III curve is with the newer 68882 unit, giving an extra factor of about 2 on 32 nodes. If in addition the Weitek floating-point units are utilized, there is a further factor of 2 in speed.

BEPS1 (Plasma) Benchmark Results		
Machine	Comments	Time/sec
Mark III	32P, CrOS, C-FFT	431
Mark III	1P, CrOS, C-FFT	7070
Mark IIIfp	32P, CrOS, C-FFT	251
Mark IIIfp	4P, CrOS, C-FFT	598
Mark IIIfp	1P, CrOS, C-FFT	1580
NCUBE	32P, CrOS, For-FFT	1170
NCUBE	16P, CrOS,For-FFT	1580
NCUBE	8P, CrOS,For-FFT	2570
NCUBE	4P, CrOS,For-FFT	4490
NCUBE	1P, CrOS,For-FFT	14700

Table 12: Results for the plasma benchmark BEPS1.

Notice that the CRAY results are benchmarked with an old version of the Fortran compiler, so these would be slightly faster with the more up-to-date CFT77. The Alliant time - shown a single blob in Figure 14 - is from [Decyk:87a] and corresponds to 8 processors. The straight lines in Figure 14 for the Mark III 68881 and 68882 are perfectly straight because the only points are for one and 32 processors.

7. Optimization Times

A goal of our study is to quantify the programmer effort expended to optimize the codes from the benchmark suite. Quantification of effort is a difficult issue for these codes; the hypercube codes are the result of months or years of effort by experts and as a result the programs are hard to classify as generic. Furthermore, the inherent specialization in the hypercube codes left little room for optimization on the time scale of a week. In contrast, the sequential codes that were run on the vector supercomputers and the shared-memory machines can be labeled generic with much less uncertainty. (Although, as mentioned in the introduction, there is not such item as the generic program.) To complicate matters further, the CRAY and SCS-40 codes were optimized by experienced CRAY-center personel, while many of the shared-memory runs were optimized by non-experts on these machines. For example, should the hours spent in the manual be recorded as optimization effort? Such was not necessary for one set of machines but required for others because of differing levels of experience among the authors. In the case of the CRAYs and SCS-40, the SDSC group kept meticulous records of their efforts and these represent meaningful and interesting optimization requirements because such centers provide such expert consulting to all users. More details on the SDSC work can be found in their report [Pfeiffer:88a].

On the positive side, the optimization times are uniquely practical measures of the effort of a real group with widely varying experience such as might be found in or available to any lab or computation-oriented business at the present time. The programmer's time to achieve a functioning executable is given in Table 13a. The time spent optimizing the codes is given in Table 13b.

Three general types of optimization occurred in our study. The CRAYs, SCS-40, Alliant, Sequent, and Encore all have compilers or precompilers for locating and exploiting vectorization and when applicable, parallelization. For these machines, the optimization

Initial effort to get a running code						
Machine	QCD	Tracker	Chem. Dyn.	Vortex	LU Decomp	MMULT
CRAY X-MP/48	2h	1h	origin	1h	8h	8h
CRAY-2	< 1h	< 1h	< 1h	< 1h	< 1h	< 1h
ETA10-E	1d	1d	ND	origin	1d	1d
Cyber 205	1d	1d	ND	origin	1d	1d
Alliant FX/8	1d	1d	1h		< 1h	<1h
Both Sequents	1w	1w	1d		<1h	<1h
Multimax			1h		<1h	<1h
Mark III	1d	origin	origin		2h	origin
NCUBE/10	origin	1d	4w		origin	<1d
CM2	4w		ND	>6w	5w	4w

Effort to optimize code						
Machine	QCD	Tracker	Chem. Dyn.	Vortex	LU Decomp	MMULT
CRAY X-MP/48	42 h	45 h	18 h	8 h	32 h	24 h
ETA10-E	0	0	ND	origin	1w	1w
Cyber 205	0	0	ND	origin	1w	1w
Alliant FX/8	2w	0	2w		2w	3d
Both Sequent	4w	4w	2w		2w	2d
Multimax			1d		1d	<1d
Mark III	0	origin	origin		0	origin
NCUBE/10	origin	0	0		origin	0
CM2	0		ND		0	2w
Cydra						

An entry of 0 means that the code was recompiled with an optimization switch on or no optimization was done.
The entry origin means that the code originated on that machine.
ND implies not done.

Table 13: Time spent in each phase of the benchmarking study.

consists of inserting straighforward compiler directives or reordering/reorganizing nested loops to aid the compiler. Such effort is measured in hours or days depending on the experience of the programmer. In some cases (TRACKER on the Sequent Symmetry) a code (originally on the hypercubes) is completely rewritten for the machine and achieves superior performance to code that was not so completely rewritten (same calculation on Alliant with a sequential Fortran code). Such effort is measured in weeks or months. The final type of optimization is that required for many applications on a hypercube; frequently the program must be thoroughly reorganized and message passing structures must be inserted. The effort on modest size codes is measured in months. Such optimization did not take place in our study; all of the benchmark codes had hypercube implementations and, in most cases, sequential implementations. Summing up, the programmer effort tables reflect the time spent on extracting performance from the vector and shared-memory machines where optimization tools are well developed.

The Connection Machine is a special case in our study: most programs (matrix multiplication and LU decomposition) did not exist prior to the beginning of this project. In order to get good performance on the CM-2, applications must be written in *Lisp and there are few if any tools like precompilers to port existing codes to it. In this respect,

the Connection Machine is in the same league as the hypercubes. However, the MIMD hypercubes do have an advantage over the CM-2 in the avaiability of compilers for C and Fortran for single nodes, so blocks of standard sequential code can be run on a hypercube processor.

8. Future Work

(1) In the future we expect to extend our performance analysis work further to include more applications and machines. We also intend to develop detailed performance models of the machines studied.

(2) Another possible development is the design and implementation of a portable programming environment for advanced architecture multiprocessors. This would allow applications to be ported easily between, for example, hypercubes and shared memory multiprocessors with the minimum amount of performance degradation. Specifications for such an environment have already been made [Walker:88d].

(3) We also intend to develop real-time graphical performance monitors. This work has begun on the NCUBE and is described in [Morison:88a].

(4) In the case of the TMC Connection Machine 2, we would have liked to have run versions of the code using the *Lisp programming language and the PARIS assembly-coded subroutine library, but had insufficient time to do this.

(5) We encourage others to optimize the benchmark codes on their favorite machines. It is important to keep a record of the changes to the code, and the time and effort expended.

(6) We would like to run our suite of codes on the Symult S2010, Intel iPSC/2, Meiko Computing Surface, Evans and Sutherland ES-1, and the Myrias SPS-2.

Acknowledgments: The support of the Department of Energy under grant number DE-FG03-85ER25009 and of the National Science Foundation under grant numbers ASC-8719501 and ASC-8719502 (at SDSC), is gratefully acknowledged. Victor Decyk and Paulet Liewer were kind enough to allow us to use some of their timing results for BEPS1. Access to the Argonne Advanced Computing Research Facility greatly facilitated doing experiments on a wide variety of computer architectures. Access to the CRAY-2 at Air Force Supercomputer Center at Kirtland Air Force Base (AFSCC-K) is also gratefully acknowledged. The Rockwell Science Center in Thousand Oaks made available their Butterfly for timing runs and Florida State University made available their ETA10-E to us. The following Caltech undergraduates participated by making runs and adapting codes to different machines: Ken Barish, Huy Cao, Curt Hagenlocher, and Jonathan Miller. Finally, we thank the authors of the programs for allowing us to use them in this project: A. Leonard for VORTEX, T. Gottschalk for TRACKER, and S. Otto and J. Flower for the QCD code.

References

[Aldcroft:88a] Aldcroft, T., Cisneros, A., Fox, G. C., Furmanski, W., and Walker, D. W. "LU decomposition of banded matrices and the solution of linear systems on hypercubes," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1635–1655. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-348b.

[Appel:85a] Appel, A. W. "An efficient program for many-body simulation," *Sci. Stat. Comput.*, 6:85, 1985.

[Aref:83a] Aref, H. "Integrable, chaotic and turbulent vortex motion in two-dimensional flows," *Ann. Rev. Fluid Mech.*, 13:345, 1983.

[Baillie:88f] Baillie, C. F., Gottschalk, T. D., and Kolawa, A. "Comparisons of concurrent tracking on various hypercubes," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 155–166. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-568.

[Baillie:88g] Baillie, C. F., and Walker, D. W. "Lattice QCD — as a large scale scientific computation." Technical Report C3P-641, California Institute of Technology, June 1988. To be published in Proceedings of International Conference on Vectors and Parallel Computing.

[Baillie:88j] Baillie, C. F. "Using a shared memory computer as a distributed memory computer." Technical Report C3P-658, California Institute of Technology, September 1988.

[Baillie:88p] Baillie, C. F., and Felten, E. W. "Benchmarking concurrent supercomputers," in E. Gelenbe, editor, *High Performance Computer Systems*. Elsevier, Amsterdam, 1988. Caltech Report C3P-453.

[Barnes:86a] Barnes, J., and Hut, P. "A hierarchical $O(N \log N)$ force calculation algorithm," *Nature*, 324:446, 1986.

[Berry:88a] Berry, M., Chen, D., Koss, P., Kuck, D., Lo, S., Pang, Y., Roloff, R., Sameh, A., Clementi, E., Chin, S., Schneider, D., Fox, G., Messina, P., Walker, D., Hsiung, C., Schwarzmeier, J., Lue, K., Orszag, S., Seidl, F., Johnson, O., Swanson, G., Goodrum, R., and Martin, J. "The perfect clube benchmarks: Effective performance evaluation of supercomputers." Technical Report CSRD Rpt. No. 827, The Performance Evaluation Club (Perfect), November 1988.

[Brickner:89a] Brickner, R., and Baillie, C. "Pure gauge QCD on the Connection Machine." Technical Report C3P-710, California Institute of Technology, February 1989.

[Brooks:84a] Brooks, E. I., Fox, G. C., Johnson, M., Otto, S. W., Stolorz, P., Athas, W., DeBenedictis, E., Faucette, R., Seitz, C., and Stack, J. "Pure gauge SU(3) lattice gauge theory on an array of computers," *Phys. Rev. Lett.*, 52:2324, 1984. Caltech Report C3P-065.

[Cabbibo:82a] Cabbibo, N., and Marinari, E. "A new method of updating SU(N) matrices in computer simulations of gauge theories," *Phys. Lett. B.*, 119:387, 1982.

[Cao:88a] Cao, H. T., and Baillie, C. F. "Caltech missile tracking program — a benchmark comparison: NCUBE and T800 vs. sequent balance and symmetry." Technical Report C3P-673, California Institute of Technology, October 1988.

[Catherasoo:87a] Catherasoo, C. J. "The vortex method on a hypercube concurrent processor," in M. T. Heath, editor, *Hypercube Multiprocessors.* SIAM, Philadelphia, 1987.

[Chorin:73a] Chorin, A. J. "Numerical study of slightly viscous flow," *J. Fluid Mech.*, 57:785, 1973.

[Christiansen:73a] Christiansen, J. P. "Numerical solution of hydrodynamics by the method of point vortices," *J. Comp. Phys.*, 13:363, 1973.

[Decyk:87a] Decyk, V. K. "Revised benchmark timings with particle plasma simulation codes." Technical Report PPG-1111, University of California, Los Angeles, 1987.

[Ding:88c] Ding, H. Q. "Performance of a QCD code on the Mark IIIfp." Technical Report C3P-624, California Institute of Technology, April 1988.

[Dongarra:82a] Dongarra, J. J., Gustavson, F. G., and Karp, A. "Implementing linear algebra algorithms for dense matrices on a vector machine." Technical Report ANL/MCS-TM-1, Argonne National Laboratory, September 1982. Argonne Technical Memorandum.

[Dongarra:87b] Dongarra, J. J., Martin, J. L., and Worlton, J. "Computer benchmarking: Paths and pitfalls," *IEEE Spectrum*, July 1987.

[Dongarra:88b] Dongarra, J. J. "Performance evaluation of various computers using standard linear equations software in a fortran environment." Technical Report MCS-TM-23, Argonne National Laboratory, 1988.

[Flower:85a] Flower, J. W., and Otto, S. "The field distribution in SU(3) lattice gauge theory," *Phys. Lett. B*, 160:128, 1985. Caltech Report C3P-178.

[Flower:87b] Flower, J. *Lattice Gauge Theory on a Parallel Computer.* PhD thesis, California Institute of Technology, 1987. Caltech Report C3P-411.

[Flower:88b] Flower, J., Apostolakis, J., Baillie, C., and Ding, H. Q. "Lattice gauge theory on the hypercube," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1278–1287. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-605.

[Fox:85b] Fox, G., Hey, A. J. G., and Otto, S. "Matrix algorithms on the hypercube i: Matrix multiplication," *Parallel Computing*, 4:17, 1987. Caltech Report C3P-206.

[Fox:88a] Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K., and Walker, D. W. *Solving Problems on Concurrent Processors*, volume 1. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1988.

[Furmanski:87a] Furmanski, W., Bower, J. M., Nelson, M. E., Wilson, M. A., and Fox, G. "Piriform (Olfactory) cortex model on the hypercube," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 977–999. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-404b.

[Gottschalk:87c] Gottschalk, T. D. "A new multi-target tracking model." Technical Report C3P-480, California Institute of Technology, October 1987.

[Gottschalk:88a] Gottschalk, T. D. "Concurrent multiple target tracking," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1247–1268. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-567.

[Greengard:86a] Greengard, L., and Rokhlin, V. "A fast algorithm for particle simulation." Technical Report YALEU/DCS/RR-459, Yale University, 1986.

[Gupta:88c] Gupta, R., Kilcup, G. W., Patel, A., Sharpe, S. R., and DeForcrand, P. "Comparison of update algorithms for pure gauge SU(3)," *Mod. Phys. Lett.*, 3:1367, 1988.

[Harstad:87a] Harstad, K. "Performance of vortex flow simulation on the hypercube." Technical Report C3P-500, California Institute of Technology, October 1987.

[Hipes:88a] Hipes, P. G., and Kuppermann, A. "Gauss-Jordan inversion with pivoting on the Caltech Mark II hypercube," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1621–1634. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-578.

[Hipes:88b] Hipes, P., Mattson, T., Wu, M., and Kuppermann, A. "Chemical reaction dynamics: Integration of coupled sets of ordinary differential equations on the Caltech hypercube," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1051–1061. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-570.

[Johnson:73a] Johnson, B. R. "The multi-channel log-derivative method for scattering calculations," *Journal of Computational Physics*, 49:23, 1973.

