JOURNAL OF COMPUTATIONAL PHYSICS 92, 296–312 (1991)

# The Parallel Fourier Pseudospectral Method

RICHARD B. PELZ

*Mechanical and Aerospace Engineering, Rutgers University, Piscataway, New Jersey 08855-0909*

Received November 7, 1988; revised August 3, 1989

Parallel algorithms of the Fourier pseudospectral method are presented for the solution of the unsteady, incompressible Navier–Stokes equations. The only major operation that requires parallelization is the multidimensional FFT. In tests performed on a 1024-node hypercube computer, an efficiency of about 83% is obtained for a three-dimensional problem with mesh size $128^3$. The all-FORTRAN code requires 17 s per timestep, rivalling rates obtained from optimized codes on current supercomputers. © 1991 Academic Press, Inc.

## I. INTRODUCTION

Spectral methods have been useful tools for the examination of many unsteady flow phenomena including turbulence [18], shear layers [15], vortex interactions [13, 14], global weather prediction [8] and astrophysical applications [1, 11]. In such methods each flow variable is represented as a finite series expansion in orthogonal polynomials which results in exponential-order accuracy, provided the flow is smooth. The pseudospectral method, which is perhaps the most widely used method for nonlinear problems, employs series expansions as interpolation functions for the flow variables. An excellent review of these methods is given by Canuto *et al.* [3].

Simulation of realistic flows possessing a large number of degrees of freedom, is presently unrealistic because of the large amount of computational resources required. This is true for spectral methods even though they require fewer mesh points for the same accuracy than most other methods. Parallel processing shows the promise of delivering necessary resources at a relatively low cost. The crucial question is, in spite of the global nature of spectral methods, can high parallel performance be obtained?

The purpose of the present work is to investigate the performance of pseudospectral methods on distributed-memory, MIMD computers with a hypercube connection network. The problem is the time-integration of the Navier–Stokes equations for incompressible flows in three dimensions. Domain decomposition is used to parallelize the problem. Algorithms for the simplest and perhaps most effective method, the Fourier pseudospectral method, are devised [16]. While the type of representation in the directions that are distributed across processors is constrained



0021-9991/91 $3.00

to be Fourier, any discretization scheme can be employed in the other directions. For simplicity, however, a Fourier series representation is used in all directions. It is shown that the multidimensional Fourier transform is the only operation in which a significant amount of communication is required. Implementation and experimentation is performed on the NCUBE/1 computer with up to 1024 processors.

An MIMD, distributed-memory computer has many relatively unsophisticated processors, each operating independently with individual codes and operating systems. They are connected loosely by a communication network which uses a fast, user-invoked message-passing system. In the hypercube network each processor is directly connected to $d = \log_2 P$ ($P$ is a power of 2) neighbors. The maximum number of intermediate processors through which a message must pass is $d-1$. Processors are assigned labels, numbers from 0 to $P-1$. A "nearest neighbor" pair is defined as two processors whose binary labels differ in only one place.

The structure of the paper is as follows. In Section II we review the spectral and timestepping schemes. The discussion in Section III is about parallelization using domain decomposition. Methods for performing the parallel multidimensional transforms are presented in Section IV. Results from actual performance tests are presented in Section V. Section VI contains some concluding remarks and extrapolation of performance to future computers.

## II. The Fourier Pseudospectral Method

In this section we review the Fourier pseudospectral method and timestepping scheme for our problem. The Navier–Stokes equations can be written

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{u} \times \boldsymbol{\omega} - \nabla P_0 + \mathbf{f} + \nu \Delta \mathbf{u}, \tag{1}$$

where $\mathbf{u}$ is the velocity field, $\boldsymbol{\omega}$ is the vorticity field, $\nu$ is the dynamic viscosity, and $\mathbf{f}$ is a conservative body force. With unit density, the stagnation pressure, $P_0$, is expressed as $P + u^2/2$, where $P$ is the static pressure.

Taking the divergence of (1) and applying the constraint of incompressibility, we find the expression for stagnation pressure in an incompressible flow is:

$$\Delta P_0 = \nabla \cdot (\mathbf{u} \times \boldsymbol{\omega}). \tag{2}$$

The boundary conditions are that the flow variables be periodic in each direction.

