THE PERFECT CLUB BENCHMARKS: EFFECTIVE PERFORMANCE EVALUATION OF SUPERCOMPUTERS

The PERFECT Club[1]
M. Berry,[2] D. Chen,[2] P. Koss,[2]
D. Kuck,[2] S. Lo,[2] Y. Pang,[2]
L. Pointer,[2] R. Roloff,[2]
A. Sameh,[2] E. Clementi,[3]
S. Chin,[3] D. Schneider,[3] G. Fox,[4]
P. Messina,[4] D. Walker,[4]
C. Hsiung,[5] J. Schwarzmeier,[5]
K. Lue,[6] S. Orszag,[6] F. Seidl,[6]
O. Johnson,[7] R. Goodrum,[8]
J. Martin[9]

[1]PERFECT stands for PERFormance Evaluation for Cost-effective Transformations. Any results presented should be regarded as the specific contributions of the individual authors.

[2]Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Urbana, Illinois.

[3]IBM, Kingston, New York.

[4]California Institute of Technology, Pasadena, California.

[5]Cray Research, Inc., Chippewa Falls, Wisconsin.

[6]Princeton University, Princeton, New Jersey.

[7]University of Houston and Houston Area Research Center (HARC).

[8]HNSX Supercomputers, Inc., Houston, Texas.

[9]IBM T. J. Watson Research Center, Yorktown Heights, New York.

The International Journal of Supercomputer Applications, Volume 3, No. 3, Fall 1989, pp. 5–40.

Summary

This report presents a methodology for measuring the performance of supercomputers. It includes 13 Fortran programs that total over 50,000 lines of source code. They represent applications in several areas of engineering and scientific computing, and in many cases the codes are currently being used by computational research and development groups. We also present the PERFECT Fortran standard, a set of guidelines that allow portability to several types of machines. Furthermore, we present some performance measures and a methodology for recording and sharing results among diverse users on different machines. The results presented in this paper should not be used to compare machines, except in a preliminary sense. Rather, they are presented to show how the methodology has been applied, and to encourage others to join us in this effort. The results should be regarded as the first step toward our objective, which is to develop a publicly accessible data base of performance information of this type.

Introduction

While there is a general awareness that supercomputer systems are faster, costlier, and have larger memory hierarchies than other computer systems, such characteristics merely imply the existence of great potential power. How much of that power can be harnessed is the central theme of performance evaluation. A significant body of literature documenting the use of benchmarking as a means for evaluating supercomputer performance has developed (Mueller-Wichards and Gentzsch, 1982; Uhr, 1984; Lubeck, Moore, and Mendez, 1985; Dongarra, Martin, and Worlton, 1987; Neves and Simon, 1987).

Unfortunately, much of the literature focuses on ad hoc approaches to evaluation of systems rather than on potential standardization of the benchmark process (Martin and Riganati, 1988). If benchmarking is to mature sufficiently to meet the requirements of system architects as well as application and algorithm developers, it must address standardization issues as well as various caveats and specific considerations that can reduce general applicability. For example, Denning and Adams (1988) have noted the particular difficulty that arises when comparing different architectures and have urged the development of a benchmark methodology for this purpose. Nevertheless, benchmarking continues to provide one of the few recognized means of acquiring useful performance information about complex systems running complex tasks (Jamieson, 1988). A report prepared by the National Research Council of the National

"Much of the literature focuses on ad hoc approaches to evaluation of systems rather than on potential standardization of the benchmark process. If benchmarking is to mature sufficiently to meet the requirements of system architects as well as application and algorithm developers, it must address standardization issues as well as specific considerations that can reduce general applicability."

Academy of Sciences, considering the importance of supercomputing to the development of science and the deficiencies inherent in the techniques currently used, encouraged an emphasis on the methodology and metrics applied to the evaluation of supercomputing systems (Infante, 1986).

A number of benchmark collections have already been developed. In particular, the more well-known sets include the Livermore Fortran kernels (McMahon, 1988), the Argonne LINPACK measurements (Dongarra, 1987), the Los Alamos benchmarks (Lubeck, 1988), and the NAS kernels from NASA/Ames (Bailey and Barton, 1985). One of the more troubling aspects of the state of benchmarking relative to these collections is that there is no well-defined relationship among their results. In the best case, if it were possible to argue that each was fully representative of a particular workload, then the relationship among the results would lose relevance, and we would be left interpreting specific results relative to specific workloads. In this case, for example, the Livermore kernels would have little meaning outside the Livermore environment and the Los Alamos benchmarks would have merit only in the context of Los Alamos. Unfortunately, it is difficult to assert a strong correlation between benchmark kernels, which may in fact be representative only of the computationally intensive inner loops of a workload, and the full workload. Without solid grounding even in specific cases, generalizing the results beyond the environments in which they were developed becomes significantly more problematic.

The general approach followed by the PERFECT Club develops ideas that have been described elsewhere (Kuck and Sameh, 1987; Martin and Mueller-Wichards, 1987). We have chosen to focus on applications, since it is the users of supercomputers, the developers of large-scale applications, who drive the need for ever increasing processing power. In characterizing the applications, particular emphasis is placed on understanding the underlying algorithms, in the hope that there is greater generality at the algorithmic level. That is, applications representing varied computational disciplines may rely on the same basic set of algorithms. The analysis of optimization strategies (and associated software transformations) applicable to these algorithms and their

subsequent impact on overall application performance constitutes our quest in PERFformance Evaluation for Cost-effective Transformations (PERFECT).

The concept of measuring performance using a suite of applications is not new and has been advocated by a number of computer manufacturers for many years. Some manufacturers have developed their own internal suite of applications and have published data on the characteristics of the application workload as well as performance measurements. However, oftentimes proprietary software from third-party vendors is used and performance comparisons are limited to a family ofarchitectures available from a particular manufac-turer. One goal of the PERFECT Club is to provide aset of portable, public-domain codes that can be executed on a variety of architectures from a variety of manufacturers.

Certainly the goal of this study is to establish guidelines for benchmarking not only within classes of similar supercomputer systems, but also across architectural models. As we have noted, this is an onerous task. Even at the algorithmic level very different techniques may be required to achieve efficient execution on distinct architectures. Although we do not claim to have a panacea, we do present a methodology that holds promise of establishing a reasonable framework for further investigations. In this report, we present preliminary results, as well as a number of caveats and issues that must be addressed as the work matures.

"We have chosen to focus on applications, since it is the users of supercomputers, the developers of large-scale applications, who drive the need for ever increasing processing power. In characterizing the applications, particular emphasis is placed on understanding the underlying algorithms, in the hope that there is greater generality at the algorithmic level."

Caveats

Although application-based benchmarking holds considerable promise in that its relationship to actual workloads can be constructed in a more representative fashion than can a set of computational kernels, it still contains numerous pitfalls. In this preliminary effort, all applications were selected to be computationally intensive. That is, because of the difficulty of porting applications that contain a significant amount of input/output (I/O) we chose not to consider the I/O problem at this stage. Because real applications often require this component of a system, and in fact the efficiency with which I/O is handled can be a critical feature of system performance, the results presented here should not be extrapolated beyond the element of reality that they represent.

"Although application-based benchmarking holds considerable promise in that its relationship to actual workloads can be constructed in a more representative fashion than can a set of computational kernels, it still contains numerous pitfalls. The results presented here should not be extrapolated beyond the element of reality that they represent."

Another portability issue that influences the ability of applications to be run on numerous systems involves the sizes of problems that are executed. Problem sizes have been restricted to permit execution within *reasonable* time on the spectrum of systems included in the study. It was desirable to have programs execute in minutes or hours, rather than days, on even the least powerful systems. Although this is a positive attribute of the collection from the standpoint of making the project feasible for the benchmarkers, it should be noted that the restriction of problem size may have a negative impact on the more powerful systems. For example, restricting problem size may inhibit the potential performance of a vector processor that is dependent on long vectors to achieve speed, or of a parallel system with a large number of processors, in which it would be beneficial to amortize the cost of the serial content of an application by spreading an extremely large calculation across many individual processors. Such restrictions may also nullify large initialization times that could be present for larger problem sizes. Given a benchmark program whose initialization phase is composed mainly of serial computations, the speedups obtained via multiprocessing will be distorted.

Because each of the applications was originally written for a particular architecture, it should be noted that in no case are we comparing speedups due to tuning to optimal serial algorithms. This need not be a problem; in fact, by understanding its implications we may be able to increase our qualitative understanding of the variances in the architectures. In some sense, the greater the speedup that is achieved by tuning for a particular system, the less suitable might the original implementation of the given algorithm or application have been for the system in question. Understanding the mismatches between applications and architectures can facilitate the partitioning of the applications and architectures into representative classes.

Related to the observation that each application was written originally as an implementation on a particular architecture is the caveat that in the methodology we are proposing there is no absolute way to omit the subjectivity involved in evaluating the level of effort expended in the process of optimizing the programs being tested.

Although the use of diaries to explain code modifications and the ranking of relative efforts invested in the modifications described is an attempt to quantify a very qualitative judgment, it does not eliminate the problem completely. We also note that optimization, as we use it, is a relative term and not an absolute one. The optimization process involves successive program modifications, each of which is intended to improve code performance in a measurable way. Thus, the performance comparisons reflect an inseparable aggregate of hardware and software characteristics, skill levels, and amount of effort expended in the optimization process.

The final caveat is that in this study we do not consider the implications of system throughput, nor do we attempt to define benchmarks for this purpose. Timings are taken in stand-alone environments, which seldom reflect the environment of an end user. Nevertheless, there is considerably more control of the environment surrounding the measurements in stand-alone situations. One obvious extension of this effort would be to define a similar methodology for multiuser systems in which throughput is as important a consideration as is the individual turnaround time associated with the applications under investigation.

The PERFECT Club Benchmark Suite

This section gives a brief history of each of the codes in the PERFECT benchmark suite, together with an overview of the algorithms employed. A list of the 13 PERFECT Club benchmark programs along with their representative applications is given in Table 1.