[Johnson:77a] Johnson, B. R. "New numerical methods applied to solving the one-dimensional eigenvalue problem," *J. Chem. Phys.*, 67:4086, 1977.

[Johnsson:88a] Johnsson, S. L. "Data parallel programming and basic linear algebra subroutines," in J. R. Rice, editor, *Mathematical Aspects of Scientific Software*, pages 183–196. The IMA Volumes in Mathematics and Its Applications, Springer-Verlag, 1988. Volume 14.

[Kuppermann:75a] Kuppermann, A. "A useful mapping of the triatomic potential energy surface," *Chem. Phys. Letters*, 32:374, 1975.

[Leonard:80a] Leonard, A. "Vortex methods for flow simulation," *J. Computational Physics*, 37:289, 1980.

[Leonard:85a] Leonard, A. "Computing three-dimensional incompressible flows with vortex elements," *Ann. Rev. Fluid Mech.*, 17:523, 1985.

[Leonard:88a] Leonard, A., and Chua, K. "Three-dimensional interactions of vortex tubes," in *Proc. Symposium on Advances in Fluid Turbulence*, May 1988. Los Alamos. To appear in Physica D. Caltech Report C3P-691.

[Liewer:88b] Liewer, P. C., Decyk, V. K., Dawson, J. D., and Fox, G. C. "A universal concurrent algorithm for plasma particle-in-cell simulation codes," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1101–1107. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-562.

[Liewer:89a] Liewer, P. C., Zimmerman, B. A., Decyk, V. K., and Dawson, J. M. "Application of hypercube computers to plasma particle-in-cell simulation codes." Technical Report C3P-717, California Institute of Technology, 1989. Paper presented at the Fourth International Conference on Supercomputing, Santa Clara.

[Martin:87a] Martin, J. L., and Mueller-Wichards, D. "Supercomputing performance evaluation: Status and directions," *The Journal of Supercomputing*, 1, May 1987.

[Morison:88a] Morison, R. "Interactive performance display and debugging using the NCUBE real-time graphics system," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 760–765. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-576.

[NAS:86a] "An agenda for improved evaluation of supercomputer performance." National Academy Press, Washington, D. C., 1986. Report to the National Academy of Sciences.

[Otto:83b] Otto, S. *Monte Carlo Methods in Lattice Gauge Theories.* PhD thesis, California Institute of Technology, 1983.

[Otto:84a] Otto, S. W., and Stack, J. D. "SU(3) heavy-quark potential with high statistics," *Phys. Rev. Lett.*, 52:2328, 1984. Caltech Report C3P-067.

[Otto:84c] Otto, S. W., and Stolorz, P. "An improvement for glueball mass calculations on a lattice." Technical Report C3P-101, California Institute of Technology, 1984. See Otto:85b for publication.

[Otto:85b] Otto, S. W., and Stolorz, P. "An improvement for glueball mass calculations on a lattice," *Physics Letters B*, 151(5,6):428, February 1985. Caltech Report C3P-343.

[Otto:87a] Otto, S. W., Baillie, Clive, F., Ding, H.-Q., Apostolakis, J., Gupta, R., Kilcup, G., Patel, A., and Sharpe, S. "Lattice gauge theory benchmarks." Technical Report C3P-450, California Institute of Technology, 1987.

[Pfeiffer:88a] Pfeiffer, W., Alagar, A., Kamrath, A., Leary, R., and Rogers, J. "Benchmarking and optimization of scientific codes on the CRAY X-MP, CRAY-2, and SCS-40 vector computers." Technical Report C3P-699, California Institute of Technology and San Diego Supercomputer Center, November 1988. Submitted for publication in The Journal of Supercomputing.

[Spalert:83a] Spalert, P. R., Leonard, A., and Baganoff, D. "Numerical simulations of separated flows." Technical Report TM-84328, NASA, 1983.

[Spalert:84a] Spalert, P. R. "Two recent extensions of the vortex method." Technical Report 84-0343, AIAA, 1984.

[Walker:88c] Walker, D. W. "Performance of a QCD code on the NCUBE hypercube," in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 1*, pages 180–187. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-490b.

[Walker:88d] Walker, D. W. "Portable programming within a message-passing model: the FFT as an example." in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2*, pages 1438–1450. ACM Press, 11 West 42nd Street, New York, NY 10036, January 1988. Caltech Report C3P-631.

[Wilson:88b] Wilson, M. A., and Bower, J. M. "A computer simulation of the olfactory cortex with functional implications for storage and retrieval of olfactory information," in *Proceedings of the IEEE Conference on Neural Information Processing Systems*. AIP Press, 1988.

Appendix A: Key to Tables

-AS:	Alliant compiler optimization switch enabling associative transformations
assembler:	Some or all of the code in assembler
C:	Source code in the language C
C*:	Source code for CM2 in the language C*
CAL	CRAY assembly language
CD:	In-source compiler directives
C-FFT:	Fast Fourier transform written in C
CFT77:	CRAY Fortran compiler
CFT 1.13:	CRAY Fortran compiler
CrOS:	Caltech crystalline communication routines
Cubix:	CrOS-based host file server for hypercubes
DI:	Direct indexing of arrays replaced indirect indexing in LU
epf:	Parallelizing preprocessor on Encore Multimax
Express:	New generation of CrOS with many additional functions
Fast CrOS :	A fast version of CrOS for NCUBE
For-FFT:	Fast Fourier transform written in Fortran-77
Fortran:	Source code in the language Fortran-77
inner:	Inner product form of matrix multiply
-K:	Kuck and Associates parallelizing preprocessor switch for Sequent
MEM:	Required too much memory
-O[gvc]:	Compiler optimization switch for Alliant FX/8: global, vector, concurrent
opt:	Optimized source without using machine language tuning
orig:	Original source code
outer:	SAXPY (outer product) form of matrix multiply
Paris:	Low-level assembly language for the Connection Machine
#P:	Number of processors
SDSC-Fortran:	A Fortran-77 version optimized by SDSC
too long:	Excessive CPU time required
unroll:	Loop unrolling used
US:	Uniform System operating system for the Butterfly
Weitek:	Used Weitek floating point coprocessor
32b:	32-bit arithmetic used in computations

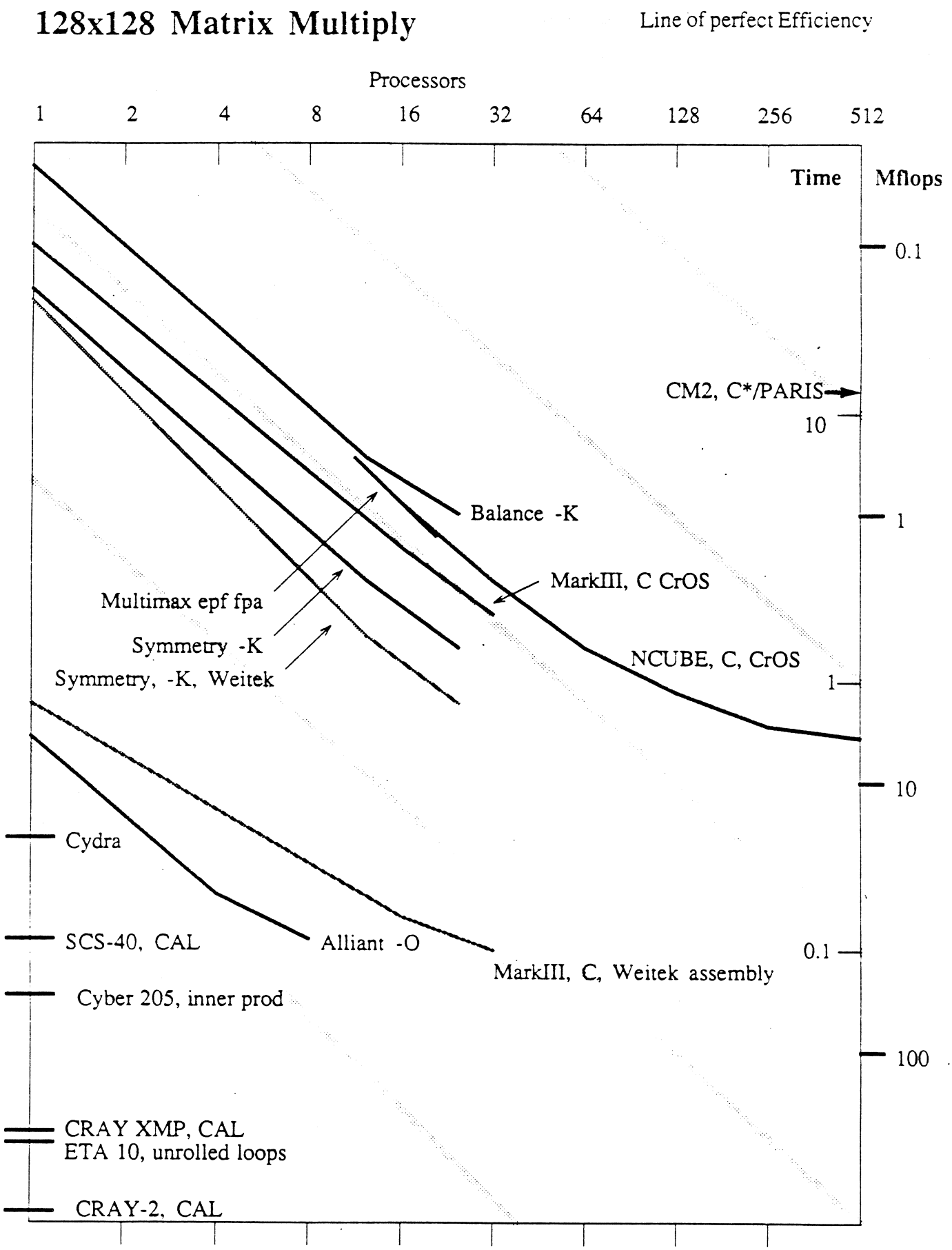

128x128 Matrix Multiply
Line of perfect Efficiency
Processors
1
2
4
8
16
32
64
128
256
512
Time
Mflops
0.1
CM2, C*/PARIS
10
1
Balance -K
MarkIII, C CrOS
Multimax epf fpa
Symmetry -K
Symmetry, -K, Weitek
NCUBE, C, CrOS
1
10
Cydra
SCS-40, CAL
Alliant -O
0.1
MarkIII, C, Weitek assembly
Cyber 205, inner prod
100
CRAY XMP, CAL
ETA 10, unrolled loops
CRAY-2, CAL

Figure 1

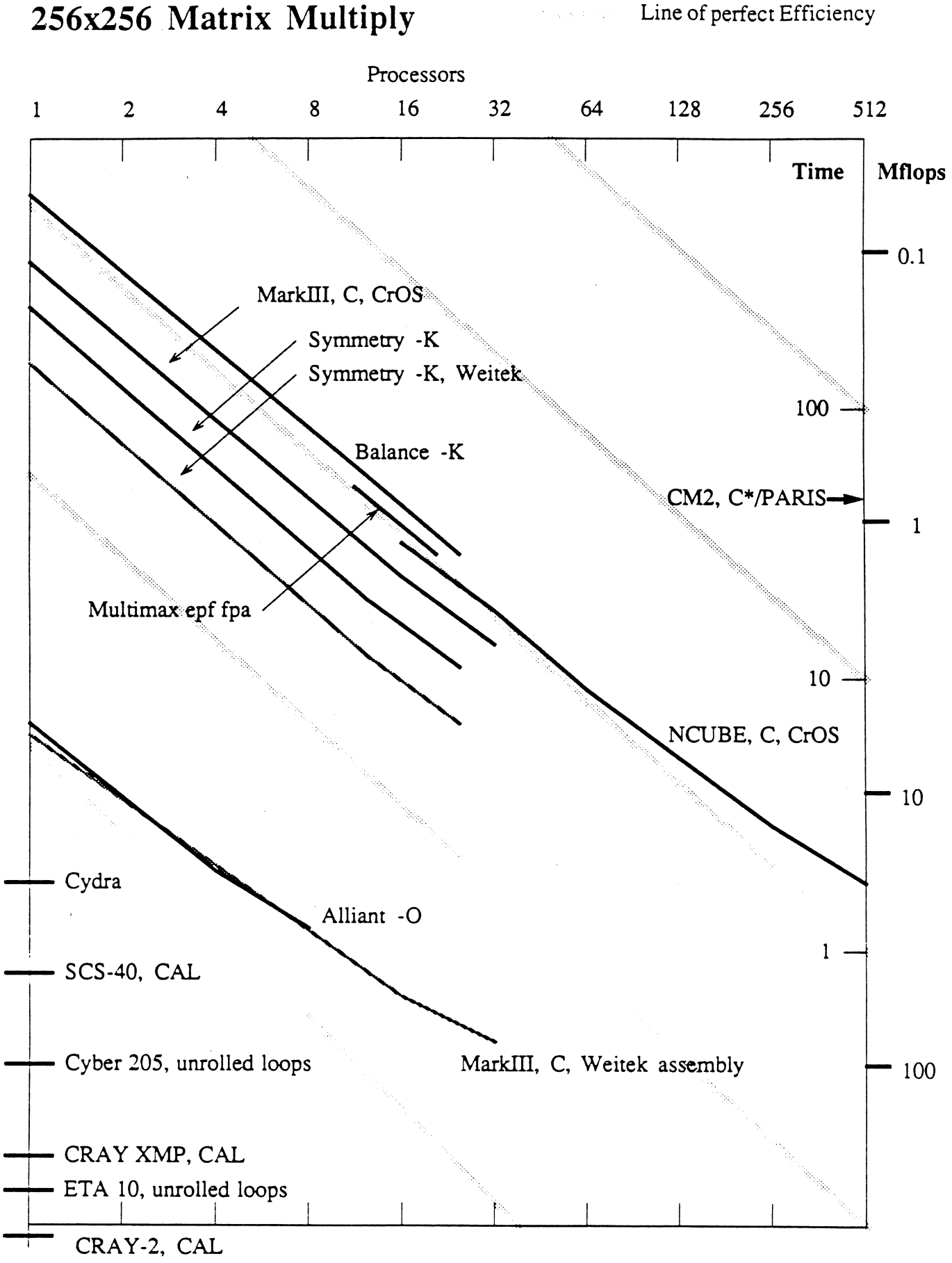

256x256 Matrix Multiply
Line of perfect Efficiency
Processors
1
2
4
8
16
32
64
128
256
512
Time
Mflops
0.1
100
1
10
10
1
100
MarkIII, C, CrOS
Symmetry -K
Symmetry -K, Weitek
Balance -K
CM2, C*/PARIS
Multimax epf fpa
NCUBE, C, CrOS
Cydra
Alliant -O
SCS-40, CAL
Cyber 205, unrolled loops
MarkIII, C, Weitek assembly
CRAY XMP, CAL
ETA 10, unrolled loops
CRAY-2, CAL