In the Fourier pseudospectral approximation, the solution, defined on an equally spaced mesh, is represented by a finite Fourier series. The coefficients are found by insisting that the series be an interpolation function at the mesh points. Using the discrete Fourier transform the Fourier coefficients are found from the values of the flow variables at the mesh points. The gradient of $\mathbf{u}$ has the form $i\mathbf{k}^{\mathrm{T}}\hat{\mathbf{u}}$, and the divergence has the form $i\mathbf{k} \cdot \hat{\mathbf{u}}$, where $\hat{\mathbf{u}}$ is the Fourier coefficient of $\mathbf{u}$ and $\mathbf{k}$ is

the wavevector. The fast Fourier transform (FFT) provides a rapid way to switch bases from mesh-point (physical space) to coefficient (spectral space) representation.

The convergence of the pseudospectral approximation to the true solution is more rapid than any algebraic power of the mesh size for smooth solutions. The convergence is exponential for analytic solutions [20].

A hybrid explicit/implicit scheme involving leap-frog and Crank–Nicholson methods is used for time advancement. Letting the superscript $n$ denote the time level $n \cdot \Delta t$, the second-order accurate (in time) approximation to (1) is

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^{n-1}}{2\,\Delta t} = (\mathbf{u} \times \boldsymbol{\omega})^n - (\nabla P_0)^n + \mathbf{f}^n + \frac{\nu}{2} \Delta(\mathbf{u}^{n+1} + \mathbf{u}^{n-1}). \tag{3}$$

The timestepping scheme can be written as an explicit step, a step to enforce incompressibility and an implicit step:

$$\mathbf{u}^* = (1 + \nu\,\Delta t\,\Delta)\,\mathbf{u}^{n-1} + 2\,\Delta t(\mathbf{u} \times \boldsymbol{\omega} + \mathbf{f})^n. \tag{4a}$$

$$(\Delta P_0)^n = \nabla \cdot \mathbf{u}^*/2\,\Delta t \tag{4b}$$

$$(1 - \nu\,\Delta t\,\Delta)\,\mathbf{u}^{n+1} = \mathbf{u}^* - 2\,\Delta t(\nabla P_0)^n. \tag{4c}$$

There are four Poisson equations to be solved per timestep in the three-dimensional problem. All derivative and Laplacian operations are performed on the Fourier coefficients. Note that for $\nabla \cdot \mathbf{u}^{n-1} = 0$ and conservative body forces, (4b) is identical to (2). By judicious arrangement of operations, the number of transforms can be kept to a minimum.

## III. Domain Decomposition

Parallelization of the problem is achieved by dividing the domain into $P$ subdomains each with the same number of points and mapping them onto $P$ processors. The number $P$ is factored into $P_x \cdot P_y \cdot P_z$ which are the number of processors that are employed in the decomposition in the $x$, $y$ and $z$ directions, respectively. Figure 1 shows a decomposed domain with $P = 16$, $P_x = 2$, $P_y = 2$ and $P_z = 4$. The subdomains are mapped sequentially onto the hypercube processors.

In physical space the arrays are indexed

$$(1 : nx,\ 1 : ny,\ 1 : nz). \tag{5}$$

Processor $q$ contains the subarray

$$q_x \cdot \frac{nx}{P_x} + 1 : (q_x + 1) \cdot \frac{nx}{P_x},\ q_y \cdot \frac{ny}{P_y} + 1 : (q_y + 1) \cdot \frac{ny}{P_y},\ q_z \cdot \frac{nz}{P_z} + 1 : (q_z + 1) \cdot \frac{nz}{P_z}, \tag{6}$$