ADM

This three-dimensional code supplied by the IBM Kingston group simulates pollutant concentration and deposition patterns in lakeshore environments by solving the complete system of hydrodynamic equations. The code was developed by Christidis (1986) at the Atmospheric and Oceanic Science Department of the University of Michigan, and was subsequently ported to the LCAP system at IBM Kingston (Christidis and Sonnad, 1987). The advection-diffusion equation for the transport, diffusion, and deposition of pollutants is also in-

cluded in the model. Model variables are integrated using a time-splitting method. The advection term is treated by a Fourier pseudospectral method in the x and y directions, coupled with the second-order explicit midpoint (or *leapfrog*) rule in time. Convection-diffusion processes in the vertical direction are treated using a semi-implicit Crank-Nicolson method.

ARC3D

ARC3D is a robust, general-purpose, implicit finite-difference code for analyzing three-dimensional fluid flow problems by solving the Euler and Navier-Stokes equations. ARC3D was developed by Pulliam and coworkers at the NASA Ames Research Center for use on the CDC 7600, but recently the code has been run on a number of vector machines, such as the CDC Cyber 205, the CRAY X-MP, and the IBM 3090. The first version of ARC3D became generally available in 1978 (Pulliam and Steger, 1980), and more recently several improvements have been incorporated into the code (Pulliam and Steger, 1985). The 1984 *diagonal algorithm* version of ARC3D, which is used as the PERFECT Club benchmark, can be used for both inviscid and viscous flows, and for steady and unsteady flows. In addition, ARC3D can be run on any smoothly varying curvilinear mesh, and vectorizes well.

The equations are discretized onto a curvilinear mesh, and the method of Beam and Warming (1976) is used to cast the equations in implicit form. Three-point central difference approximations are used, leading to a pentadiagonal system of equations. Numerical dissipation terms are added to improve stability, and a turbulent viscosity model is used to produce the steady-state flow field.

The test problem used as the benchmark was turbulent flow past an ellipsoid. The ratio of major to minor axes was 6 to 1, the Mach number $M_\infty = 1.2$, and the angle of attack, $\alpha = 19°$. The Reynolds number for the flow is 2.25×10^4. The computational grid has 40 points along the surface in the streamwise direction, 21 points around the circumference from the windward to the leeward side, and 30 points radially. The ARC3D code was supplied to the PERFECT Club by the Cray Research group.

BDNA

The BDNA code makes use of the BIOMOL package (Swamy and Clementi, 1987a) for performing molecular dynamics simulations of biomolecules in water. This package aims at an understanding of the hydration, structure, and dynamics of nucleic acids and, more broadly, the role of water in the operation of biological systems. BDNA was developed by Swamy and Clementi (1987b) of IBM Kingston, and is based on an earlier molecular dynamics code by Laaksonen.

The MCY potential function (Matsuoka, Clementi, and Yoshimine, 1976) is used to describe the water-water interaction. Atom-atom potential functions of the form

$$u_{ij}(R) = \frac{A}{R^{12}} - \frac{B}{R^6} + C\frac{q_i q_j}{R}$$

are used to describe water-ion, water–nucleic acid, and ion-ion interactions. The translational equation of motion of each molecule is solved by a Gear fifth-order predictor-corrector method. The rotational motion of water is represented using quaternions, and the rotational equations of motion are solved using the second-order quaternion method using a sixth-order predictor-corrector algorithm.

BDNA is a simulation of the hydration structure of potassium counterions and water in B-DNA. There are 1,500 water molecules and 20 counterions placed in a parallelopiped box of dimension 33.8 Å × 44.0 Å × 44.0 Å. One complete helical turn of B-DNA consisting of 10 base pairs is considered.

DYFESM

This code, submitted by the CSRD group, is a two-dimensional, dynamic, finite element code for the analysis of symmetric anisotropic structures (see Noor and Camin, 1976; Noor and Peters, 1985) developed by Noor and Peters at the NASA Langley Research Center, Hampton, Virginia, for the CRAY X-MP. An explicit leapfrog temporal method with substructuring is used to solve for the displacements and stresses, along with the velocities and accelerations at each time step. At each time step, accelerations are computed via a precondi-

tioned conjugate gradient method in which the preconditioning matrix is chosen to be the orthotropic part of the global stiffness matrix, with all nonorthotropic terms set to zero. This code has been previously used for determining the response of laminated anisotropic shallow panels having quadrilateral planforms.

FLO52Q

FLO52Q, provided by the Princeton group, was developed by Jameson at Princeton University (see Jameson, 1983, and references therein) to analyze the transonic inviscid flow past an airfoil by solving the unsteady Euler equations. The two-dimensional domain is discretized into quadrilateral cells. In the case considered there were 128 intervals around the profile, and 32 radially. Application of the Euler equations to these cells yields a set of coupled ordinary differential equations, to each of which a dissipative term is added to suppress nonphysical oscillations near regions of steep gradients, such as shock waves. The set of differential equations is solved by incorporating a simple sawtooth multigrid strategy into the multistage time-stepping scheme. This results in a more rapid convergence to the steady-state solution than that obtained by using just a single grid. Three of five available grid sizes are used. This scheme is good at modeling shock waves, and the algorithm can be vectorized with a vector length equal to a grid dimension.

MDG

This code was developed by the IBM Kingston group. MDG performs a molecular dynamics calculation of 343 water molecules in the liquid state at room temperature and pressure. The code, written by Lie and Clementi (1986) of IBM, uses the MCY configuration interaction potential for rigid water-water interactions (Matsuoka, Clementi, and Yoshimine, 1976), and extends it to include the effects of intramolecular vibration. MDG can be used to predict a wide variety of static and dynamic properties of liquid water.

The code solves the Newtonian equations of motion for 343 water molecules in a cubical box using the sixth-order predictor-corrector method of Gear (1971). The total potential is the sum of the intramolecular and in-

termolecular potentials. The intramolecular potential used is that denoted by DMBPT(∞) in the double-excitation infinity-order many-body perturbation theory calculations of Bartlett, Shavitt, and Purvis (1979). The intermolecular potential used is the MCY potential.

MG3D

This seismic migration code, which was written for a CRAY X-MP by Reshef and Kessler (1989) at Tel Aviv University, Tel Aviv, Israel, is used to investigate the geological structure of the Earth. Signals of different frequencies are emitted at the Earth's surface, and after interacting with the geological strata, are received at another point on the surface. The data collected at the surface by this technique can then be used to extrapolate backward in time to get a three-dimensional image of the structure below the surface. The data are Fourier-transformed in time, and the depth extrapolation in the z direction can proceed independently (in parallel) for each frequency.

The advancement in the z direction is done with a one-step integration algorithm. Most of the execution time is usually spent performing the fast Fourier transformations (FFTs), although for realistic problems significant I/O time can be expected. The specific problem size used by the PERFECT Club was selected to minimize the I/O requirements of the job, as this was one area that was chosen to be excluded from our measurements. Our sample problem used an $x \times y$ grid of 125 $\times$ 120 points, respectively, and we considered one step (downward in the Earth) in the z direction. Data from 1,500 time values were Fourier-transformed and a frequency cutoff was used in order to reduce the problem size in the frequency domain.

OCEAN

This code, which solves the dynamical equations of a two-dimensional Boussinesq fluid layer (see Curry et al., 1984) was submitted by the Princeton group. This code is needed in order to study the chaotic behavior of free-slip Rayleigh-Bénard convection (Lorenz, 1963). The equations of motion for a Boussinesq fluid layer are

$$\frac{\partial \mathbf{v}}{\partial t} = \mathbf{v} \times \omega - \nabla\left(p + \frac{v^2}{2}\right) + \alpha T\hat{\mathbf{z}} + \nu\nabla^2\mathbf{v}, \quad (1)$$

$$\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T = \beta w + \kappa\nabla^2 T, \quad (2)$$

$$\nabla \cdot \mathbf{v} = 0. \quad (3)$$

Here $\mathbf{v} = (u,v,w)$ is the velocity field at $\mathbf{x} = (x,y,z)$, $\omega = \nabla \times \mathbf{v}$ is the vorticity, p is the pressure, T is the deviation of the temperature from the conduction profile $-\beta z$, ν is the kinematic viscosity, κ is the thermal conductivity, and α is the thermal expansion coefficient. A spectral method (see Orszag, 1971) is used to solve Eqs. (1), (2), and (3). The nonlinear terms are evaluated by fast-transform methods with aliasing terms removed. Time-stepping is done by a leapfrog scheme for the nonlinear terms, and an implicit scheme (Crank-Nicolson or backward Euler method) for the viscous terms. The pressure term is computed in Fourier representation by local algebraic manipulation of the incompressibility constraint Eq. (3). The flow is assumed to occur in a *box* region with periodic and free-slip (no-stress) impermeable boundary conditions applied. The original version of this code was designed for a CRAY-1 computer system.

QCD

Quantum chromodynamics (QCD) is the gauge theory of the strong interaction that 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 short 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 these 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 et al., 1987, for a summary). The PERFECT Club QCD benchmark uses the pseudo-heat-

bath algorithm of Cabbibo and Marinari (1982) to update the SU(3) matrices on the lattice links. This algorithm uses a Monte Carlo technique to generate a chain of configurations which are distributed with a probability proportional to $\exp(-S(U))$, where $S(U)$ is the action of configuration U. If the only contributions to the action come from the gauge field, then the action is local. The inclusion of dynamical fermions gives rise to a nonlocal action, which complicates the algorithm. The PERFECT Club QCD code, written at the California Institute of Technology for the iPSC/1, MARK III, and NCUBE/10 hypercubes, ignores the effects of dynamical fermions. The code therefore represents a pure-gauge model in the *quenched* approximation.

A major component of the QCD code is the updating of the SU(3) matrices associated with each link in the lattice, and it is this operation that is benchmarked in the PERFECT timings. Two basic operations are involved in updating the lattice, namely, the multiplication of SU(3) matrices, and the generation of pseudo-random numbers. The lattice size for the benchmark was taken as 8^4.