Figure 2

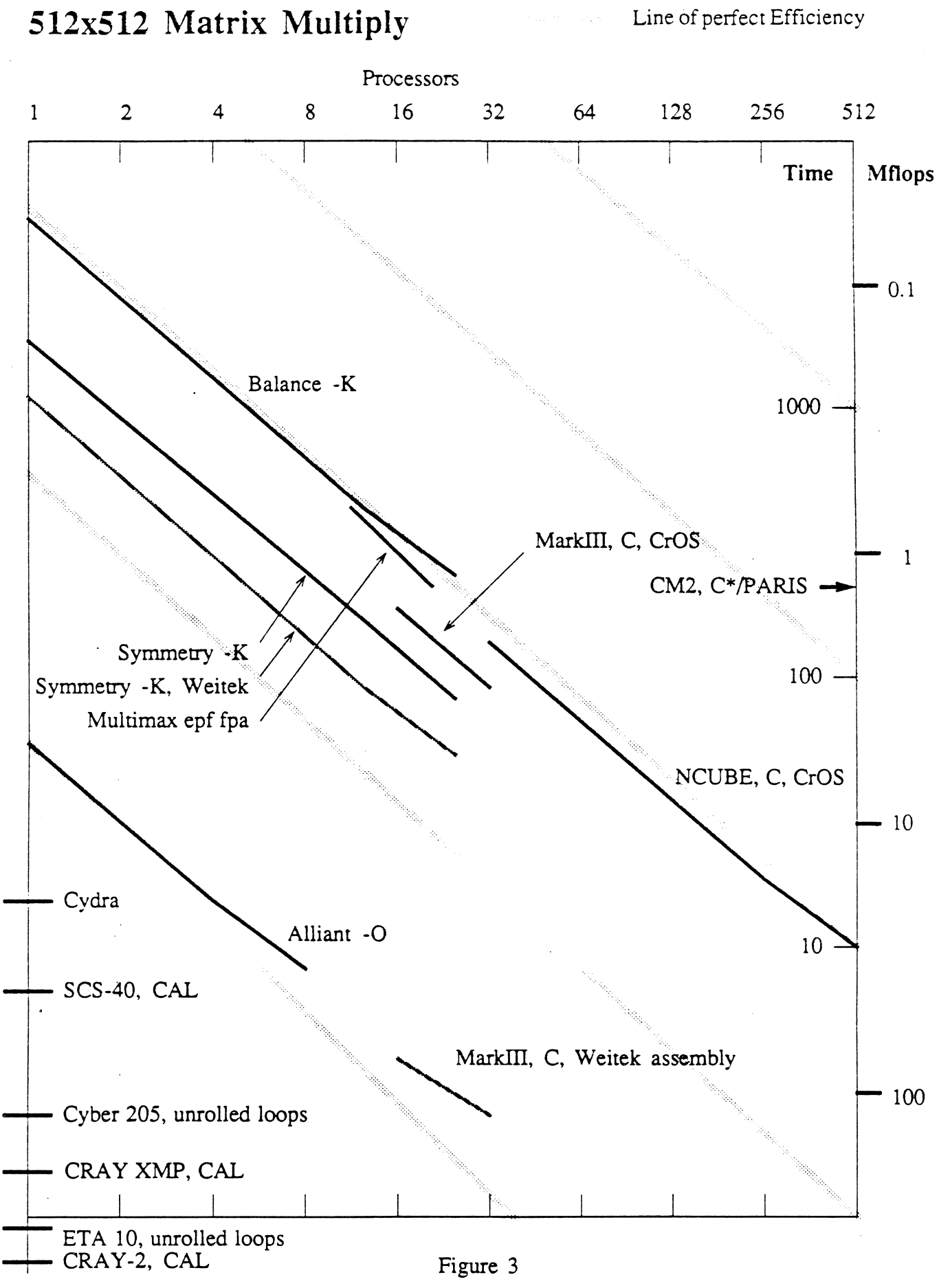

512x512 Matrix Multiply
Line of perfect Efficiency
Processors
1
2
4
8
16
32
64
128
256
512
Time
Mflops
0.1
1000
1
100
10
10
100
Balance -K
MarkIII, C, CrOS
CM2, C*/PARIS
Symmetry -K
Symmetry -K, Weitek
Multimax epf fpa
NCUBE, C, CrOS
Cydra
Alliant -O
SCS-40, CAL
MarkIII, C, Weitek assembly
Cyber 205, unrolled loops
CRAY XMP, CAL
ETA 10, unrolled loops
CRAY-2, CAL

Figure 3

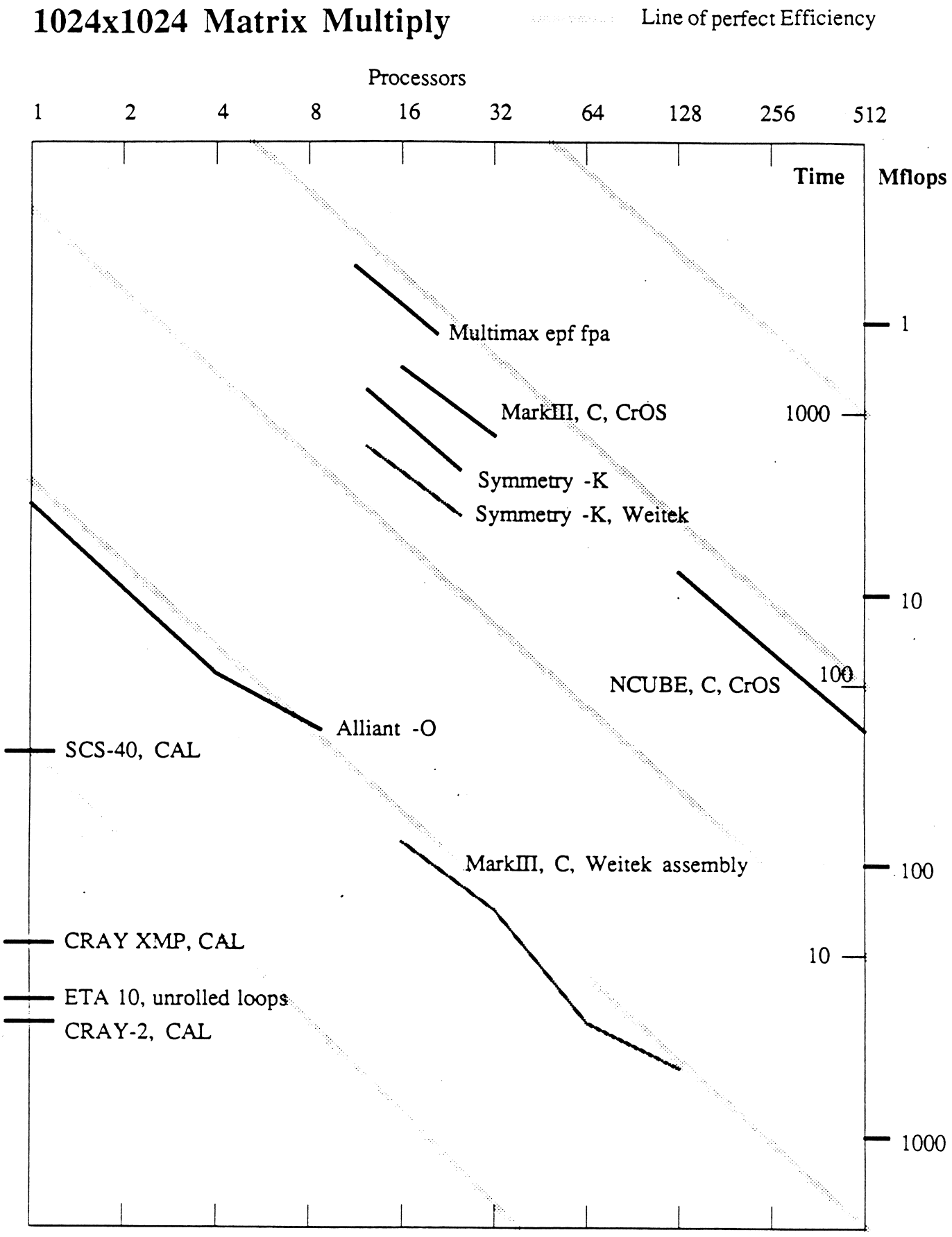

1024x1024 Matrix Multiply
Line of perfect Efficiency
Processors
1
2
4
8
16
32
64
128
256
512
Time
Mflops
Multimax epf fpa
MarkIII, C, CrOS
Symmetry -K
Symmetry -K, Weitek
NCUBE, C, CrOS
Alliant -O
SCS-40, CAL
MarkIII, C, Weitek assembly
CRAY XMP, CAL
ETA 10, unrolled loops
CRAY-2, CAL
1
1000
10
100
100
10
1000

Figure 4

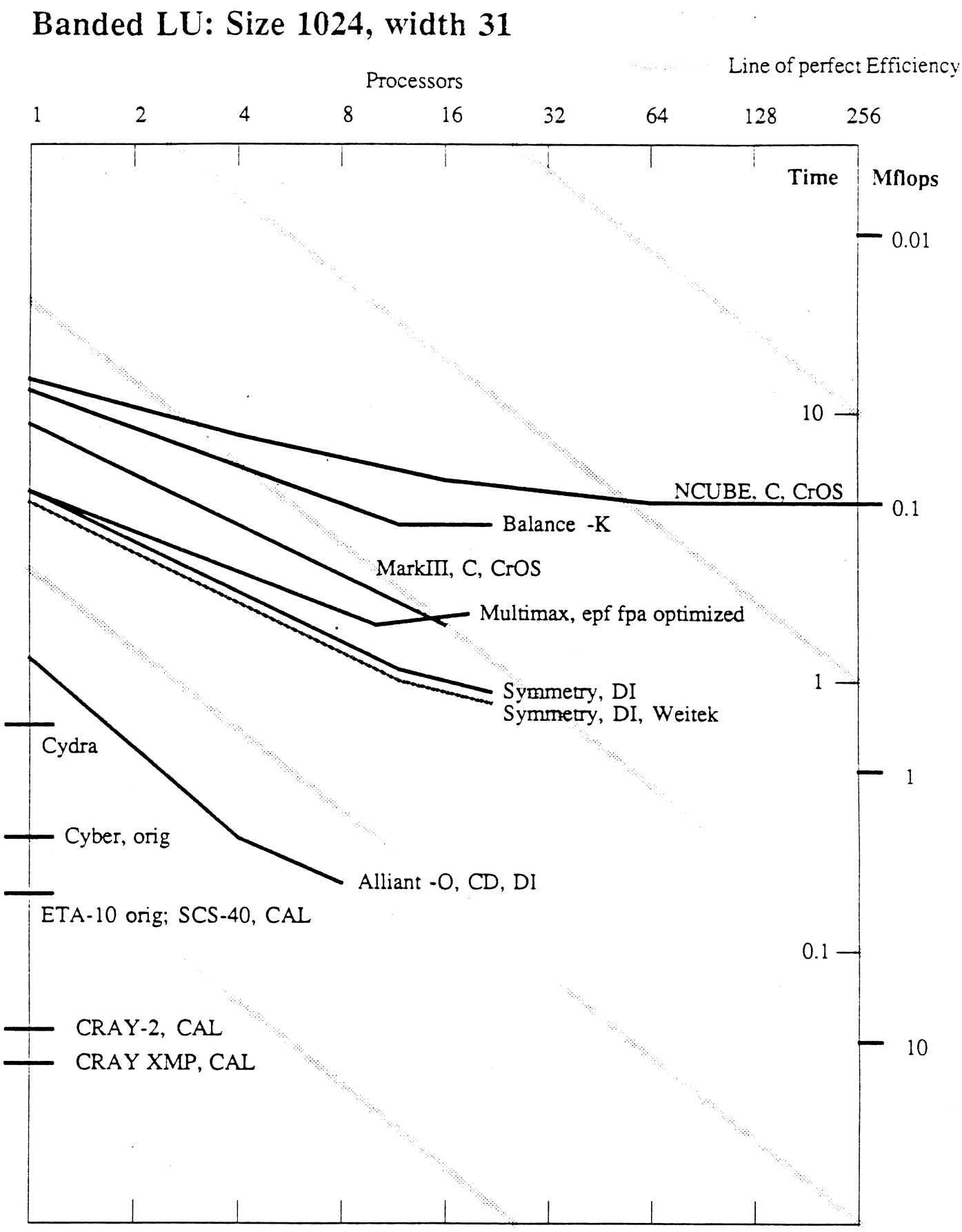

Banded LU: Size 1024, width 31
Line of perfect Efficiency
Processors
1
2
4
8
16
32
64
128
256
Time
Mflops
0.01
10
0.1
NCUBE, C, CrOS
Balance -K
MarkIII, C, CrOS
Multimax, epf fpa optimized
1
Symmetry, DI
Symmetry, DI, Weitek
Cydra
1
Cyber, orig
Alliant -O, CD, DI
ETA-10 orig; SCS-40, CAL
0.1
CRAY-2, CAL
10
CRAY XMP, CAL