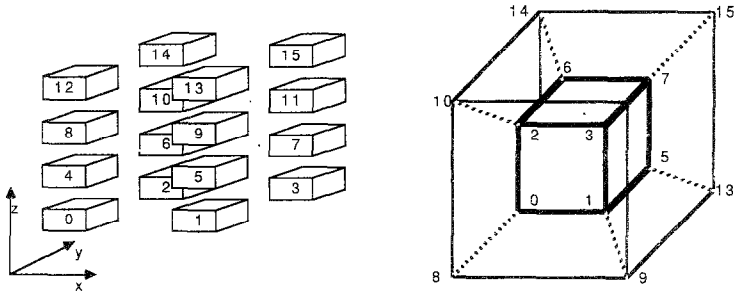


FIG. 1. A domain that is decomposed in three directions and the mapping onto a 16-node hypercube. $P_x = 2$, $P_y = 2$, and $P_z = 4$.

where

$$q_x = q \text{ modulo } P_x$$

$$q_y = \frac{q}{P_x} \text{ modulo } P_y$$

$$q_z = q/P_x P_y.$$

In spectral space the decomposed arrays have the same structure. The wavenumber array for a complex array is

$$\left(0 : \frac{nx}{2} - 1;\ 0 : \frac{ny}{2} - 1,\ 0,\ -\frac{ny}{2} + 1 : -1;\ 0 : \frac{nz}{2} - 1,\ 0,\ -\frac{nz}{2} + 1 : -1\right). \tag{7}$$

In the decomposition, $P_x \leqslant nx/2$, so that complex numbers are not split.

Now we identify which operations in (4) require flow variables from another processors and which do not.

If the flow variables are in spectral space, then the derivative, Laplacian, and inverse Laplacian operations do not require any communication of data between processors. If the flow variables, $\mathbf{u}$ and $\omega$, are in physical, mesh-point representation, the calculation of the nonlinear term, $\mathbf{u} \times \omega$, does not require communication between processors. The only operation in (4) that does require communication between processors is the Fourier transform of vectors distributed across processors. In the next section we shall discuss ways of performing this task.

## IV. The Parallel Multidimensional FFT

Due to the importance in areas such as image and signal processing, there have been many investigations of parallel implementation of the FFT, [4, 5, 10, 19].

There are two competing methods for the multi-dimensional transform. One involves using a one-dimensional, parallel FFT algorithm for vectors distributed across processors and a one-dimensional, serial FFT for the vectors that are not distributed. We shall call this the distributed FFT method. The other is to rearrange the data so that all transforms are done locally. Chu [5] examined these methods for the two-dimensional transform and found that although the latter method is asymptotically faster, their performance was similar in practice. We shall examine both for our application.

The three-dimensional Fourier transform of an array in physical space to spectral space proceeds first with a real-to-conjugate symmetric (RCS) Fourier transform of $ny \cdot nz$ vectors of length $nx$. A complex-to-complex (CC) Fourier transform is then performed on $(nx/2) \cdot nz$ vectors of length $ny$. Finally, a CC transform is performed on $(nx/2) \cdot ny$ vectors of length $nz$.

In examining the distributed FFT method, let us first consider decomposition in the $y$ and $z$ directions; only the CC transform must be performed in parallel. The minimum communication, parallel extension of the Cooley-Tukey FFT is given by Swarztrauber [19].

The graph of this algorithm is given in Fig. 2 for a vector length $n = 8$ and $P = 4$. The complex vectors $x_j$, $x_j^{(i)}$, and $\overline{x_j}$, $j = 0, ..., n-1$, are the input, $i$th stage and output vectors, respectively. The upward bracket, line, and downwards bracket indicate that the elements above the upper bracket are sent across the network and positioned under the lower bracket in the receiving processor. The circle with plus sign represents the addition of the elements from above and the replacement of the left with the sum. The circle with the minus sign and $W$ next to it represent the subtraction of the right from the left input elements. The result is multiplied by $W^j$, the $j$th root of unity. The product replaces the right input element in memory. The number of arithmetic operations is $5\,(n/P)\log_2 n$. There are $2d+1$ nearest-neighbor communications of vector length $n/(2P)$.

For $m$ transforms, each element is an $m$-tuple and the transform is performed on the last index. This prevents gather-scatter operations before communications. Local index transposition, however, must be performed on the array during the multi-dimensional transform.