SPEC77

This global spectral model for simulating atmospheric flow, which was submitted by the CSRD group, was originally developed at the National Meteorological Center (NMC) for a CDC Cyber 205, and has been described in detail (Sela, 1980, 1982). The spectral method of solving PDEs was pioneered by Orszag (1970) and Eliasen, Machenhauer, and Rasmusen (1970); subsequently, Bourke (1974) showed how to apply it to atmospheric modeling. Only the forecast module of the NMC code is used in the benchmarking.

The formulation of the model is based on expressing the unknown functions in terms of their spherical harmonic expansions in the horizontal direction. In the vertical direction, the quadratic conserving finite difference formula of Arakawa and Mintz (1974) is used. A semi-implicit, backward time integration scheme is applied, and initial conditions are obtained from a spectral Hough operational analysis (Flattery, 1967). The ocean interacts with the atmosphere by means of evaporation and sensible heating, and the moisture cycle consists of

large-scale precipitation, and convection of the type described by Kuo (1965).

SPICE

This code is a general-purpose circuit simulation program for nonlinear direct-current, nonlinear transient, and linear alternating-current analysis developed by the Integrated Circuits Group of the Electronics Research Laboratory and the Department of Electrical Engineering and Computer Science at the University of California, Berkeley (Nagel, 1975). The CSRD group submitted the SPICE version, SPICE2G.6, used in the PERFECT Club benchmarking suite. SPICE is widely used in industry, and is one of the most important tools in the computer-aided design of integrated circuits. Within SPICE, problems are formulated as stiff differential-algebraic equations. The numerical methods of solution include stiffly stable numerical integration algorithms, the Newton-Raphson method for solving the nonlinear algebraic equations, and sparse linear system solvers (see Newton and Sangiovanni-Vincentelli, 1983).

TRACK

The TRACK code was developed at the California Institute of Technology by Gottschalk and coworkers 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, 1988, 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. The sequential Fortran version used in the PERFECT benchmarks was written by Gottschalk at the beginning of 1988.

If the target's acceleration is assumed to be known, then the path of an individual object is described fully by a four-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 kinematical model with a stochastic acceleration component. The output from this phase is then

passed to the precision parameter estimation module, which uses Newton-Raphson iteration to solve an equation giving a more precise estimate of the launch parameter vector.

In a multitarget scenario, sensor data points are associated with tracks by means of the *track-splitting* algorithm described by Gottschalk (1987). 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 that discards the poorer of any duplicate tracks. The initialization of new tracks is managed by a separate module called the *batch mode initializer*, which limits possible new tracks to a plausible set.

TRFD

This code is a kernel simulating the computational aspects of a two-electron integral transformation, and forms part of the HONDO quantum mechanical package developed by the IBM Kingston group (Dupuis et al., 1988). The evaluation of these types of integral transformations is a necessary first step in computing correlated wavefunctions (Watts and Dupuis, 1987), and is used in determinations of molecular electronic structure.

The two-electron integral transformation transforms the electron repulsion integrals from the atomic orbital (AO) basis set to the molecular orbital (MO) basis set according to a fourth-order tensor transformation equation. The evaluation of the integral transformation is formulated as a series of matrix multiplications.

The PERFECT Club Methodology

In this section, we present the Fortran language standard used to port the PERFECT Club benchmarks to a variety of supercomputers (see Tables A1 and A2 in the Appendix). The verification and performance measures used for benchmark evaluation are discussed. We conclude the section with a discussion on software transformations that can be used to optimize the benchmarks.

"The Fortran 77 language standard that was adopted by the PERFECT Club can be classified into five categories: syntax, documentation, declarations, input/output (I/O), and general programming techniques."

PROGRAMMING LANGUAGE STANDARD

All 13 application programs in the PERFECT Club benchmark suite were successfully ported and verified on the machines listed in Table 2 (with the exception that only MDG and QCD were ported to the iPSC/1, MARK III, and NCUBE/10 hypercubes). The Fortran 77 language standard that was adopted by the PERFECT Club (after a few porting iterations) can be classified into five categories: syntax, documentation, declarations, input/output (I/O), and general programming techniques. The modification of the original Fortran 77 program in Table 1 according to the following guidelines ensured a successful run on the machines in Table 2. We note that many of the requirements reflect well-known ANSI Fortran 77 standards (special thanks to Bruce Leasure of Kuck and Associates Inc. of Savoy, Illinois, for his help with the Fortran 77 guidelines).

Syntax

A PROGRAM statement should be supplied.
Hollerith fields should be avoided.
An apostrophe (') should be the only string delimiter.
Redundant operators (A + + B) should be removed.
Multiple operators (A/ − B) should be ordered (A/(− B)).
Special characters (e.g. $) should not be used in variable names.
Variable names should be no more than six characters long.
Avoid using doubly-nested DO-loops defined by only one label. Branches (GOTOs) to the label between the loops may be fatal.

Documentation

Multiple subroutine RETURNs should be well documented.
All unavoidable machine-specific constants and/or intrinsic functions must be documented.

Declarations

Declare all variables.
Avoid the use of EQUIVALENCE. If not possible, do not EQUIVALENCE character and non-character data.

Use REAL*8 and REAL*4 for DOUBLE PRECISION and REAL declarations, respectively (CDC Cyber 205 is the only exception).
All timing variables should be declared REAL*4.
Use only one length per CHARACTER declaration.
Declarations in COMMON blocks should be made in the following order: COMPLEX*16, COMPLEX*8, REAL*8, REAL*4, INTEGER, Other.

Input/Output

An OPEN statement must exist for each file accessed.
Variables (e.g., IN, IOUT) should be used for logical units in all READs and WRITEs.
The use of IOSTAT = and ERR = should be avoided.
The storage requirements for all input, output, and temporary files should be well documented. The use of external I/O devices (e.g., Cray Solid State Disk) should also be indicated.

Programming Techniques

Do not use a FUNCTION name as a variable name.
Avoid the use of GOTO statements.
Do not use the intrinsic routines ICHAR and CHAR.
Ensure that any string manipulation involves a string of nonzero length.
Ensure that all variables are assigned before they are referenced.
Use a SAVE statement for reuse of local variables, when executing with stacks (e.g., CRAY X-MP/416).

Although there is no guarantee that the above language standard is suitable for all machines, it does reflect a timely effort by the PERFECT Club to identify several major porting errors one may encounter on a given machine.

PROGRAM VERIFICATION

Beyond the porting of each of the PERFECT Club benchmark programs, particular attention was given to the numerical accuracy of the computed solution(s) on each machine. In order to obtain a simple yet reliable measure for the correctness of the computed results, only a few critical variables or constants (e.g., pressure, kinetic energy) were examined from the output. The va-

"Beyond the porting of each of the PERFECT Club benchmark programs, particular attention was given to the numerical accuracy of the computed solution(s) on each machine. A self-checking component of the benchmark suite enables a programmer to determine immediately the correctness of the outputs without extensive analysis of the output data."

lidity of these quantities is measured according to

$$\frac{|X_o - X_e|}{|X_e|} \leq \epsilon,$$

where X_o and X_e are the observed and expected results, respectively, and ϵ is the maximum allowable relative error. Since the outputs of a few of the benchmark programs in Table 1 are quite sensitive to machine round-off errors (not all machines in Table 2 conform to the IEEE floating-point standard), a large relative error (e.g., $\epsilon = 0.10$) is sometimes allowed. Upon program completion, a *verification file* indicates whether the computed results are VALID or INVALID based upon the relative errors for each critical output variable. This self-checking component of the PERFECT Club benchmark suite enables a programmer to determine immediately the correctness of the outputs without extensive analysis of the output data.

PERFORMANCE MEASURES

The fundamental measurements that were recorded for each of the programs in Table 1 on the machines in Table 2 were the CPU and wall-clock times, Initially, these measures were taken on the original (or *baseline*) code without any manual optimization by the programmer using 64-bit arithmetic (32-bit precision used on Hypercube machines listed in Table 2 [iPSC/1, MARK III, NCUBE/10] for QCD, and on the Alliant FX/8 for optimized versions of QCD and SPEC77). Optimization achieved by means of a compiler or a preprocessor, however, was certainly allowed. A dedicated computing environment was used for all measurements with the exception of those made on the NEC SX-2. In all cases, the elapsed wall-clock time measurements were used instead of CPU times for purposes of computing the normalized execution rates for each program in the suite. In those cases for which elapsed wall-clock time measurements were not available, CPU times were used to determine normalized execution rates. The input data sets utilized by the programs were chosen to yield *reasonable* run times on the full spectrum of machines. Specifically, after reviewing the measurements taken for a few preliminary runs with various data sets on each machine, a single data set or set of parameters was se-

lected for measuring the performance of each program on all the machines.

Having recorded the baseline measurements for the benchmark suite, our attention was then focused on the use of software transformations (see Table 3) for manually optimizing the suite. This effort was conducted through the use of *optimization diaries*, which are described in the next section. These diaries simply identify the type, frequency, and performance improvement associated with each transformation used by the programmer to optimize each program on a particular machine. The performance improvement per diary entry, P_i, is measured as

$$P_i = \frac{\Delta T_i}{T_{i-1}}, \tag{4}$$

where $\Delta T_i = T_i - T_{i-1}$ is the change in the run time (wall-clock or CPU) from diary entry $i - 1$ to i (relative to the run time for T_{i-1}), and T_0 is the baseline measurement mentioned above. The performance improvement for the ith transformation given the implementation of the previous $i - 1$ transformations is given by

$$\overline{P}_i = \frac{T_0}{T_i}. \tag{5}$$

The ratio $\overline{P}_i$, in effect, measures the successive speed improvement as recorded in the optimization diaries. To assess the amount of human effort required to achieve each P_i, the author of the diary indicated the amount of time (minutes, hours, days, weeks, or months) needed to implement the corresponding transformation. These assessments were then ranked according to degree of difficulty by each programmer for the compilers and machines being used (see Tables A1 and A2 in the Appendix). This type of measurement is, of course, subjective, but by focusing on the rank order of the human effort rather than attempting to determine absolute values, the diary can reveal a hierarchy of cost-effective transformations.