Figure 5

Banded LU: Size 1024, width 127

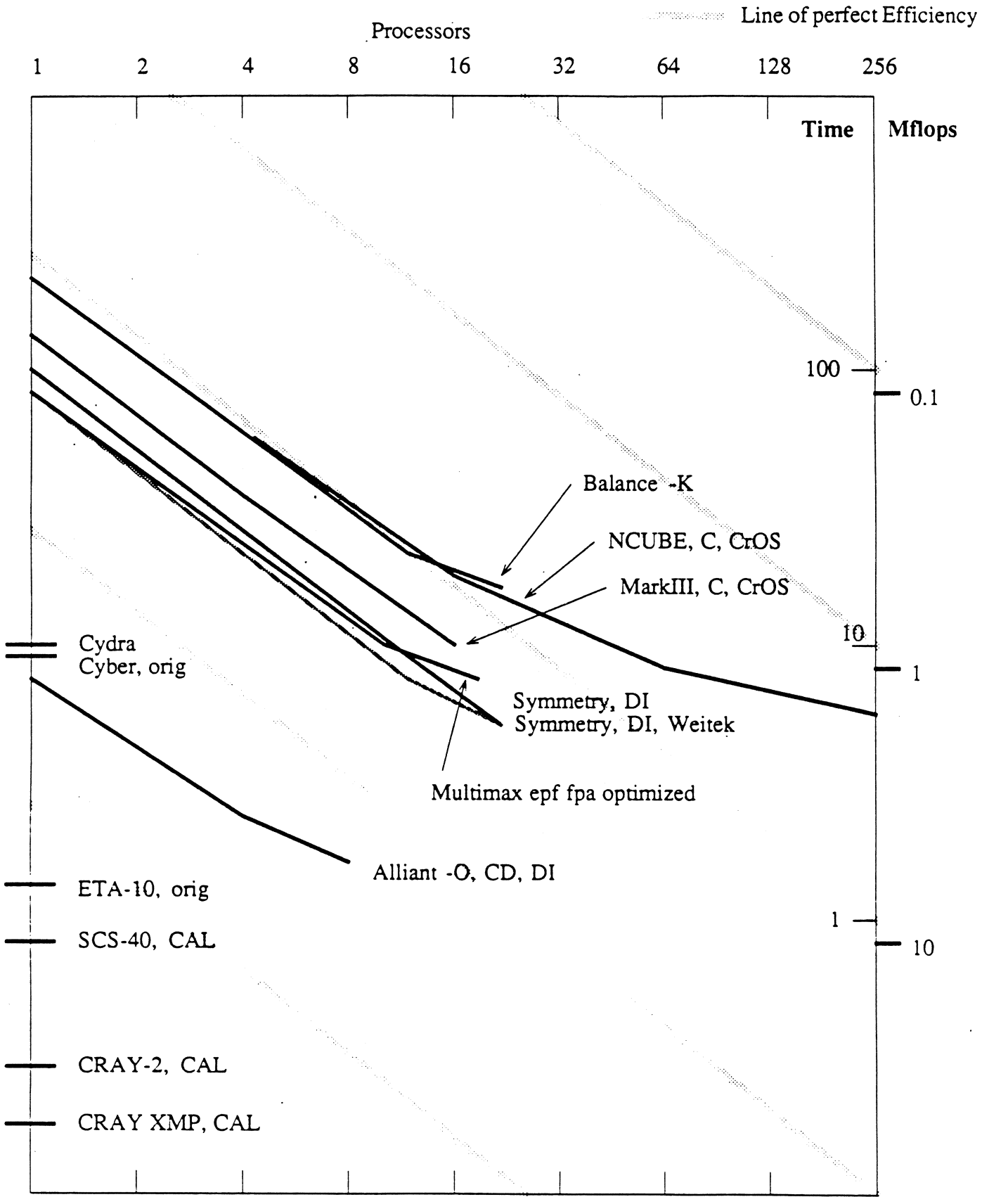


Figure 6

Banded LU: Size 1024, width 511

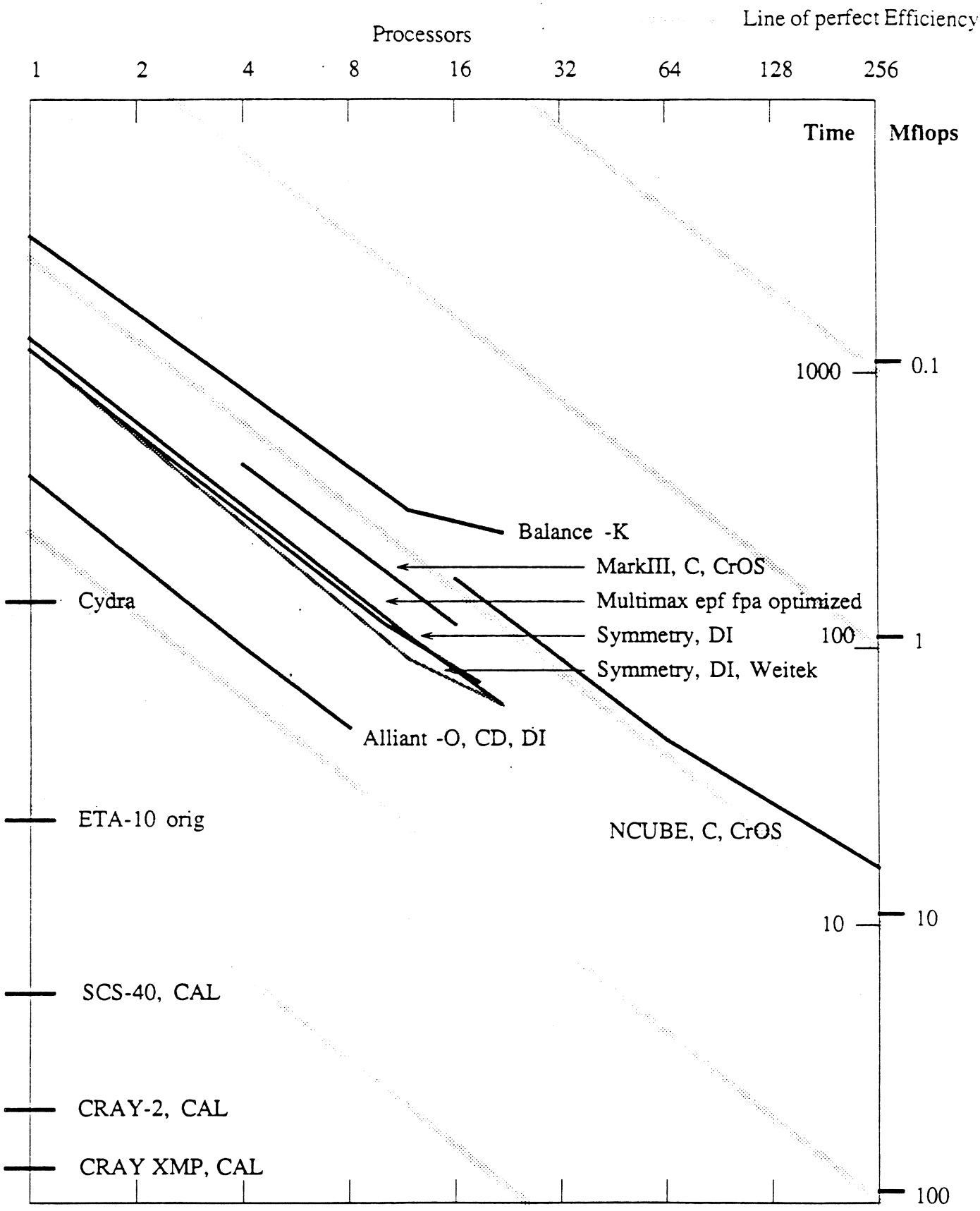


Figure 7

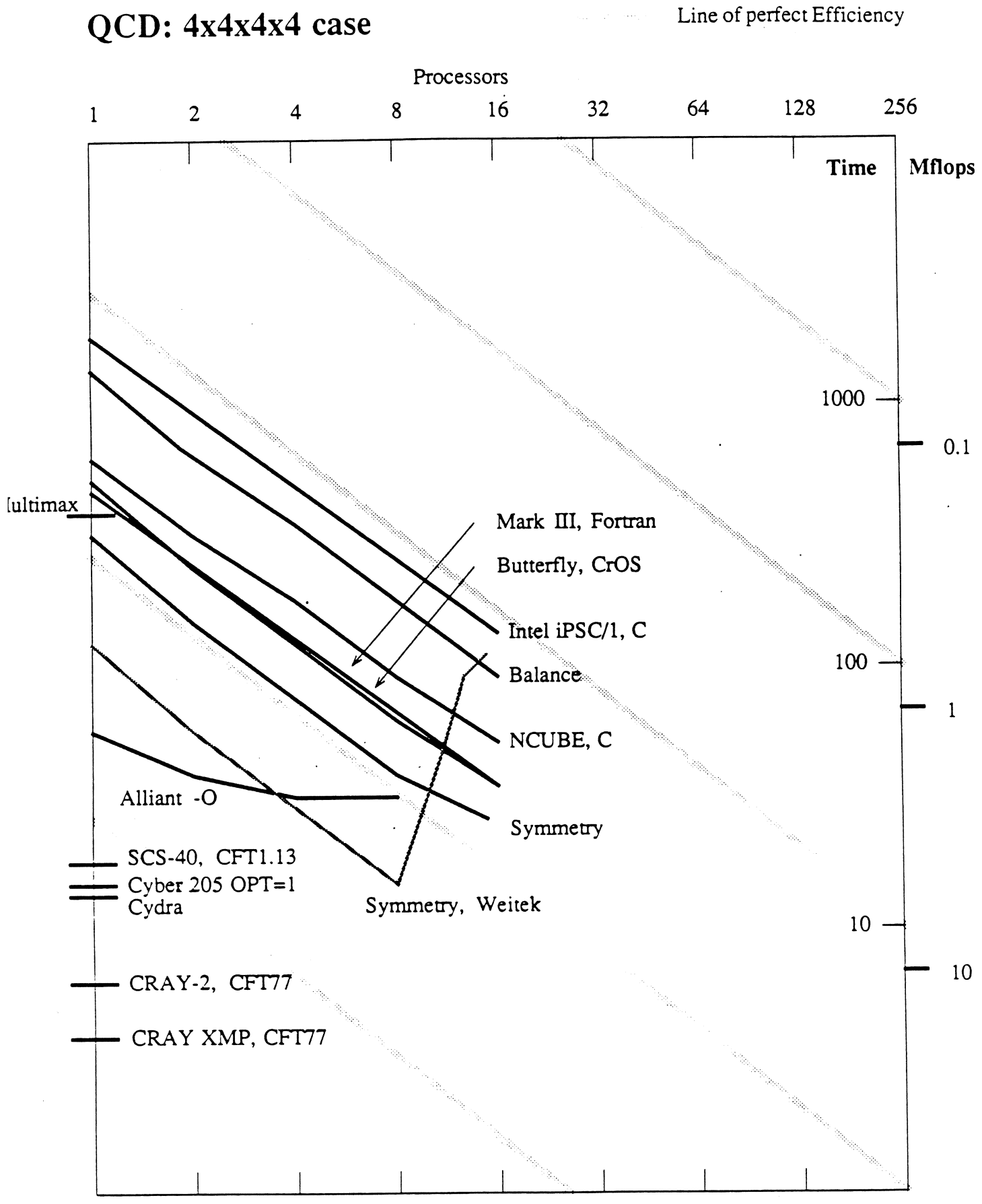

QCD: 4x4x4x4 case
Line of perfect Efficiency
Processors
1
2
4
8
16
32
64
128
256
Time
Mflops
1000
0.1
100
1
10
10
[ultimax
Mark III, Fortran
Butterfly, CrOS
Intel iPSC/1, C
Balance
NCUBE, C
Alliant -O
Symmetry
SCS-40, CFT1.13
Cyber 205 OPT=1
Cydra
Symmetry, Weitek
CRAY-2, CFT77
CRAY XMP, CFT77

Figure 8

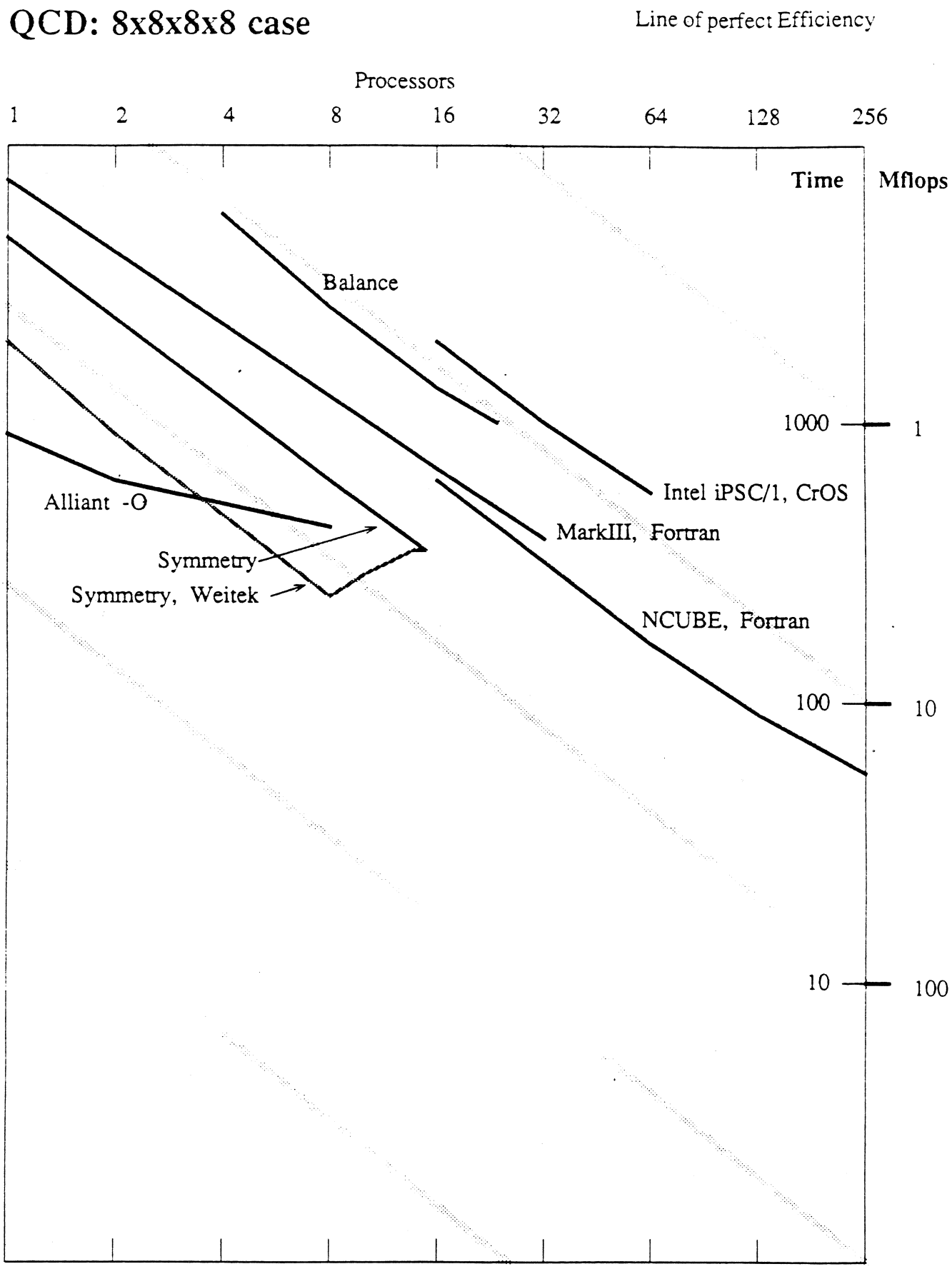

QCD: 8x8x8x8 case
Line of perfect Efficiency
Processors
1
2
4
8
16
32
64
128
256
Time
Mflops
Balance
Intel iPSC/1, CrOS
MarkIII, Fortran
NCUBE, Fortran
Alliant -O
Symmetry
Symmetry, Weitek
1000
1
100
10
10
100