"Beyond the objective timing measurements, attention was focused on the use of software transformations through the use of optimization diaries. These diaries identify the type, frequency, and performance improvement associated with each transformation in the optimization process. Although subjective, the diary can reveal a hierarchy of cost-effective transformations."

SOFTWARE TRANSFORMATIONS

The software transformations listed in Table 3 can significantly affect hardware use as well as the total number of operations required to obtain a result. There are three basic categories of transformations considered.

The first category comprises *compiler directives* (or assertions), which are treated as comment statements in Fortran 77. These directives allow a compiler to safely perform certain operations. Transformations 1, 2, and 3 are in this category.

The second category of transformations consists of software-restructuring techniques that involve source code modifications. A simple example is a local change (say, within one DO-loop) that enables a Fortran 77 compiler to recognize vectorization or concurrency capabilities. A more complicated change, *data blocking*, can alter arrays and loops throughout an entire code. Transformations 4–15 and 18–20 are in this category.

The third category involves algorithm modifications in which a more efficient numerical method (perhaps one that is better suited to vectorization or parallelization) replaces the original method. This final category comprises transformations 16 and 17. In transformation 16, the details of the particular library routine used are, in general, transparent to the user. Subsequently, this transformation may or may not involve actual algorithm modifications. In contrast, transformation 17 is an algorithm replacement performed by the user. An example is the replacement of a linear-congruential random number generator by a parallelizable and vectorizable random number generator.

In summary, these software transformations may be viewed as responses of the individual programmers to the available hardware features or system software. Given two machines of similar architecture, one does not always obtain similar optimization diaries for a particular PERFECT Club benchmark program. Identifying differences in the type and the impact of software transformations used in these diaries should lead to a better understanding of the performance of supercomputers, in general.

Results

In this section, we present the baseline (compiler-optimized only) and *hand*-optimized measurements for the PERFECT benchmark suite (Table 1) on the machines listed in Table 2. Figures 1 through 6 illustrate the range of performance obtained when the PERFECT Club benchmarks are simply ported and compiler-optimized

Table 1
The PERFECT Club Benchmark Suite

Program	Application	Lines of Source	PERFECT Club Sponsor	Original Source Machine
ADM	Air pollution	6,142	IBM	IBM 3090
ARC3D	Computational fluid dynamics	3,605	Cray Res.	CDC 7600
BDNA	Nucleic acid simulation	3,962	IBM	IBM 3090
DYFESM	Structural dynamics	7,599	CSRD	CRAY X-MP
FLO52Q	Computational fluid dynamics	2,250	Princeton	CRAY 1
MDG	Liquid water simulation	1,231	IBM	IBM 3090
MG3D	Seismic migration	2,754	Cray Res.	CRAY X-MP
OCEAN	Computational fluid dynamics	4,215	Princeton	CRAY 1
QCD	Quantum chromodynamics	2,342	Cal. Tech	Mark I
SPEC77	Weather simulation	3,880	CSRD	CDC Cyber 205
SPICE	Circuit simulation	18,504	CSRD	CDC 6600
TRACK	Signal processing	3,770	Cal. Tech	Mark III
TRFD	Quantum mechanics	479	IBM	IBM 3090

Table 2
Machines Used by the PERFECT Club

Machine	Description	Peak MFLOPS
Alliant FX/8	8 vector processor mini-supercomputer	94.4
CDC Cyber 205	2-pipe vector processor (memory-to-memory)	200
CRAY X-MP/416	4 vector processor supercomputer	940
IBM 3090-600S	6 vector processor supercomputer	800
iPSC/1	Intel 80286/80287-based hypercube with 64 processors	25.6
MARK III	Hypercube with 32 MC68020 processors	12.8
NCUBE/10	Hypercube with 512 custom scalar processors	209
NEC SX-2	Vector-pipelined uniprocessor supercomputer	1,300

Table 3
Software Transformations Used in Optimization Diaries

No.	Description
1.	Compiler directive: concurrent loop (*microtask*)
2.	Compiler directive: concurrent subroutine call (*macrotask*)
3.	Compiler directive: ignore data dependence
4.	Vectorize loop
5.	Remove data dependence
6.	Common subexpression elimination
7.	Modify for concurrent subroutine call
8.	Removal of redundant operations
9.	Modify for stride one memory access
10.	Loop fusion
11.	Loop unrolling
12.	Loop interchange
13.	Cache management
14.	Register management
15.	Replace Fortran statements by assembly language code
16.	Library subroutine
17.	Change of algorithm
18.	Loop restructuring: move loop inside/outside of subroutine
19.	Subroutine elimination: contents moved inline
20.	I/O modification: formatted to unformatted I/O
21.	Unknown diary modification

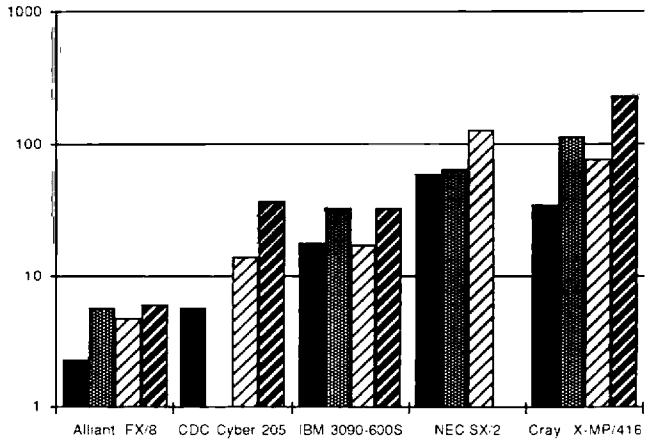

Fig. 1 Preliminary baseline and optimized execution rates for the PERFECT benchmarks: ADM, ARC3D, and BDNA

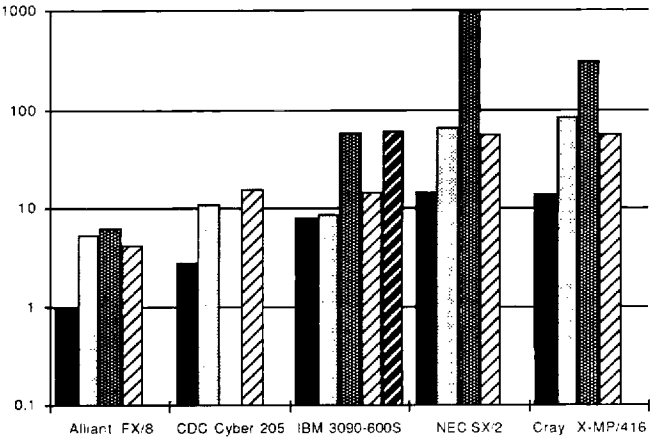

Fig. 2 Preliminary baseline and optimized execution rates for the PERFECT benchmarks: DYFESM and FLO52Q

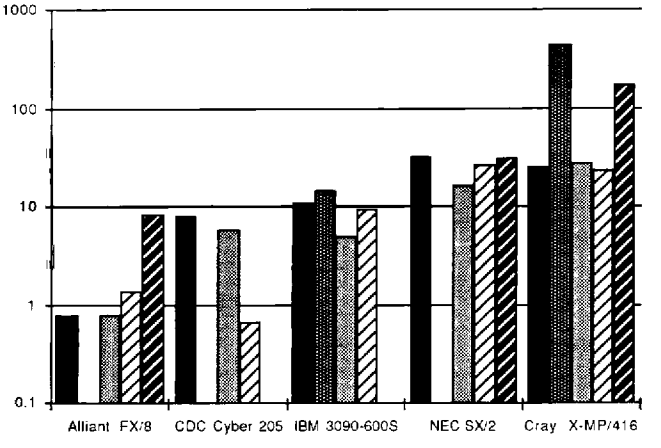

Fig. 3 Preliminary baseline and optimized execution rates for the PERFECT benchmarks: MG3D, OCEAN, and SPEC77

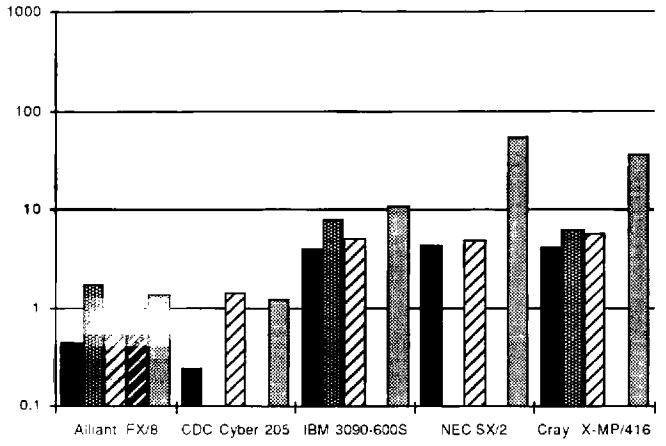

Fig. 4 Preliminary baseline and optimized execution rates for the PERFECT benchmarks: SPICE, TRACK, and TRFD

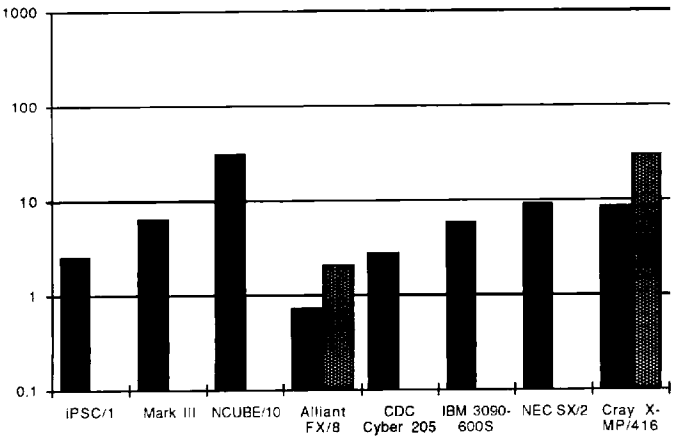

Fig. 6 Preliminary baseline and optimized execution rates for the PERFECT benchmark: QCD

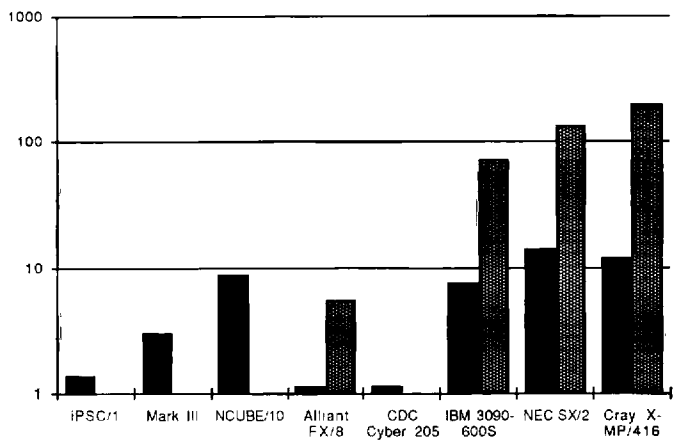

Fig. 5 Preliminary baseline and optimized execution rates for the PERFECT benchmark: MDG

on each machine. In these figures the current performance gained through manual optimization is indicated by (OPT). The relative performance measures reported in Figures 1 through 6 were computed by dividing the total floating-point operation count as measured by a hardware monitor on one CPU of a CRAY X-MP/416 for the original source codes in the PERFECT Club benchmark suite by the execution times on each machine. In this way, the operation counts are simply used as a normalization factor, allowing comparison of applications containing significantly different amounts of work. Specifically, by applying this normalization we are able to compare execution *rates* rather than execution times. The former are preferable to the latter since they do not vary with the total volume of work in the problems, and thus are more convenient indicators of performance. Given the difficulty of acquiring accurate floating-point operation counts (baseline or optimized) for all the codes on each machine listed in Table 2, our intention is to measure the relative performance of each machine in solving the particular application problems reflected by the suite. We note that all eight processors of the Alliant FX/8 were used for both baseline and optimized versions of the codes, and that only 256 nodes of the NCUBE/10 were used for the baseline QCD measurements. Whereas all baseline measurements for the

Table 4
Summary of PERFECT Club Activity

	Machine							
Program	Alliant FX/8	CDC Cyber 205	CRAY X-MP/416	IBM 3090-600S	iPSC /1	MARK III	NCUBE /10	NEC SX-2
ADM	B	B	**B**	B				**B**
ARC3D	O	B	**O**	O				**O**
BDNA	B	B	**B**	O				**B**
DYFESM	O	B	**O1**	O1				**O**
FLO52Q	O	O	**O**	O				**B**
MDG	O	B	**O**	O	B	B	B	**O**
MG3D	B	B	**O**	O1				**B**
OCEAN	B	B	**B**	B				**B**
QCD	O	B	**O1**	B	B	B	B	**B**
SPEC77	O	B	**O**	B				**O**
SPICE	O	B	**O1**	O1				**B**
TRACK	O	B	**B**	B				**B**
TRFD	B	B	**B**	B				**B**

B denotes a baseline measurement, O denotes an optimized measurement (diary), O1 denotes optimization on one processor only.

CRAY X-MP/416 utilized only one CPU, most of the optimized measurements recorded (with the exception of DYFESM, QCD, and SPICE) used all four CPUs. For the IBM 3090-600S, optimized measurements on all six processors were obtained for ARC3D, BDNA, FLO52Q, and MDG, while optimized one-processor measurements were recorded for DYFESM and MG3D. All baseline measurements for the IBM 3090-600S reflect single processor runs.

As mentioned in the previous section, the particular software transformations used to hand-optimize many of the PERFECT Club programs were recorded in optimization diaries. Table 4 indicates the machines and programs for which diaries have been collected, and Figures 1 through 6 reveal the improved execution rates that have been achieved through this activity. To illustrate how this improvement was achieved on various machines, we discuss (collectively) diaries for the PERFECT Club benchmarks: DYFESM, FLO52Q, and MDG.

DYFESM

The optimization of the two-dimensional dynamic finite element program, DYFESM, primarily relied upon the

Table 5
Diary Entries for DYFESM on the Alliant FX/8, CRAY X-MP/416, and IBM 3090-600S

Machine	Entry	Transformation(s) (see Table 3)	Description (Dominant Subroutines Affected)
Alliant FX/8	1	16	BLAS2 matrix-vector multiplication. (MATMUL)
	2	16	Block Cholesky algorithm using BLAS3 matrix-matrix kernels inserted. (CHOSOLV)
	3	10,2	Loops fused to yield one parallel loop with a concurrent subroutine call. (MNLBYX)
CRAY X-MP/416 (1 CPU)	1	19	Contents of several short subroutines moved inline to remove excessive calling overhead. (ASSEMR, GETEU)
	2	16	BLAS2 matrix-vector multiplication. (MATMUL)
	3	12	Loops interchanged to remove dotproducts. (MNLBYX)
	4	12,11	Same as 3, but with loop unrolling.
	5	4	Cholesky algorithm redesigned to remove dotproducts and ensure columnwise operations. (CHOSOLV)
	6	11	Unroll loops in Cholesky solver. (CHOSOLV)
IBM 3090-600S (1 CPU)	1	8	Eliminated multiplications by zero. (MATMUL)
	2	4	Vectorized matrix-vector multiplication loops (MATMUL). Scalar compiler optimization (S3) used on all short loops.
	3	16	BLAS2 matrix-vector multiplication. (MATMUL)

availability of efficient numerical software libraries. In Table 5, we list the specific transformations from Table 3 as recorded in diaries for the Alliant FX/8, CRAY X-MP/416, and IBM 3090-600S (only one processor was used in the optimization of DYFESM on the CRAY X-MP/416 and IBM 3090-600S).

We note the use of optimized matrix-vector (BLAS2) and matrix-matrix (BLAS3) kernels (see Gallivan, Jalby, and Meier, 1987) as well as an efficient Cholesky algorithm (for solving symmetric positive definite linear systems) on all three machines.

The improvement ratios, Eq. (4) and Eq. (5), for the transformations in Table 5 are illustrated in Figures 7 and 8, and the initial profile and final profile (after diary transformations) for DYFESM, which illustrate the breakdown in execution time for the most dominating subroutines, are given in Table 6. Although an optimized time for DYFESM on the NEC SX-2 has been reported, no diary is available. The numbers contained within the bars in Figures 7 and 8 correspond to the transformations listed in Table 3.

In Figures 7 and 8 we observe the significant performance improvement (large P_i's) resulting from the use of optimized library subroutines (transformation 16)

Table 6
Initial and Final Profiles for DYFESM on the Alliant FX/8, CRAY X-MP/416 (1 processor), and IBM 3090-600S (1 processor)

	% Execution Time					
	Initial			Final		
Subroutine	Alliant FX/8	CRAY X-MP/416	IBM 3090-600S	Alliant FX/8	CRAY X-MP/416	IBM 3090-600S
ASSEMR	4	5	—	11	—	4
BLOCKMX	0.4	18	—	1	27	—
CHOSOLV	28	17	20	0.4	21	36
GETEU	1	5	—	2	—	—
MATMUL	33	11	54	0.5	—	—
MNLBYX	27	15	11	1	15	20
SOLVXDD	2	7	3	5	17	5

ASSEMR = assemble right-hand-side terms.
BLOCKMX = element-level multiplication.
CHOSOLV = triangular system solver for preconditioner.
GETEU = load elemental displacements.
MATMUL = matrix multiplication.
MNLBYX = form nonlinear terms.
SOLVXDD = conjugate gradient method.

on both the Alliant FX/8 and the CRAY X-MP/416. While the elimination of dotproducts proved to be significant on the CRAY X-MP/416, the use of concurrent subroutine calls proved to be equally important on the Alliant FX/8. On the IBM 3090-600S, it is interesting to note that $\overline{P}_3 \sim 2$ is essentially achieved through trivial code modifications and careful compiler usage.

To quantify the human effort needed to incorporate the transformations of Table 5 into DYFESM, we have chosen to rank-order (increasingly more difficult and time-consuming from left to right) the transformations in Figure 8 according to the programmer's assessment of how his or her time was spent in the hand-optimization phase. Hence, as one would expect, the use of library subroutines is extremely cost-effective since relatively little human effort is required to yield a substantial overall improvement rate.

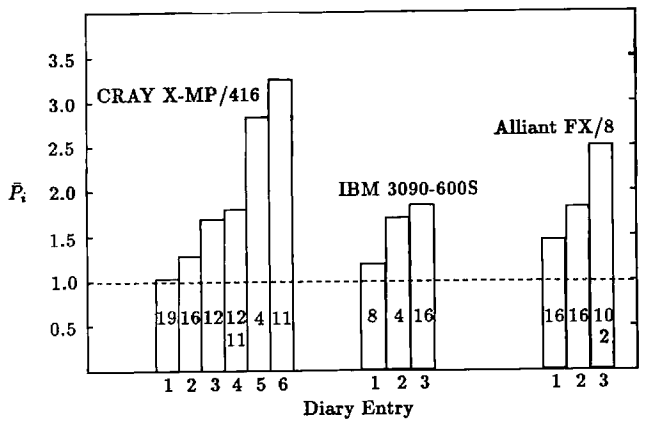

Fig. 7 Cumulative performance improvement (Eq.5) for DYFESM

FLO52Q

The optimization of the transonic flow benchmark program, FLO52Q, on the Alliant FX/8, CDC Cyber 205, CRAY X-MP/416, and IBM 3090-600S consisted primarily of software-restructuring transformations (second category under "Software Transformations"). The initial and final profiles for this highly vectorizable program are listed in Table 7. No dramatic changes from the initial to the final profile are observed on any of the four machines. We do note, however, the disparity in the most dominant subroutine (initial profile only) among the four machines (either EFLUX or PSMOO). The specific transformations used in the diaries for FLO52Q on the Alliant FX/8, CDC Cyber 205, CRAY X-MP/416, and IBM 3090-600S are listed in Tables 8 and 9. The key to improving the performance of this benchmark program on each of these machines lies in the vectorization or parallelization of the multitude of doubly-nested DO-loops. The numbers of loop iterations for these nested loops are usually of differing magnitudes. For example, one loop can have as many as 41 iterations while the other may have as few as 9 iterations. From Figures 9 and 10, we note that the most substantial improvement ($P_1 = 0.66, \overline{P}_1 = 2.8$) is achieved by the use of microtasking (in conjunction with loop fusion to obtain larger vector lengths) on the CRAY X-MP/416. On