Figure 9

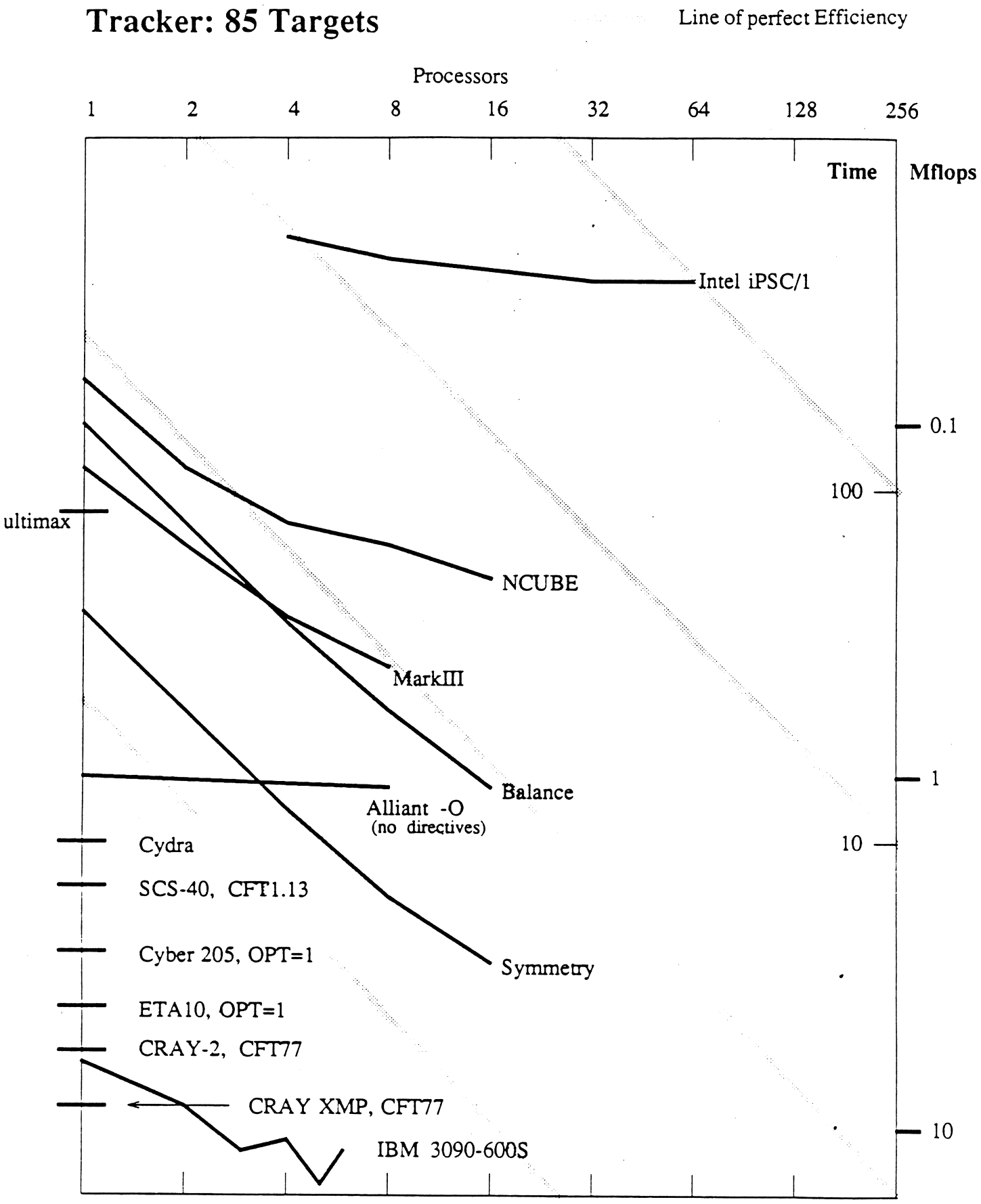

Tracker: 85 Targets
Line of perfect Efficiency
Processors
1
2
4
8
16
32
64
128
256
Time
Mflops
Intel iPSC/1
0.1
100
ultimax
NCUBE
MarkIII
Balance
1
Alliant -O
(no directives)
Cydra
10
SCS-40, CFT1.13
Cyber 205, OPT=1
Symmetry
ETA10, OPT=1
CRAY-2, CFT77
CRAY XMP, CFT77
10
IBM 3090-600S

Figure 10

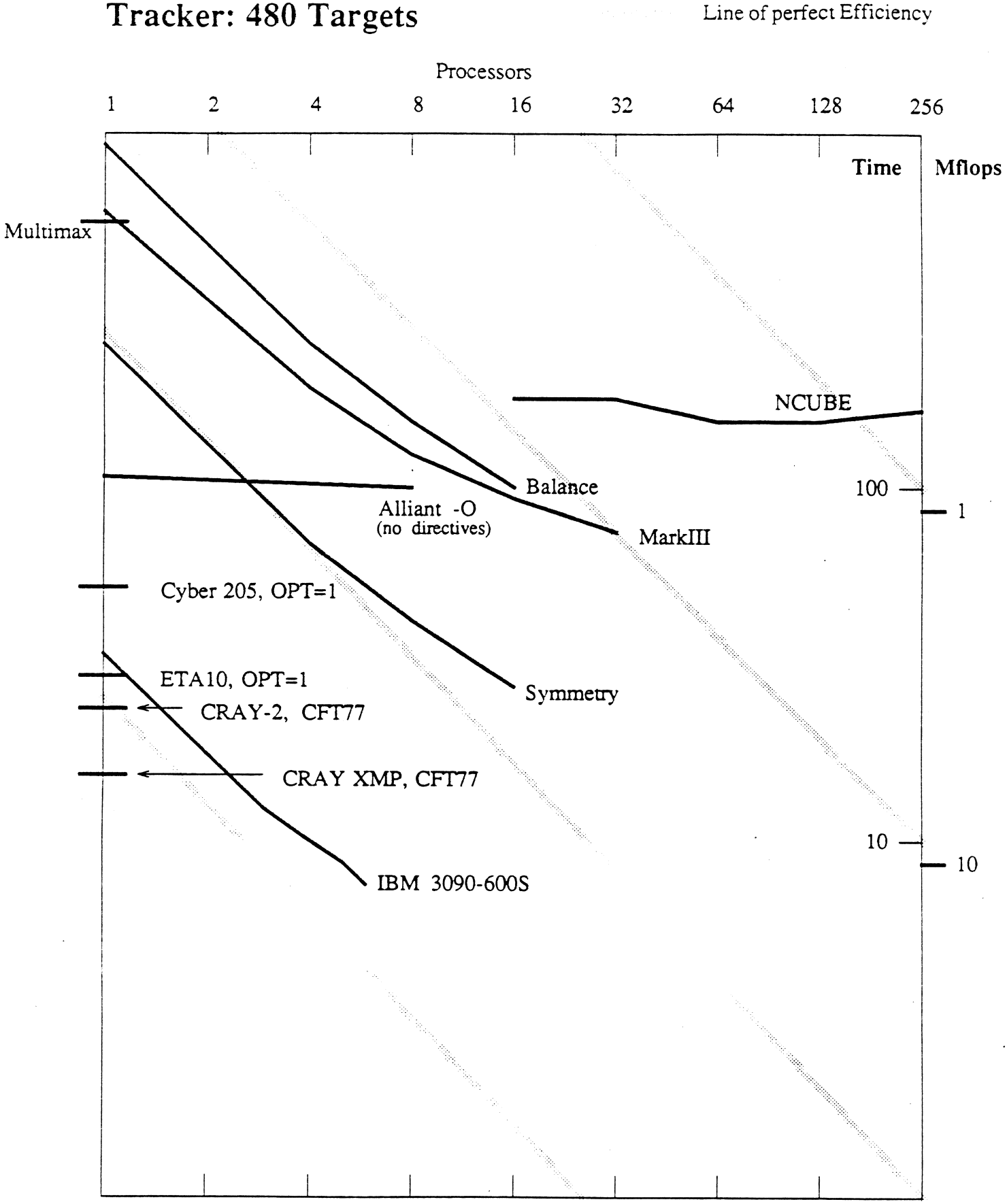

Tracker: 480 Targets
Line of perfect Efficiency
Processors
1
2
4
8
16
32
64
128
256
Time
Mflops
Multimax
NCUBE
Balance
Alliant -O
(no directives)
MarkIII
100
1
Cyber 205, OPT=1
ETA10, OPT=1
Symmetry
CRAY-2, CFT77
CRAY XMP, CFT77
10
10
IBM 3090-600S

Figure 11

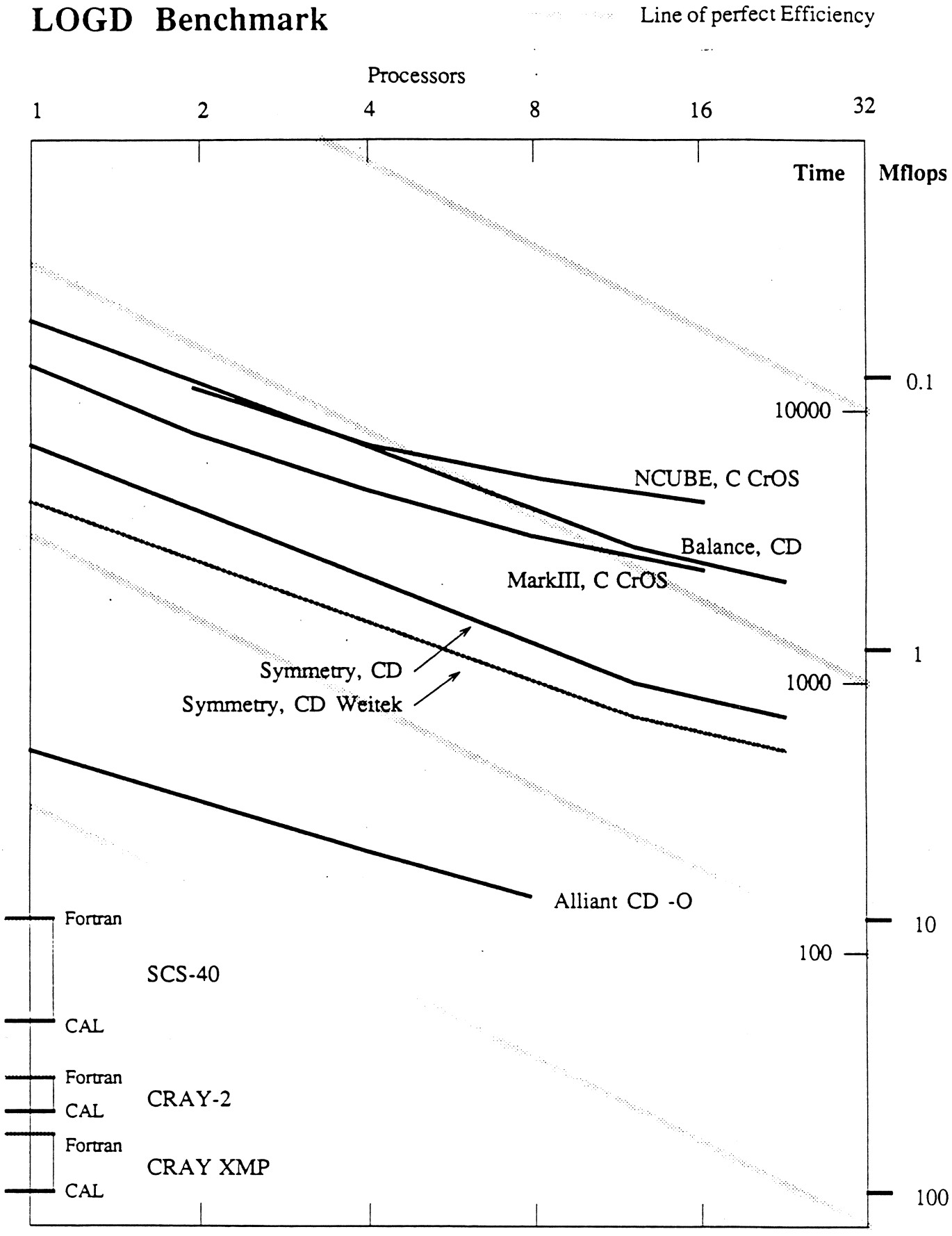

LOGD Benchmark
Line of perfect Efficiency
Processors
1
2
4
8
16
32
Time
Mflops
0.1
10000
NCUBE, C CrOS
Balance, CD
MarkIII, C CrOS
Symmetry, CD
Symmetry, CD Weitek
1
1000
Alliant CD -O
10
100
Fortran
SCS-40
CAL
Fortran
CRAY-2
CAL
Fortran
CRAY XMP
CAL
100