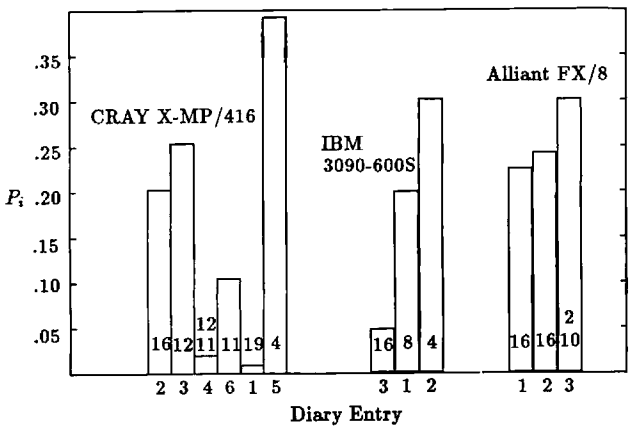

Fig. 8 Rank-ordered (in ascending human effort) performance improvement per diary entry (Eq.4) for DYFESM

Table 7
Initial and Final Profiles for FLO52Q on the Alliant FX/8, CDC Cyber 205, CRAY X-MP/416, and IBM 3090-600S

	% Execution Time							
	Initial				Final			
Subroutine	Alliant FX/8	CRAY X-MP/416	CDC Cyber 205	IBM 3090-600S	Alliant FX/8	CRAY X-MP/416	CDC Cyber 205	IBM 3090-600S
DFLUX	18	16	14	24	22	17	16	10
EFLUX	18	24	29	21	22	27	17	22
EULER	12	16	—	12	15	17	8	11
PSMOO	36	18	43	26	20	21	33	30

DFLUX = blended 2nd- and 4th-order damping.
EFLUX = computation of Euler fluxes.
EULER = multistage Runge-Kutta advance of flow variables.
PSMOO = additional smoothing calculated implicitly.

the Alliant FX/8, concurrent subroutine calls and limited doubly-nested loop restructuring to allow both concurrent iterations of the outermost DO-loop and total vectorization of the innermost DO-loop yielded only small reductions in total time. Although most vector lengths are rather short (≤ 41), some improvement was achieved on the CDC Cyber 205, with the use of a vector preprocessor and explicit *linked-triad* vector operations. The former (along with the third diary entry) yielded the most significant individual transformation improvement ($P_1 = P_3 = 0.32$) in Figure 10. On the

Table 8
Diary Entries for FLO52Q on the Alliant FX/8 and CRAY X-MP/416

Machine	Entry	Transformation(s) (see Table 3)	Description (Dominant Subroutines Affected)
Alliant FX/8	1	2,11,8	Subroutine split into two separate subroutines, both of which can be called concurrently. Loops within these two new subroutines were unrolled. (PSMOO)
	2	1,10,11	Doubly-nested DO-loops restructured into optimal concurrent-outer vector-inner (COVI) DO-loop pairs. (DFLUX)
	3	11	Loops unrolled in order to alleviate overhead within existing COVI loops. (EFLUX)
CRAY X-MP/416	1	1,10	Loop fusion and insertion of microtasking directives before outermost DO-loops. (DFLUX, EFLUX, EULER, PSMOO)

Table 9
Diary Entries for FLO52Q on the IBM 3090-600S and CDC Cyber 205

Machine	Entry	Transformation(s) (see Table 3)	Description (Dominant Subroutines Affected)
IBM 3090-600S	1	4,8,10	Function redefined to allow vectorization in some loops (STEP), while other loops enforced to be scalar (PSMOO). Loop fusion and redundant division removal. (BCFAR, DFLUX, PSMOO, STEP)
	2	7	Concurrent subroutine calls instrumented. (EULER, DFLUX, PSMOO, STEP)
CDC Cyber 205	1	4	VAST-2 vector preprocessor used to vectorize DO-loops. (DFLUX, EFLUX, EULER, PSMOO)
	2	4	Loops reindexed and redesigned for explicit linked-triad (vector ← vector + scalar × vector) operations. (PSMOO)
	3	6,8	Redundant vector operations removed and contiguous array sections extended. One-to-one correspondence between array elements enforced by removing from COMMON, inserting subroutine arguments, and dynamically dimensioning arrays. (DFLUX, EFLUX, EULER)

IBM 3090-600S, parallelization of the four most time-consuming routines (using concurrent subroutine calls) yielded a significant improvement ($\overline{P}_2 = 1.8$). In general, for the highly vectorizable programs in the PERFECT Club benchmark suite, such as FLO52Q, relatively few transformations in Table 3 were selected in the optimization diaries. This is due in large part to the success of the individual compilers in generating efficient vector and/or concurrent DO-loops.

MDG

Although many of the diaries recorded for a particular benchmark program in Table 1 were unique to the respective machines in Table 2, the molecular dynamics benchmark, MDG, provides one example in which practically the same transformations were chosen for optimization on the Alliant FX/8, CRAY X-MP/416, IBM 3090-600S, and NEC SX-2. Diaries for MDG on the iPSC/1, MARK III, and NCUBE/10 machines are not included in our comparison. The similarity in the MDG diaries can be largely attributed to the few dominating subroutines listed in Table 10 (initial and final profiles on the CRAY X-MP/416 were not available). While the

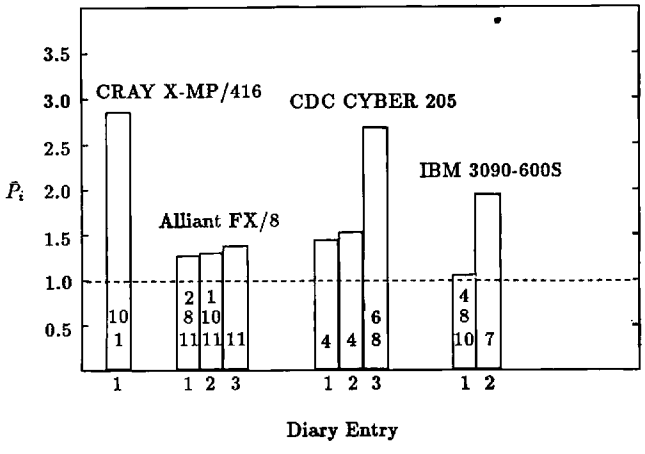

Fig. 9 Cumulative performance improvement (Eq.5) for FLO52Q

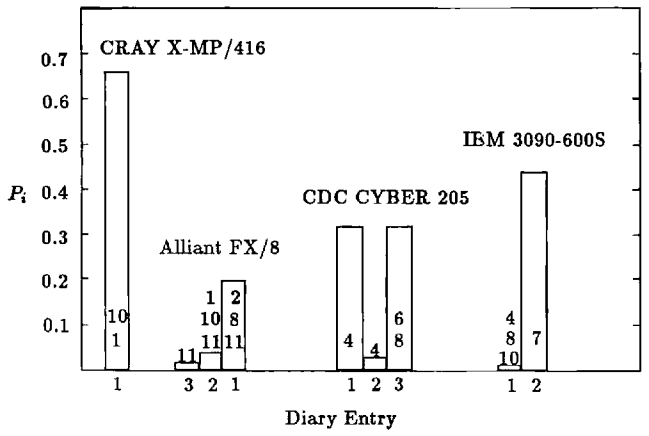

Fig. 10 Rank-ordered (in ascending human effort) performance improvement per diary entry (Eq.4) for FLO52Q

original MDG program is predominantly sequential on all the machines listed in Tables 11 and 12, the use of simple array indexing schemes can easily produce a vectorizable implementation. For example, the innermost loop of the doubly-nested interaction-forces loops (subroutine INTERF), which together determine the forces on each atom of each molecule, can be easily vectorized on all four machines if scalars within the loop are expanded to arrays so as to improve data-reuse via efficient vector register management. The suppression of redundant operations associated with the determination of relative distances between particles was also commonly used to aid in vectorization.

Since the outermost loop in INTERF can be redesigned to iterate over distinct pairs of molecules (innermost loop determines the contributions to the forces on each atom within a pair), parallelization on the Alliant FX/8, CRAY X-MP/416, and IBM 3090-600S is simply achieved by scheduling sets of independent molecule pairs across the number of available processors. Attention was given to both optimal load balancing and redundant array operations associated with the scheduling selected. The overhead from excessive calls to a child subroutine, CSHIFT, was removed by moving that subroutine inline (transformation not applied on the Alliant FX/8).

Figures 11 and 12 illustrate the performance improvement achieved using the vector/parallel strategies discussed above. As shown, the greatest impact made on each machine is primarily due to the optimization of the

Table 10
Initial and Final Profiles for MDG on the Alliant FX/8, IBM 3090-600S, and NEC SX-2

	% Execution Time					
	Initial			Final		
Subroutine	Alliant FX/8	IBM 3090-600S	NEC SX-2	Alliant FX/8	IBM 3090-600S	NEC SX-2
CSHIFT	37	23	29	22	3	—
INTERF	47	71	65	0.1	87	83
POTENG	4	4	5	—	8	6

CSHIFT = impose cyclic boundary conditions.
INTERF = compute intermolecular forces.
POTENG = potential energy.

Table 11
Diary Entries for MDG on the Alliant FX/8 and CRAY X-MP/416

Machine	Entry	Transformation(s) (see Table 3)	Description (Dominant Subroutines Affected)
Alliant FX/8	1	1,8,11	Parallelize the outermost interaction-forces loop using concurrent calls to a subroutine constructed from statements in the original subroutine. Scalar variables are expanded to multidimensional arrays for vectorization. Loop-invariant array operations removed. (INTERF)
	2	3	Enforce concurrent and vector operations using compiler directives to ignore potential data dependence. (INTERF)
	3	1,8,11	Parallelize the outermost DO-loop using strategy in entry 1. (POTENG)
CRAY X-MP/416	1	3,4,8,12,19	Excessive number of calls to CSHIFT subroutine removed by moving body of this child subroutine into the parent subroutine, INTERF. Loop interchanging done to yield larger vector lengths. Compiler directives used to enforce vectorization where potential data dependence exists. Redundant array operations removed. (CSHIFT, INTERF, POTENG)
	2	1	Microtask (parallelize) outermost loops and (INTERF, POTENG)

Table 12
Diary Entries for MDG on IBM 3090-600S and NEC SX-2

Machine	Entry	Transformation(s) (see Table 3)	Description (Dominant Subroutines Affected)
IBM 3090-600S	1	4,12,14,19	Excessive number of calls to CSHIFT subroutine removed by moving body of this child subroutine into the parent subroutine, INTERF. Expand scalar variables to multidimensional arrays for data reuse in vector registers. Loops interchanged to yield longer vector lengths. Doubly-nested intermolecular forces loop removed from POTENG and inserted in INTERF. (CSHIFT, INTERF, POTENG)
	2	2	Force calculations parallelized using concurrent subroutine calls. (INTERF)
NEC SX-2	1	19	Excessive number of calls to CSHIFT removed by moving subroutine inline. (CSHIFT, INTERF)
	2	4,11,14	Unrolled innermost loop and expanded scalar variables for vector operations within this loop. DO-loops fully unrolled to array dimensions. (INTERF)
	3	4,11,14	Unrolled innermost loop and expanded scalar variables for vector operations. (POTENG)

"The quotation of a single performance number for any advanced architecture is a disservice to manufacturers and users. The complexity of these machines requires a new level of detail in measurement and comprehension of the results. This paper has laid the groundwork for what we hope will be a new era in benchmarking and evaluating the performance of computers."

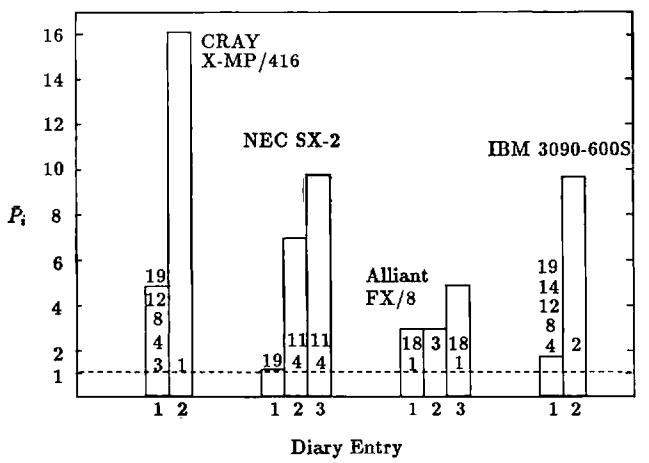

Fig. 11 Cumulative performance improvement (Eq.5) for MDG

dominant subroutine, INTERF. The parallelization of this subroutine was considered as the most difficult transformation to implement on the Alliant FX/8 and IBM 3090-600S. Less effort was required using *micro-tasking* constructs on the four CPUs of the CRAY X-MP/416 (see Fig. 12).

Conclusions

This paper has laid the groundwork for what we hope will be a new era in benchmarking and evaluating the performance of computers. The complexity of these machines requires a new level of detail in measurement and comprehension of the results. The quotation of a single number for any given advanced architecture is a disservice to manufacturers and users alike, for several reasons. First, there is a great variation in performance from one computation to another on a given machine; typically the variation may be one or two orders of magnitude, depending on the type of machine. Second, the ranking of similar machines often changes as one goes from one application to another, so the best machine for circuit simulation may not be the best machine for computational fluid dynamics. Finally, the performance depends greatly on a combination of compiler characteristics and the human effort expended on obtaining the results.

Our objective in forming the PERFECT Club was first to assemble and then make available a suite of applications codes that represent typical applications. Furthermore, we wanted them to be portable, state-of-the-art codes. We feel that we have taken a first step in this direction. More application areas and several codes per application should be included to give a better representation of what real users run.

The measurements we have made are very preliminary—simply time and execution rate for one data set per code. This data set was chosen to allow all the machines to participate in a reasonable way. We attempted to avoid I/O effects and were successful, with the exception of the seismic migration program, MG3D, on some machines. In the future, the effects of I/O should be measured, as should a number of other architectural features.

In future optimization diaries, inconsistencies in the

frequency of measurements and execution profiles would need to be more closely monitored in order to make realistic comparisons in the strategies adopted across the various machines. Although most programmers are reluctant to report them, transformations that yield longer run times ($P_i < 1$ in Eq. 5) should also be recorded. The exchange of diaries is also encouraged so that a collective optimization of a given program is achieved across all machines. Such an exchange could facilitate the future design of both efficient and portable application benchmark programs. Finally, the use of available hardware and software tools is strongly encouraged for the collection of accurate execution rate measurements, for example.

Ultimately, we believe that an online data base that would contain many programs, diaries, and measured performance information for a number of machines should be made publicly available. Notice that if one excluded the source programs from the data base, proprietary software packages could be included in the data base as an aid in code selection.

The magnitude of such an effort should not be underestimated. To reach the modest objective reported in this paper required 18 months of elapsed time, and many person-months of effort by a number of the coauthors of this paper. While much of the work is straightforward, reaching an agreement on a number of the points required considerable effort. In this sense, we feel

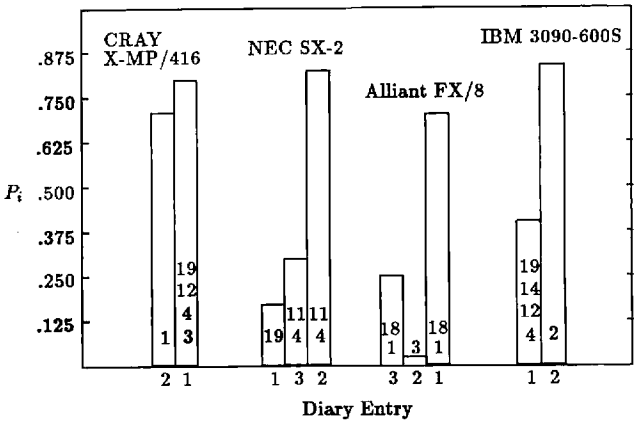

Fig. 12 Rank-ordered (in ascending human effort) performance improvement per diary entry (Eq.4) for MDG

Appendix

Table A1

Machines Used by PERFECT Club Members

	Machine			
Descriptor	Alliant FX/8	CRAY X-MP/416	IBM 3090-600S	CDC Cyber 205
Location	CSRD	Cray Research	IBM-Gaithersburg	Princeton U.
Clock cycle (nanoseconds)	170	8.5	15.5	20
Memory size (megabytes)	32	128	256	32
Number of processors (total used)	8/8	4/(1–4)	6/(1–6)	1
Processor connection	Shared memory	Shared memory	Shared memory	—
Number of add/multiply pipes (per processor)	1/1	1/1	1/1	2/2
Operating system	Concentrix 3.0	UNICOS 3.0.10	VM/XA SP2 CMS 5.5	VSOS 2.3.5
Fortran compiler	FX/Fortran 3.1.33	CFT77 2.0	VS Fortran 2.3.0	FTN200

Appendix
Table A2
Machines Used by PERFECT Club Members

	Machine			
Descriptor	**NEC SX-2**	**NCUBE/10**	**iPSC/1**	**Mark III**
Location	HARC		Cal. Tech	
Clock cycle (nanoseconds)	6	145	—	—
Memory size (megabytes)	128	0.5/node	0.5/node	4/node
Number of processors (total used)	1	2**N(N = 0–9)	2**N(N = 0–6)	2**N(N = 0–5)
Processor connection	—	Hypercube	Hypercube	Hypercube
Number of add/multiply pipes (per processor)	4/4	—	—	—
Operating system	SXOS 2.13	AXIS/VERTEX	XENIX 3.4/NX OS	UNIX 5.2.2
Fortran compiler	Fortran 77 SX Rev 36	CFG V1.6	RM V2.20A	Silicon Valley V2.4

that the initial step we have taken will help expedite progress toward our goal.

ACKNOWLEDGMENT

The work of the authors at the Center for Supercomputing Research and Development (CSRD), University of Illinois at Urbana-Champaign, was supported in part by the National Science Foundation under grant US NSF MIP-8410110, the U.S. Department of Energy under grant US DOE-DE-FG02-85ER25001, the Air Force Office of Scientific Research under grant AFOSR-85-0211, and by a donation from IBM.

The work of the authors at the California Institute of Technology, Pasadena, was supported in part by the National Science Foundation under grant NSF ASC-8719501 and the U.S. Department of Energy under grant US DOE-DE-FG03-85ER25009.

The work of the authors at Princeton University, Princeton, New Jersey, was supported in part by the Air Force Office of Scientific Research under grant AFOSR-49620-87-C-0036, and the Office of Naval Research under grant ONR-N00014-82-C-0451.

BIOGRAPHY

The PERFECT Club, now in its second year, is a collaboration between investigators whose three primary goals are to develop a methodology for creating a Fortran benchmark suite of portable applications programs, to produce a collection of realistic benchmark application

codes that have been measured and verified on a variety of supercomputers, and to identify and categorize the types of transformations used in optimizing the suite on each computer through the use of optimization diaries.

The University of Illinois Center for Supercomputing Research and Development (CSRD) serves as the coordinating host for the club. New members are invited to serve in any of four capacities: benchmarking the suite on a new computer; providing additional application codes to be included in the suite; providing architectural insights into the performance of the suite; and providing algorithmic insights into the performance of the suite.