Figure 12

Vortex Benchmark

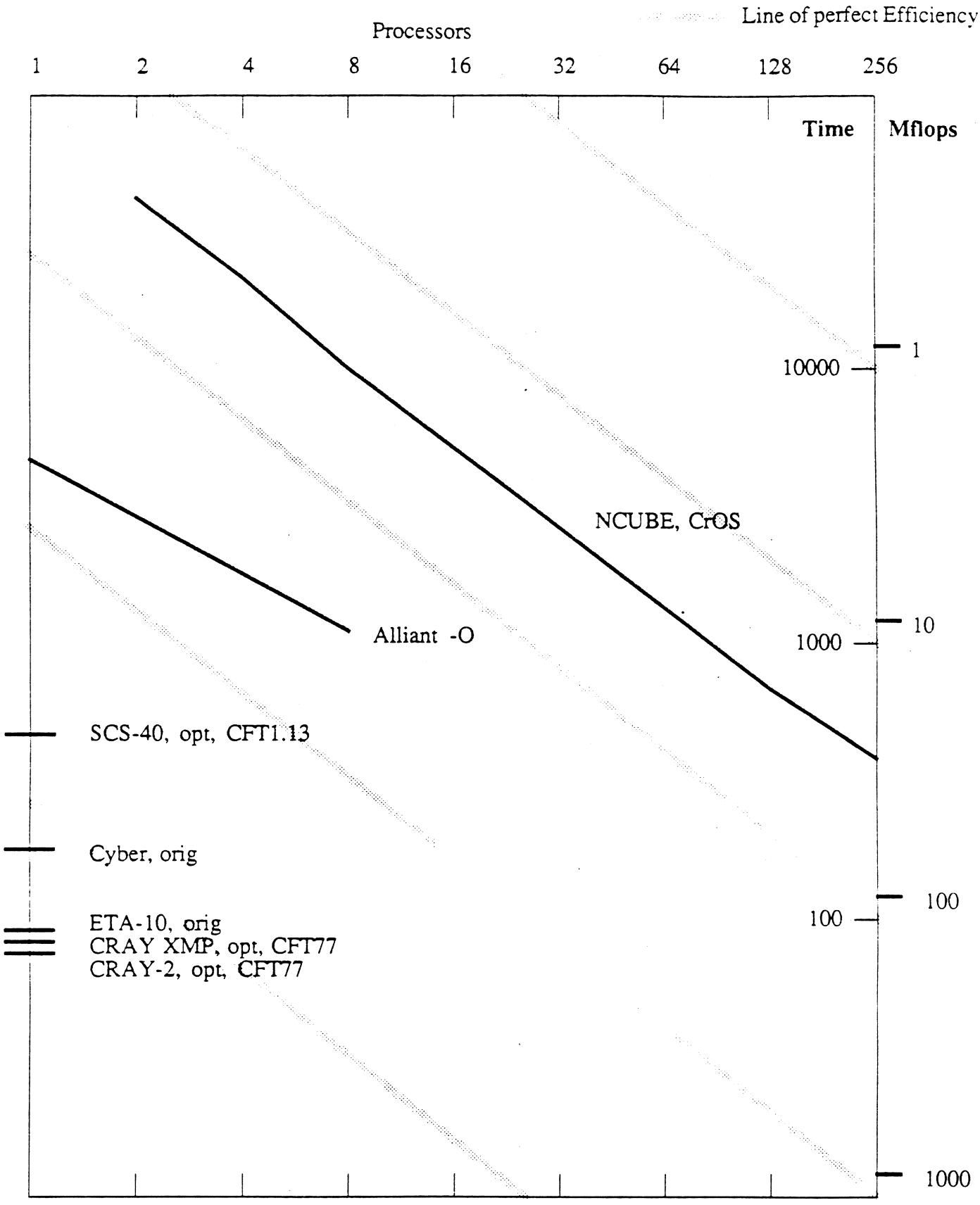


Figure 13

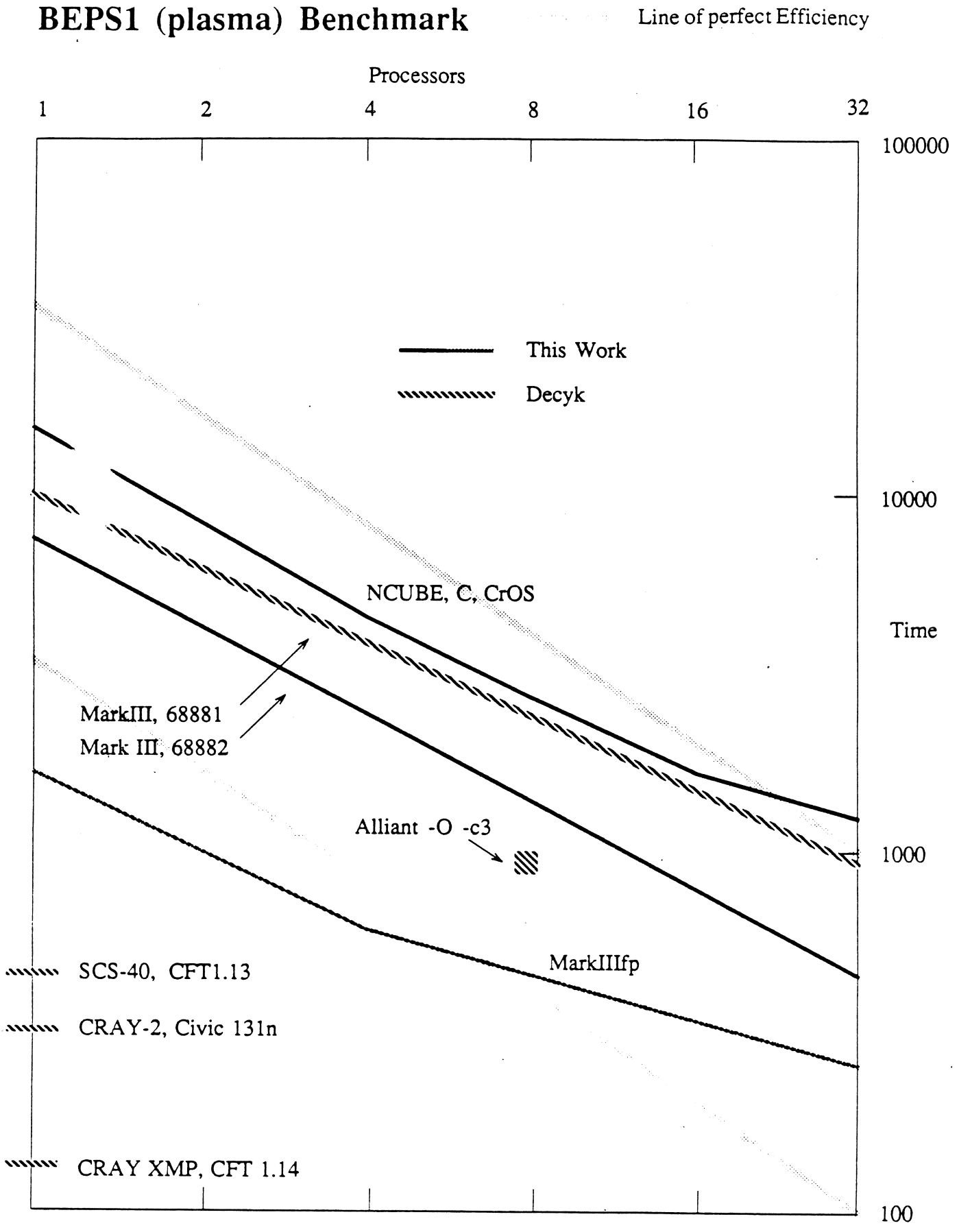

BEPS1 (plasma) Benchmark
Line of perfect Efficiency
Processors
1
2
4
8
16
32
100000
10000
1000
100
Time
This Work
Decyk
NCUBE, C, CrOS
MarkIII, 68881
Mark III, 68882
Alliant -O -c3
MarkIIIfp
SCS-40, CFT1.13
CRAY-2, Civic 131n
CRAY XMP, CFT 1.14

Figure 14

SPEC Frequently Asked Questions (FAQ) / SPEC Primer

Update Dec. 15, 1995: Major overhaul
Rearranged CPU benchmarks section:
SPEC95 benchmarks first
New section for SPEChpc96 suite

A SPEC Primer (as of December 15, 1995)
==========

Contents:

1. What is SPEC

SPEC, the Standard Performance Evaluation Corporation, is a non-profit corporation formed to "establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers" (from SPEC's bylaws). The founders of this organization believe that the user community will benefit greatly from an objective series of applications-oriented tests, which can serve as common reference points and be considered during the evaluation process. While no one benchmark can fully characterize overall system performance, the results of a variety of realistic benchmarks can give valuable insight into expected real performance.

Current members of SPEC include:

(1) Open Systems Group (OSG): Amdahl, AT&T, Auspex Systems, Bull, Compaq Computer, Cray Research, Dansk Data Elektronik, Data General, Digital Equipment, Electronic Data Systems, FirePower Systems, Fujitsu, HAL Computer Systems, Hewlett-Packard, Hitachi, IBM, Intel, Intergraph, International Computer (ICL), Locus Computing, Motorola, Network Appliance, Nikkei Datapro, Novell, Olivetti, Pyramid Technology, Ross Technology, Siemens Nixdorf Informationssysteme, Silicon Graphics, Sun Microsystems, Tricord Systems, Unisys, ZIFF Davis Publishing.

OSG Associates: Center for Scientific Computing (Finland), Defense Logistics Agency / Systems Design Center, Leibniz-Rechenzentrum (Germany),

NASA AMES Research Center, National Taiwan University (Taiwan), Oregon Graduate Institute, OSF Research Institute, Princeton University, Technische Universitaet Chemnitz-Zwickau (Germany), University of Aizu (Japan), University of California at Berkeley.

(2) High Performance Group (HPG): Convex Computers, Cray Research, Digital Equipment, Electronic Data Systems, Fujitsu America, Hewlett-Packard, International Supercomputing Technology Institute (ISTI, France), Kuck & Associates, NEC/HNSX Supercomputer, Silicon Graphics, Sun Microsystems.

HPG Associates: University of Illinois, University of Michigan, University of Minnesota.

Legally, SPEC is a non-profit corporation registered in California.

SPEC basically performs two functions::

- SPEC develops suites of benchmarks intended to measure computer performance. These suites are packaged with source code and tools and are extensively tested for portability before release. They are available to the public for a fee covering development and administrative costs. By license agreement, SPEC members and customers agree to run and report results as specified in each benchmark suite's documentation.

- SPEC publishes a quarterly report of SPEC news and benchmark results: The SPEC Newsletter. This provides a centralized source of information for SPEC benchmark results. Both SPEC members and non-SPEC members may publish in the SPEC Newsletter, though there is a fee for non-members. (Note that results may be published elsewhere as long as the format specified in the SPEC Run Rules and Reporting Rules is followed.)

As stated above, the organization SPEC really comprises two groups, each with their own benchmarks:
- Open Systems Group (OSG): Benchmarks in an UNIX / NT / VMS environment.
- High Performance Computing Group (HPCG): Benchmarking in a numeric computing environment, with emphasis on high-performance numeric computing.

Many people think about the OSG benchmarks only (often: the CPU benchmarks only) when they hear "SPEC". Since these benchmarks are indeed the best known ones of SPEC's benchmarks, most of this text deals with OSG's benchmarks. However, it also covers SPEC's other, possibly less known activities.

2. How to Contact SPEC

Effective December 7, 1995, SPEC's address has changed. It is now

SPEC (Standard Performance Evaluation Corporation)
10754 Ambassador Drive, Suite 201
Manassas, VA 22110, USA
Phone: (703) 331-0180
Fax: (703) 331-0181
E-Mail: spec-ncga@cup.portal.com

Dianne Rice is the Director of Operations for SPEC, her office is responsible for the adminstration of SPEC's products and for assisting customers with questions and orders. Technical questions regarding the SPEC benchmarks (e.g., problems with execution of the benchmarks), are usually referred to SPEC's technical support person..

3. SPEC Products and Services

==========

SPEC's main product is the benchmark suites. The code in the suites are developed by SPEC from code donated by various sources. SPEC works on portability and creates tools and meaningful workloads for the codes chosen as benchmarks. Therefore, the SPEC benchmarks are not the same as public domain programs that may exist under a similar name or the same name, and their execution times will, in general, be different.

The SPEC benchmark sources are generally available, but not free. SPEC is charging separately for its benchmark suites. The income from the benchmark source sales is intended to support the administrative costs of the corporation. Buyers of the benchmark tapes / CD-ROM's receive a shrink-wrapped site license with their first order of any of SPEC's products along with the manual explaining the rules for result publications. Currently, SPEC is in the transition from QIC 24 tapes (written in UNIX tar format) to CD-ROM as the distribution media for its benchmarks.

Current prices are*

SPEC95	$ 600.00	CPU intensive integer and floating-point benchmarks, new custumers
SPEC95	$ 300.00	..., current CINT92 / CFP92 licensees
CINT92	$ 425.00	CPU intensive integer benchmarks
CFP92	$ 575.00	CPU intensive floating point benchmarks
CINT92&CFP92	$ 900.00	
SDM	$ 1450.00	UNIX Software Development Workloads
SFS	$ 1200.00	System level file server (NFS) workload
SPEChpc96	$ 1200.00	High-performance computing benchmarks

* Accredited universities receive a 50 percent discount on SPEC benchmark products (SPEChpc96: 75 percent discount).

The SPEC Newsletter is published quarterly and contains result publications for a variety of machines, about 100-200 result pages per issue, together with articles dealing with SPEC and benchmarking.

SPEC Newsletter Prices (1 year subscription, 4 issues)

$550.00 for USA Orders $575 for International Orders

4. Current SPEC Benchmarks

4.1 "New" CPU Benchmarks, SPEC95

In August 1995, SPEC introduced the SPEC95 CPU benchmarks as a replacement for the older SPEC92 CPU benchmarks (see below, section 4.2). These benchmarks measure the performance of CPU, memory system, and compiler code generation. They normally use UNIX as the portability vehicle, but they have been ported to other operating systems as well. The percentage of time spent in operating system and I/O functions is generally negligible.

Although the SPEC95 benchmarks are sold in one package, they are internally composed of two collections:
- CINT95, integer programs, representing the CPU-intensive part of system or commercial application programs;
- CFP95, floating-point programs, representing the CPU-intensive part of numeric-scientific application programs.

Results are reported for CINT95 and for CFP95 on individual report pages, results can be reported for either one or for both.

Integer benchmarks: CINT95

This suite contains eight benchmarks performing integer computations,
all of them written in C. The individual programs are:

Number and name	Application
099.go	Artificial intelligence; plays the game of "Go"
124.m88ksim	Moto 88K chip simulator; runs test program
126.gcc	New version of GCC; builds SPARC code
129.compress	Compresses and decompresses file in memory
130.li	LISP interpreter
132.ijpeg	Graphic compression and decompression
134.perl	Manipulates strings (anagrams) and prime numbers in Perl
147.vortex	A database program

Floating-point benchmarks: CFP92

This suite contains 10 benchmarks performing floating-point computations.
all of them written in Fortran77. The individual programs are:

Number and name	Application
101.tomcatv	A mesh-generation program
102.swim	Shallow water model with 1024 x 1024 grid
103.su2cor	Quantum physics; Monte Carlo simulation
104.hydro2d	Astrophysics; Hydrodynamical Navier Stokes equations
107.mgrid	Multi-grid solver in 3D potential field
110.applu	Parabolic/elliptic partial differential equations
125.turb3d	Simulates isotropic,homogeneous turbulence in a cube
141.apsi	Solves problems regarding temperature, wind, velocity and distribution of pollutants
145.fpppp	Quantum chemistry
146.wave5	Plasma physics; electromagnetic particle simulation

More information about the individual benchmarks is contained in
description files in each benchmark's subdirectory.

The CPU benchmarks can be used for measurement in two ways:
- Speed measurement
- Throughput measurement

Speed Measurement

The results ("SPEC Ratio" for each individual benchmark) are
expressed as the ratio of the wall clock time to execute one single
copy of the benchmark, compared to a fixed "SPEC reference time".
For the SPEC95 benchmarks, a Sun SPARCstation 10/40 was chosen as
the reference machine.