The senior members of the PERFECT Club are the authors of this paper: from CSRD at the University of Illinois at Urbana-Champaign, Illinois, Michael Berry, Ding-Kai Chen, Peter Koss, David Kuck, Sy-Shin Lo, Youlan Pang, Lynn Pointer, Richard Roloff, and Ahmed Sameh; from IBM, Kingston, New York, Enrico Clementi, Steven Chin, and David Schneider; from the California Institute of Technology, Pasadena, Geoffrey Fox, Paul Messina, and David Walker; from Cray Research, Inc., Chippewa Falls, Wisconsin, Chris Hsiung and Jim Schwarzmeier; from Princeton University, Princeton, New Jersey, Kok-Meng Lue, Steve Orszag, and Fred Seidl; from the University of Houston, Olin Johnson; from HNSX Supercomputers Inc., Houston, Texas, Richard Goodrum; and from the IBM T. J. Watson Research Center, Yorktown Heights, New York, Joanne Martin.

For more information concerning activities and membership, please contact: The PERFECT Club, The Center for Supercomputing Research and Development, University of Illinois, 305 Talbot Laboratory, 104 S. Wright Street, Urbana, IL 61801-2932. Phone: (217) 333-6223.

SUBJECT AREA EDITOR

Bill L. Buzbee

REFERENCES

Arakawa, A., and Mintz, Y. 1974. The UCLA Atmospheric Circulation Model. Los Angeles: University of California, Dept. of Meteorology.

Bailey, D., and Barton, J. 1985. The NAS kernel benchmark program. NASA Technical Memorandum 86711. Moffett Field, Calif.: Ames Research Center, National Aeronautics and Space Administration.

Bartlett, R., Shavitt, I., and Purvis, G. 1979. *J. Chem. Phys.* 71:281.

Beam, R., and Warming, R. 1976. An implicit finite

difference algorithm for hyperbolic systems in conservative law form. *J. Comput. Phys.* 22:87.

Bourke, W. 1974. A multi-level spectral model. Formulation and hemispheric integrations. *Monthly Weather Review* 102:687.

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

Christidis, Z. 1986. Hydrodynamic mesoscale modeling of atmospheric transport and pollutant deposition in the vicinity of a lake. Ph.D. thesis, Atmospheric and Oceanic Science Department, University of Michigan, Ann Arbor.

Christidis, Z., and Sonnad, V. 1987. Parallel implementation of a pseudo-spectral method on a loosely coupled array of processors. Technical Report KGN-143. Kingston, N.Y.: IBM.

Curry, J., Herring, J., Loncaric, J., and Orszag, S. 1984. Order and disorder in two- and three-dimensional Bénard convection. *J. Fluid. Mech.* 174:1.

Denning, J., and Adams, G. 1988. Research questions for performance analysis of supercomputers. In *Performance evaluation of supercomputers,* edited by J. Martin. Amsterdam: North-Holland, p. 403.

Dongarra, J. 1987. Performance of various computers using standard linear equations software in a Fortran environment. Technical Report MCS-RM-23. Argonne, Ill.: Argonne National Laboratory.

Dongarra, J., Martin, J., and Worlton, J. 1987. Computer benchmarking: paths and pitfalls. *IEEE Spectrum* July: 38.

Dupuis, M., Watts, J., Villar, H., and Hurst, G. 1988. HONDO: version 7.0 documentation. *Quantum Chemistry Program Exchange Bulletin* 8:2 (IBM Technical Report KGN-169, Kingston, N.Y.).

Eliasen, E., Machenhauer, B., and Rasmusen, E. 1970. On a numerical method for integration of the hydrodynamic equations with a spectral representation of the horizontal fields. Report No. 2. Copenhagen: University of Copenhagen, Institute for Theoretical Meteorology.

Flattery, T. 1967. Hough functions. Technical Report 20. Chicago: University of Chicago, Dept. of Geophysical Sciences.

Gallivan, K., Jalby, W., and Meier, U. 1987. The use of BLAS3 in linear algebra on a parallel processor with a hierarchical memory. *SIAM J. Sci. Statist. Comput.* 8(6):1079.

Gear, C. 1971. *Numerical initial value problems in ordinary differential equations.* Englewood Cliffs, N.J.: Prentice-Hall.

Gottschalk, T. 1987. A new multi-target tracking model. Presented at the third conf. on hypercube concurrent computers and applications, ACM, New York City (Caltech Report C^3P-480).

Gottschalk, T. 1988. Concurrent multiple target tracking. Presented at the third conf. on hypercube

concurrent computers and applications, ACM, New York City (Caltech report C³P-567).

Infante, E. (Committee Chairman). 1986. *An agenda for improved evaluation of supercomputer performance.* Washington, D.C.: Energy Engineering Board, Commission on Engineering and Technical Systems, National Research Council (2101 Constitution Ave. NW).

Jameson, A. 1983. Solution of the Euler equations for a two-dimensional transonic flow by a multigrid method. *Appl. Math. Comput.* 13:327.

Jamieson, L. 1988. Using algorithm characteristics to evaluate parallel architectures. In *Performance evaluation of supercomputers,* edited by J. Martin. Amsterdam: North-Holland, p. 21.

Kuck, D., and Sameh, A. 1987. A supercomputing performance evaluation plan. Presented at the first internat. conf. on supercomputing, Athens, Greece.

Kuo, H. 1965. On formation and intensification of tropical cyclones through latent heat release by cumulus convection. *J. Atmos. Sci.* 22:40.

Lie, G., and Clementi, E. 1986. Molecular dynamics simulation of liquid water with an ab initio flexible water-water interaction potential. *Phys. Rev. A* 33:2679.

Lorenz, E. 1963. Deterministic non-periodic flow. *J. Atmos. Sci.* 20:130.

Lubeck, O. 1988. Supercomputer performance: the theory, practice, and results. Technical Report LA-11204-MS. Los Alamos, N.M.: Los Alamos National Laboratory.

Lubeck, O., Moore, J., and Mendez, R. 1985. A benchmark comparison of three supercomputers: Fujitsu VP-200, Hitachi S810/20, and CRAY X-MP/2. *Computer* 18:12.

Martin, J., and Mueller-Wichards, D. 1987. Supercomputer performance evaluation: status and directions. *J. Supercomput.* 1:87.

Martin, J., and Riganati, J. 1988. On benchmarking standards for high performance computing. *IEEE Comput.* 21:9:68.

Matsuoka, O., Clementi, E., and Yoshimine, M. 1976. *J. Chem. Phys.* 64:1351.

McMahon, F. 1988. The Livermore Fortran kernels test of the numerical performance range. In *Performance evaluation of supercomputers,* edited by J. Martin. Amsterdam: North-Holland, p. 143.

Mueller-Wichards, D., and Gentzsch, W. 1982. Performance comparisons among several parallel and vector computers on a set of fluid flow problems. Technical Report IB 262-82 R01. Göttingen, West Germany: DFVLR.

Nagel, L. 1975. SPICE2: a computer program to simulate semiconductor circuits. Memorandum ERL-M520. Berkeley: University of California, College of Engineering, Electronics Research Laboratory.

Neves, K., and Simon, H. 1987. Supercomputer performance evaluation:

benchmarking applications on supercomputers. In *Proceedings of the second internat. conf. on supercomputing*. St. Petersburg, Fla.: International Supercomputing Institute, p. 374.

Newton, R., and Sangiovanni-Vincentelli, A. 1983. Relaxation-based circuit simulation. *IEEE Trans. ED* 30(9):1184.

Noor, A., and Camin, R. 1976. Symmetry considerations for anisotropic shells. *Comp. Meth. Appl. Mech. Engrg.* 9:31.

Noor, A., and Peters, J. 1985. Model-size reduction techniques for the analysis of symmetric anisotropic structures. *Engrg. Comp.* 2(4):285.

Orszag, S. 1970. Transform methods for calculation of vector coupled sums: application to the spectral form of the vorticity equations. *J. Atmos. Sci.* 27:890.

Orszag, S. 1971. Numerical simulation of incompressible flows within simple boundaries, Galerkin (spectral) representations. *Stud. Appl. Math.* 50:293.

Otto, S., Baillie, C., Ding, H-Q., Apostolakis, J., Gupta, R., Kilcup, G., Patel, A., and Sharpe, S. 1987. Lattice gauge theory benchmarks. Report C^3P-405R. Pasadena: California Institute of Technology.

Pulliam, T., and Steger, J. 1980. Implicit finite-difference simulations of three-dimensional compressible flow. *AIAA J.* 18:159.

Pulliam, T., and Steger, J. 1985. Recent improvements in efficiency, accuracy, and convergence for implicit approximate factorization algorithms. AIAA-85-0360. Presented at the 23rd aerospace science meeting, January 14–17, Reno, Nevada.

Reshef, M., and Kessler, D. 1989. Practical implementation of three-dimensional post-stack depth migration. *Geophysics,* in press.

Sela, J. 1980. Spectral modeling at the National Meteorological Center. *Monthly Weather Review* 180:1279.

Sela, J. 1982. The NMC spectral model. Technical Report NWS 30 (May). National Oceanic and Atmospheric Administration.

Swamy, K., and Clementi, E. 1987a. BIOMOL—a molecular dynamics simulation package for nucleic acid hydration. Report KGN-97. Kingston, N.Y.: IBM.

Swamy, K., and Clementi, E. 1987b. Hydration structure and the dynamics of B- and Z-DNA in the presence of counterions via molecular dynamics simulations. Report KGN-94. Kingston, N.Y.: IBM.

Uhr, L. 1984. On benchmarks: dynamically improving experimental comparisons. Computer Sciences Technical Report No. 571. Madison: University of Wisconsin-Madison.

Watts, J., and Dupuis, M. 1987. Towards efficient parallel computation of correlated wave functions. Implementation of the two-electron integral transformation on the LCAP parallel supercomputer. Technical Report KGN-100. Kingston, N.Y.: IBM.