As is apparent from results publications, the different SPEC ratios
for a given machine can vary widely. SPEC encourages the public to
look at the individual results for each benchmarks. Users should
compare the characteristics of their workload with that of the
individual SPEC benchmarks and consider those benchmarks that
best approximate their jobs. However, SPEC also recognizes the
demand for aggregate result numbers and has defined the integer
and floating-point averages.

The following averages have been defined for speed measurements with
the SPEC95 benchmarks:

SPECint_base95 = geometric mean of the 8 SPEC ratios from CINT95 when

compiled with conservative optimization for each benchmark;

SPECfp_base95 = geometric mean of the 10 SPEC ratios from CFP95 when compiled with conservative optimization for each benchmark;

SPECint95 = geometric mean of the 8 SPEC ratios from CINT95 when compiled with aggressive optimization for each benchmark;

SPECfp95 = geometric mean of the 10 SPEC ratios from CFP92 when compiled with aggressive optimization for each benchmark.

For a more detailed explanation about the difference between conservative ("baseline") and aggressive ("peak") optimizations, see below.

Throughput (Rate) Measurement

With this measurement method, called the "homogenuous capacity method", several copies of a given benchmark are executed. This method is particularly suitable for multiprocessor systems. The results, called SPEC rate, express how many jobs of a particular type (characterized by the individual benchmark) can be executed in a given time. (The SPEC reference time happens to be a week, the execution times are normalized with respect to SPEC reference machine). The SPEC rates therefore characterize the capacity of a system for compute-intensive jobs of similar characteristics.

Similar as with the speed metric, SPEC has defined averages:

SPECint_rate_base95 = geometric mean of the 8 SPEC rates from CINT95 when compiled with conservative optimization for each benchmark;

SPECfp_rate_base95 = geometric mean of the 10 SPEC rates from CFP95 when compiled with conservative optimization for each benchmark;

SPECint_rate95 = geometric mean of the 8 SPEC rates from CINT95 when compiled with agressive optimization for each benchmark;

SPECfp_rate95 = geometric mean of the 8 SPEC rates from CINT95 when compiled with agressive optimization for each benchmark.

Because of the different units, the values SPECint95/SPECfp95 and SPECrate_int95/SPECrate_fp95 cannot be compared directly.

Baseline vs. peak measurements

Historically, the non-baseline (also called "peak") metrics have been the first ones introduced by SPEC; this is also the reason why they have the shorter names. However, in 1994, the SPEC Open Systems Steering Committee decided to introduce "baseline results". The results (for both speed and throughput measurements) have to be measured with more restrictive run rules, regulating the use of compiler/linker optimization options ("flags"). The most important baseline rules are:

- Only one set of flags for all benchmarks per suite, per language. For the SPEC95 benchmarks, their number is limited to 4, not counting flags that are necessary for portability.
- No "assertion flags" (flags asserting a certain property of the code not apparent in the source code, e.g. a "non-aliasing" property).
- "Ease-of-use" requirements for feedback-directed optimizations (SPEC95), or no feedback-directed optimization allowed (SPEC92).

As a general guideline, a system vendor is expected to endorse the general use of the baseline options by all customers who seek to achieve good application performance.

The detailed baseline rules, together with an explanation, are listed in an article "Reviewing the New Baseline Rules" (R. Weicker, J. Reilly) in SPEC Newsletter Vol. 6 No. 2 (June 1994). This article covers the SPEC92 baseline rules but many of the explanations carry over to the new suite as well. For the exact SPEC95 baseline rules, see the detailed benchmark documentation.

The intention is that baseline results represent the performance a not-so-sophisticated user would achieve, whereas the traditional "peak" rules allow a selection of optimization flags that is more typical for sophisticated users.

Effective June 1994, when SPEC's CPU benchmark results are reported, the reports must include baseline results. Baseline-only reporting is allowed. A test sponsor is free to mention only peak results in marketing literature, but baseline results must be available and provided upon request.

4.2 "Old" CPU Benchmarks, SPEC92

The previous generation of SPEC CPU benchmarks was introduced in 1992. Like the 1995 benchmarks, they are internally composed of two collections, CINT92 and CFP92; results are reported separately.

CINT92, current release: Rel. 1.1

This suite contains six benchmarks performing integer computations, all of them written in C. The individual programs are:

Number and name	Application	Approx. size gross	net
008.espresso	Logic Design	14800	11000
022.li	Interpreter	7700	5000
023.eqntott	Logic Design	3600	2600
026.compress	Data Compression	1500	1000
072.sc	Spreadsheet	8500	7100
085.gcc	Compiler	87800	58800
		123900	85500

The approximate static size is given in numbers of source code lines, including declarations (header files). "Gross" numbers include comments and blank lines, "net" numbers exclude them.

A somewhat more detailed, though still short description of the CINT92 benchmarks (from an article by Jeff Reilly, in SPEC Newsletter Vol. 4, No. 4):

008.espresso	Generates and optimizes Programmable Logic Arrays.
022.li	Uses a LISP interpreter to solve the nine queens problem, using a recursive backtracking algorithm.
023.eqntott	Translates a logical representation of a Boolean equation to a truth table.
026.compress	Reduces the size of input files by using Lempel-Ziv coding.
072.sc	Calculates budgets, SPEC metrics and amortization schedules in a spreadsheet based on the UNIX cursor-controlled package "curses".
085.gcc	Translates preprocessed C source files into optimized Sun-3 assembly language output.

CFP92, current release: Rel. 1.1

This suite contains 14 benchmarks performing floating-point computations. 12 of them are written in Fortran, 2 in C. The individual programs are:

Number and name	Application	Lang.	Approx. size gross	net
013.spice2g6	Circuit Design	F	18900	15000
015.doduc	Simulation	F	5300	5300
034.mdljdp2	Quantum Chemistry	F	4500	3600
039.wave5	Electromagnetism	F	7600	6400
047.tomcatv	Geometric Translation	F	200	100
048 ora	Optics	F	500	300
052.alvinn	Robotics	C	300	200
056.ear	Medical Simulation	C	5200	3300
077.mdljsp2	Quantum Chemistry	F	3900	3100
078.swm256	Simulation	F	500	300
089.su2cor	Quantum Physics	F	2500	1700
090.hydro2d	Astrophysics	F	4500	1700
093.nasa7	NASA Kernels	F	1300	800
094.fpppp	Quantum Chemistry	F	2700	2100
			57900	43900

Short description of the benchmarks:

013.spice2g6	Simulates analog circuits (double precision).
015.doduc	Performs Monte-Carlo simulation of the time evolution of a thermo-hydraulic model for a nuclear reactor's component (double precision).
034.mdljdp2	Solves motion equations for a model of 500 atoms interacting through the idealized Lennard-Jones potential (double precision).
039.wave5	Solves particle and Maxwell's equations on a Cartesian mesh (single precision).
047.tomcatv	Generates two-dimensional, boundary-fitted coordinate systems around general geometric domains (vectorizable, double precision).
048 ora	Traces rays through an optical surface containing spherical and planar surfaces (double precision).
052.alvinn	Trains a neural network using back propagation (single precision).
056.ear	Simulates the human ear by converting a sound file to a cochleogram using Fast Fourier Transforms and other math library functions (single precision).
077.mdljsp2	Similar to 034.mdljdp2, solves motion equations for a model of 500 atoms (single precision).
078.swm256	Solves the system of shallow water equations using finite difference approximations (single precision).
089.su2cor	Calculates masses of elementary particles in the framework of the Quark Gluon theory (vectorizable, double precision).
090.hydro2d	Uses hydrodynamical Navier Stokes equations to calculate galactical jets (vectorizable, double precision).
093.nasa7	Executes seven program kernels of operations used frequently in NASA applications, such as Fourier transforms and matrix manipulations (double precision).
094.fpppp	Calculates multi-electron integral derivatives (double precision).

Similar as the SPEC95 benchmarks, the SPEC92 CPU benchmarks can be used for measurement in two ways:
- Speed measurement
- Throughput measurement

The metrics are defined similarly as for SPEC95 (geometric mean), except that for historical reasons, the order of the metric name components is different:

```
SPECbase_int92 =      geometric mean, integer, baseline, speed;
SPECbase_fp92 =       geometric mean, FP, baseline, speed;
SPECint92 =           geometric mean, integer, peak, speed;
SPECfp92  =           geometric mean, FP, peak, speed;
SPECrate_base_int92 = geometric mean, integer, baseline, throughput;
SPECrate_base_fp92 =  geometric mean, FP, baseline, throughput;
SPECrate_int92 =      geometric mean, integer, peak, throughput;
SPECrate_fp92  =      geometric mean, FP, peak, throughput.
```

Why customers should now use SPEC95

The SPEC92 CPU benchmarks have gained high popularity in the computer community, and SPEC is aware of the fact that SPEC92 results will probably still be quoted for some time. However, SPEC feels that the SPEC95 benchmarks represent a significant step towards better CPU benchmarking:

- The SPEC95 benchmarks, in particular the integer benchmarks, are drawn from a wider variety of application ares;
- The SPEC95 benchmarks are more realistic in their memory system use (less locality, more cache misses);
- Benchmarks that have tended to be susceptible to special-case compiler optimizations have been dropped;
- All benchmarks, including those that have been carried over from SPEC92 to SPEC95, have been given new input data sets for longer execution times (hence, also new distinguishing numbers), resulting in better measurement accuracy.

SPEC therefore encourages the computer community to move over to the new CPU benchmarks as fast as possible. SPEC will cease publication of SPEC95 results in the SPEC Newsletter after June 1996.

Since the machines used by SPEC to generate the reference times are different (Sun SPARCstation 10/40 for SPEC95, VAX 11/780 for SPEC92), the metric values are in different numerical ranges. The SPEC92 benchmark averages measured on the SPEC95 reference machine, with baseline compilation options, are

SPECbase_int95 = 41.26 SPECbase_fp92 = 34.35

However, SPEC warns that these numbers should not be used to derive a general conversion formula from SPEC92 numbers to SPEC95 numbers. There are different benchmarks in the two suites, with different characteristics. Hardware and compilers will, in general, react differently to these different characteristics. While there will be a certain correlation, there is no "magic formula" to convert a SPEC92 metric to the corresponding SPEC95 metric.

4.3 SDM Benchmark Suite

SDM stands for "System Development Multi-tasking"; the benchmarks in this suite (current release: 1.1) characterize the capacity of a system in a multi-user UNIX environment. Contrary to the CPU benchmarks, the SDM benchmarks contain UNIX shell scripts (consisting of commands like "cd", "mkdir", "find", "cc", "nroff", etc.) that

exercise the operating system as well as the CPU and I/O components of the system. The suite contains two benchmarks:

057.sdet Represents a large commercial UNIX/C based software development environment. This characterization is based on AT&T analysis and models developed by Steve Gaede, formerly with AT&T Bell Laboratories.

061.kenbus1 Represents UNIX/C usage in a Research and Development environment. This characterization is based on data collection and analysis at Monash University by Ken McDonell.

For each benchmark, throughput numbers (scripts, i.e. simulated user loads per hour) are given for several values of concurrent workloads. The reader can determine the peak throughput as well as the ability of a system to sustain throughput over a range of concurrent workloads. Since the workloads for the two benchmarks are different, their throughput values are also different and cannot be compared directly.

4.4 SFS Benchmark Suite

SFS stands for "system-level file server"; SFS Release 1 is designed to provide a fair, consistent and complete method for measuring and reporting NFS file server performance.

SFS Release 1.1 contains one benchmark, 097.LADDIS. This benchmark measures NFS file server performance in terms of NFS response time and throughput. It does this by generating a synthetic NFS workload based on a workload abstraction of an NFS operation mix and an NFS operation request rate.

Running 097.LADDIS requires a file server (the entity being measured) and two or more "load generators" connected to the file server via a network medium. The load generators are each loaded with 097.LADDIS and perform the 097.LADDIS workload on file systems exported by the file server.

SFS Release 1.1 results include full server configuration information (hardware and software) and a graph of server response time versus NFS load for the 097.LADDIS operation mix. Compared with the predecessor version SFS 1.0, Release 1.1 contains bug fixes and improved documentation. The workload did not change, so results are comparable.

In April 1995, the Open Systems Steering Committee established a separate steering sub-committee for the SFS activities, the System File Server Steering Committee (SFSSC). However, there is still only one membership in the Open Systems Group; OSG members have access to both the SFS and the other OSG benchmarks (CPU, SDM).

4.5 SPEChpc96 Benchmark Suite

At the Supercomputing'95 conference (Dec. 1995), the SPEC High Performance Computing Group (HPCG) announced the SPEChpc96 benchmarks suite, consisting of two benchmarks:

- SPECseis96: An industrial application based on modern seismic processing programs used in the search for oil and gas.
- SPECchem96: An improved version of programs called GAMESS (General Atomic and Molecular Electronic Structure Systems) that came from the US Department of Energy's National Resource for Computations in Chemistry.

There are four metrics defined for both benchmarks, depanding on the data size (small, medium, large, extra large):

SPEC96seis_SM	SPECchem96_SM
SPEC96seis_MD	SPECchem96_MD
SPEC96seis_LG	SPECchem96_LG
SPEC96seis_XL	SPECchem96_XL

All benchmarks are written in Fortran77.

Initial results have been presented at the Supercomputing'95 conference; more results will be reported in the March 1996 SPEC Newsletter.

4.6 Comprehensive result lists

Comprehensive lists of all current SPEC result (average or "single figure of merit" only) published during the year have been printed in the year-end issues of the SPEC Newsletter in December 1992 through 1995. They contain the system name, the respective value, and a pointer to the newsletter issue where the full result report can be found. Readers are warned that "SPEC does not recommend that readers use any one value for making comparisons. There is a wealth of information in the Reporting Pages which cannot be easily reduced to summary values".

All previous SPEC results published in the SPEC Newsletter are accessible online via
  hppt://netlib2.cs.utk.edu/performance/html/PDSsearch.html

5. Outdated SPEC Benchmarks

SPEC has published the first CPU benchmark suite in 1989, the last release was 1.2b. It contained 10 compute-intensive programs, four integer (written in C) and six floating-point (written in Fortran). The following average values had been defined:

```
SPECint89  = geometric mean of the SPEC ratios of the four integer
             programs in rel. 1.2b (CPU-Suite of 1989).
SPECfp89   = geometric mean of the SPEC ratios of the six
             floating-point programs in rel. 1.2b.
SPECmark89 = geometric mean of all 10 SPEC ratios of the
             programs in rel. 1.2b.
```

In addition, there was the possibility of throughput measurements, with two copies of a benchmark running per CPU, called "Thruput Method A" (There was never a "Method B"). The following average values had been defined:

```
SPECintThruput89 = geometric mean of the Thruput Ratios of the
                   four integer programs.
SPECfpThruput89 = geometric mean of the Thruput Ratios of the
                  six floating-point programs.
SPECThruput89 ("aggregate thruput") = geometric mean of the
                  Thruput Ratios of all 10 programs.
```

SPEC discontinued selling SPEC Release 1.2b benchmarks after December 1992; and discontinued result publications for this suite in June 1993.

6. Forthcoming SPEC Benchmarks

6.1 Benchmark areas considered

A number of areas have been considered or are being considered
by SPEC (Open System group) for future benchmark efforts:
- New CPU benchmarks
- Client/Server benchmarks
- Commercial computing benchmarks
- RTE-based benchmarks
- I/O benchmarks

The benchmark suites that OSG is currently working on are:

6.2 Web Server Benchmark

This benchmark will measure the performance of computers (both hardware and software) that are used as WWW servers. Similar to the SFS benchmark, the benchmark code runs on one or more load generators (clients) that generate HTTP request over a network. The size of the files on the server and the distribution of accesses will follow patterns observed on some real-world WWW servers.

Similarly to the SFS benchmark, the Web server benchmark will try to measure only the server's performance, independent from the particular client(s) used. Similarly, even though many WWW servers are accessed over a Wide Area Network, the benchmark will use a dedicated Local Area Network for uniform measurement conditions; it will not measure WAN overhead.

6.3 SFS Benchmark Version 1.2

The SFS Steering Committee is working on a new version of the SFS benchmark suite; its main features will be:
- Support of the NFS protocol version 3
- a modified workload ("LADDIS light").

Both the WWW benchmark and version 2 of the SFS benchmark are expected to be released in the first half of 1996.

6.4 New Benchmarks in General

SPEC is always open to suggestions from the computing community for future benchmarking directions. Of course, SPEC is even more receptive to proposals for actual programs that can be used as benchmarks. In particular, the SPEC Open Systems Group is currently looking for programs that have similar characteristics as system code (integer code, large, low code locality, structured according to somewhat modern software engineering principles). Also, benchmark candidate programs in languages other than C and Fortran are welcome (C++, Ada, ...). The High Performance Group is looking for large application programs that stress modern high-performance computers to their limits.

7. Membership in SPEC

The costs for SPEC membership are:

Annual Dues	$ 5000.00	Initiation Fee	$ 1000.00

The dues for membership in OSG and HPG are separate.

There is also the category of a "SPEC Associate", intended for

accredited educational institutions or non-profit organizations:

Annual Dues $ 1000.00 Initiation Fee $ 500.00

Among the membership benefits are Newsletters and benchmark suites as they are available, with company-wide licenses within their countries. Most important is early access to benchmarks that are being developed, and the possibility to participate in technical work on the benchmarks.

The benefits for associates are the same as for members except that OSG associates have no voint rights. The intention for associates is that they can act in an advisory capacity to SPEC, getting first-hand experience in an area that is widely neglected in academia but nonetheless very important in the "real world", and providing technical input to SPEC's task.

SPEC meetings (Open Systems Group and SFS subgroup) are held about every two months, for technical work and decisions about the benchmarks. The SPEC High Performance Computing group meets four times in the year.

Every member or associate can participate in these meetings and make proposals. In the Open Systems Group, decisions are made by the Open Systems Steering Committee (nine members) elected by the general membership at the Annual Meeting. In the High Performance Computing Group, the Steering Committee consists of all members present at a meeting. All SPEC members of OSG or HPG vote before a benchmark is finally accepted.

8. SPEC Order Form

==========
SPEC Product Order Form
==========

Name: Dr. N. P. TOPHAM
Company: DEPT. COMPUTER SCIENCE, UNIVERSITY of EDINBURGH
Address: JCMB, KINGS BUILDINGS
Line 2: MAYFIELD ROAD
Line 3:
City: EDINBURGH
State/ZIP: EH9 3JZ
Country/Postal Code: UK
Phone: +44 131 650 5171
Fax: +44 131 667 7209

==========

1. Shipping and Handling Charges
(Orders shipped within the U.S are sent UPS 2nd Day Service)
A. International orders, please add an additional $35
B. Canada & Caribbean Islands, please add an additional $20
C. Three or more tapes $70

2. Expedited shipments, please provide your overnight service account number or we can also bill directly to a credit card.

#_____ Federal Express
#_____ United Parcel Service
#_____ DHL
#_____ Airborne

3. Multiple Tape/CD-ROM Copies Discount (Per suite)
[ ] 2-4 = 10%
[ ] 5-9 = 20%
[ ] 10+ = 40%

4. Sale Tax of 4.5% applicable to all orders
   within the state of Virginia

Item No.	Products	Unit Cost	Educational** Non-Profit Cost
	SPEC Newsletter (To order back issues, contact the SPEC office)		
N1 [ ]	1-year subscription	$ 550	550
N2 [ ]	1-year subscription (International)	$ 575	575
	SPEC Benchmark Suites		
T3 [ ]	SPEC SFS Release 1.1	$1200	600
T4 [ ]	SPEC CINT92 Release 1.1*	$ 425	218
T5 [ ]	SPEC CFP92 Release 1.1*	$ 575	288
T7 [ ]	SPEC SDM Release 1.1	$1450	725
	Subtotal	_____	_____
1. Shipping/Handling Charges (see above)		_____	_____
2. Multi-orders discount (see above)		_____	_____
3. Expedited Orders (see above)			
4. Sales Tax - VIRGINIA ONLY (see above)		_____	_____
	TOTAL	_____	_____

Method of Payment

[ ] Check made payable to SPEC in U.S. Dollars
[ ] Wire Transfer (contact SPEC office for bank information)

Credit Card
[ ] Visa [ ] Mastercard [ ] American Express

Name on credit card (cardholder) _____

Credit Card # _____

Expiration Date _____

Signature _____

Date _____

Please Mail or Fax your order to:

SPEC (Standard Performance Evaluation Corporation)
10754 Ambassador Drive, Suite 201
Manassas, VA 22110, USA
Phone: (703) 331-0180
Fax: (703) 331-0181
E-Mail: spec-ncga@cup.portal.com

SPEC Product Order Form

9. Acknowledgments

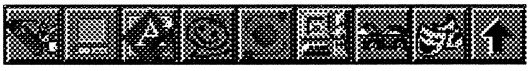
This summary of SPEC's activities has been written by Reinhold Weicker (Siemens Nixdorf, Paderborn/Germany), originally for the

"Usenet" bulletin board under the title "Answers to Frequently Asked Questions about SPEC". Over the months, updates have been made by Jeff Reilly (Intel) and Reinhold Weicker.

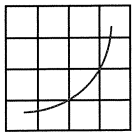
Managerial and technical inquiries about SPEC should be directed to SPEC at spec-ncga@cup.portal.com.

E-Mail questions that do not interfere too much with our real work :-) can also be mailed to:

From North America	jwreilly@mipos2.intel.com
From Europe or elsewhere	weicker.pad@sni.de



Ion Cionca



spec

STANDARD PERFORMANCE EVALUATION CORPORATION
10754 Ambassador Drive, Suite 201 • Manassas, VA 20109
Phone: (703) 331-0180 • FAX: (703) 331-0181 • E-mail: info@specbench.org

License No. 1141

SITE LICENSE AGREEMENT FOR SPEC BENCHMARK PRODUCTS

READ THE TERMS AND CONDITIONS OF THIS AGREEMENT CAREFULLY BEFORE OPENING THIS PACKAGE CONTAINING THE BENCHMARK PROGRAM AND THE ACCOMPANYING DOCUMENTATION (THE "MATERIALS"). THIS AGREEMENT REPRESENTS THE ENTIRE AGREEMENT CONCERNING THE MATERIALS BETWEEN YOU AND SPEC, SUPERSEDES ANY PRIOR PROPOSAL, REPRESENTATION OR UNDERSTANDING BETWEEN THE PARTIES. THE RESTRICTIONS AND LIMITATIONS OF THIS AGREEMENT DO NOT APPLY TO THE DIRECTORY './BENCHSPEC/085.GCC' WHICH MAY BE FREELY COPIED AND DISTRIBUTED SUBJECT TO THE TERMS OF THE FREE SOFTWARE FOUNDATION'S LICENSE FOR THE GNU C COMPLIER. BY OPENING THIS PACKAGE CONTAINING THE MATERIALS, YOU ARE ACCEPTING AND AGREEING TO THE TERMS OF THIS AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY THESE TERMS, YOU SHOULD PROMPTLY RETURN THE MATERIALS IN UNOPENED FORM, AND YOU WILL RECEIVE A REFUND OF YOUR MONEY.

Company: **University of Edinburgh**
Site: **Edinburgh, UK**

1. **Grant.** **SPEC** agrees to grant and **USER** agrees to accept a non-transferable and non-exclusive license to use the materials, subject to the restrictions and conditions set forth below.

2. **Copies and Location.** **USER** may make only exact and complete copies of the original of the materials, without modification or merger with other software. The original and copies thereof may be used only within a radius of 10 miles centered at the above-identified Site.
USER will not permit any third party to use or copy the original or any copy thereof. In the event of unauthorized transfer or copying, **USER** will pay **SPEC** a penalty in the amount of one additional fee for each such transfer or copy.

3. **Scope of Use.** The materials may be used only for measuring the performance of computers. For example, the materials may not be used for general spreadsheet purposes.

4. **Fee.** The fee paid by **USER** to **SPEC** for the materials is in consideration of **SPEC**'s cost in connection with the distribution of the materials, and is not a royalty.

5. **Reports.** In order to preserve the integrity of **SPEC**'s performance standards, **USER** agrees to run the materials and report measured results strictly in accordance with the rules published by **SPEC** for the materials.

6. **Term.** This Agreement is effective upon **USER** opening the package containing the materials and shall continue until terminated. **USER** may terminate the agreement at any time by returning the materials and all copies thereof to **SPEC**.

The license granted as to the current release of the materials shall terminate when **SPEC** issues and makes available to **USER** a subsequent release of such materials. In the event of a breach by **USER** of any provision hereof, **SPEC** may terminate this Agreement, and the licenses granted hereunder, in whole or as to any materials upon 30 days written notice unless the default is cured within such 30-day period.

7. **Use of Name.** **USER** may use **SPEC**'s name in conjunction with the publication of **USER's** performance results only with the prior written consent of **SPEC** which will not unreasonably be withheld. It is acknowledged that all trademark and trade name rights in the name are owned exclusively by **SPEC** and such ownership shall be acknowledged in any use of the name by **USER.**

8. **Indemnity.** USER agrees to indemnify and hold SPEC harmless from any claims, expenses or liabilities caused by **USER**'s use of the materials or by **USER's** publication or use of data arising from its use of the materials.

9. **Disclaimer of Warranties.** **SPEC MAKES NO WARRANTIES WHATSOEVER, EXPRESSED OR IMPLIED, RELATING TO THE SALE, USE OR PERFORMANCE OF THE MATERIALS, INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE OF PURPOSE, AND ALL SUCH WARRANTIES ARE HEREBY DISCLAIMED AND EXCLUDED BY SPEC.**

USER RECOGNIZES THAT THE PRODUCTS ARE THE RESULTS OF A COOPERATIVE, NON-PROFIT EFFORT AND THAT SPEC DOES NOT CONDUCT A TYPICAL BUSINESS. USER ACCEPTS THE MATERIALS "AS IS" AND WITHOUT ANY WARRANTY, EXPRESSED OR IMPLIED. HOWEVER, IF ANY MEDIA DEFECT IS FOUND IN THE MATERIALS WITHIN NINETY (90) DAYS AFTER SHIPMENT TO USER, SPEC SHALL REPLACE THE MEDIA WITH A COPY WITHOUT MEDIA DEFECT, WHICH SHALL BE USER'S SOLE REMEDY.

10. **Limitation of Liability.** **SPEC** assumes no liability with respect to the materials, including liability for infringement of intellectual property rights, negligence, or any other liability.

SPEC is not aware of any infringement of copyright or patent that may result from its transfer to **USER** of the materials. If **USER** receives any notice of infringement, such notice shall be immediately communicated to **SPEC** who will take immediate action to evaluate the claim and, if practicable, modify the materials as necessary to avoid infringement. **USER** waives any claim against **SPEC** in the event of such infringement.

SPEC's total liability for any reason shall not exceed the amount of the fee paid to **SPEC**. In no event will **SPEC** be liable for any indirect, special, incidental or consequential damages arising out of or in connection with this agreement, including the sale, use or performance of the materials, even if **SPEC** shall have knowledge of the possibility of such potential.

11. **Export Assurance.** In respect to the Export Administration Regulations (EAR) of the United States Department of Commerce, and in further consideration of all the present and future technical data to be disclosed by **SPEC**, **USER** hereby gives assurance to **SPEC** that **USER** will not knowingly, without prior written authorization from the Department of Commerce, Bureau of Export Administration, export or re-export or otherwise disclose directly or indirectly, either the technical data received from **SPEC** or the direct product of such technical data to any country in the Country Group Q, S, W, Y, Z as defined in the EAR. The countries falling under the above mentioned groups, as of the date of this agreement include:

Albania
Anguilla
Argentina
Bolivia
Bulgaria
Cambodia
Cayman Islands
Chile
Columbia
Cuba
Czech & Slovak Republic
Ecuador**
(including the Galapagos Islands)
Falkland Islands (Islas Malvinas)
French Guiana **
(including Inini)**
Guadeloupe
Guyana
Laos
Latvia
Libya
Lithuania
Martinque
Mongolian People's Republic
Montserrat
North Korea
Paraguay
Peru
Poland
Romania
Saint Kitts-Nevis
Saint Lucia
San Vincent and the Grenadines
Surinam
Turks and Caicos Islands
Union of Soviet Socialist Republics **
(includes all geographic areas formerly part of USSR)**
Uruguay
Venezuela
Vietnam

This is subject to change and should be checks by USER before any export or re-export. (The term "direct product" refers to the immediate product (including processes and services) produced directly by the use of the technical data.

12. **Governing Law.** This agreement shall be construed and governed in accordance with the laws of the State of California. If any term of this agreement is declared void or not enforceable by any competent court of jurisdiction, all other terms shall remain in effect.

13. **No Waiver.** The failure of either party to enforce any term herein or to take action in the event of any violation of these terms shall not be deemed a waiver by that party as to the subsequent enforcement of any rights.