An additional $\theta(d)$ communications of the complete subvectors are required to rearrange the bit-reversed array. This operation is time-consuming. In the Fourier pseudospectral method, however, this reorganization is not necessary. If the arrays are arranged sequentially in spectral space, they will be in bit-reversed order in physical space. Since the nonlinear term (in 4a) is a pointwise calculation, the particular order of the variables $\mathbf{u}$ and $\boldsymbol{\omega}$ is irrelevant, provided each array is consistently ordered. The inverse CC transform of an array in bit-reversed order proceeds similarly to the forward transform. Only nearest-neighbor exchanges of data are required.

The parallel transform in the $x$ direction needs special attention. In transforming from physical space, vectors with real elements are transformed into conjugate symmetric vectors. Cooley, Lewis, and Welch [6] have shown that the RCS transform

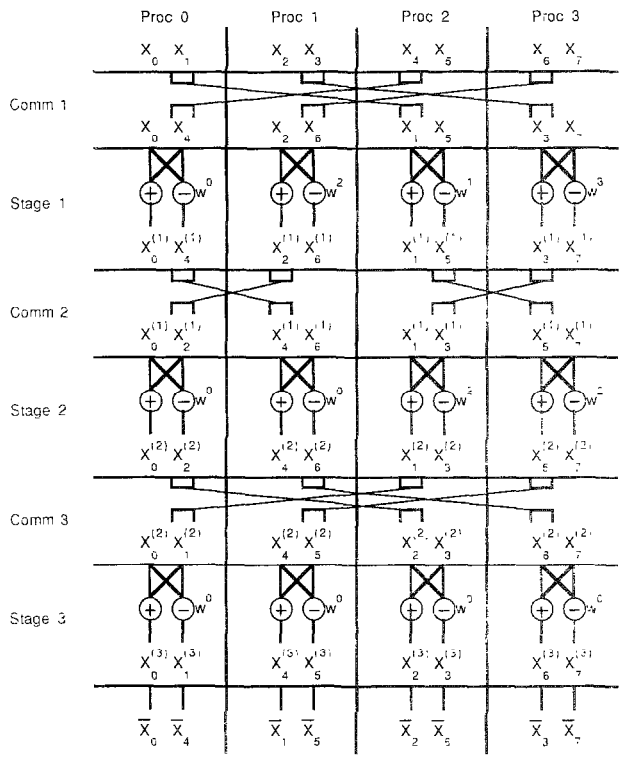


FIG. 2. A graph of the load-balanced, parallel, FFT algorithm. $P=4$, $n=8$.

can be accomplished using a CC transform of a vector of half the length. Suppose the real vector, $y_j$, $j=0, ..., 2n-1$, is to be transformed into the conjugate symmetric vector $C_k$, $k=0, ..., n$. We first create a complex vector $x_j = y_{2j} + iy_{2j-1}$, $j=0, ..., n-1$. The $n$-point CC transform is then employed to transform $x$ into $A_k$, $k=0, ..., n-1$. The following reflection operation is then used to create the conjugate symmetric vector

$$C_k = \tfrac{1}{4}[A_k + A^*_{n-k} - iW^{k}_{2n}{}^*(A_k - A^*_{n-k})] \tag{8a}$$

$$C_{n-k} = \tfrac{1}{4}[A_{n-k} + A^*_k + iW^k_{2n}(A_{n-k} - A^*_k)], \tag{8b}$$

where $k=0, 1, ..., n/2$, $W^k_{2n}$ is the $k$th root of $2n$ roots of unity and $(\ )^*$ denotes the complex conjugate. The creation of the complex vector $x_j$ is a local operation. The parallel CC transform proceeds as described above. The reflection operation (8) requires communication.

Figure 3 shows the graph of a nearest-neighbor communication algorithm that is used to rearrange a sequentially ordered vector into a form that can be used for the reflection. To accomplish this, $d$ communication stages with vector length $n/P$ or $n/P-1$ are necessary. This is approximately the same amount of communication necessary for the parallel CC transform. The communication time is a large part of the total time of the vector manipulation; hence, the RCS transform is less efficient than the parallel CC transform. The RCS transform still requires much less time, however, than the CC transform of the original real vector.

If the binary reflected grey code sequence is used to map the subvectors onto the processors, the reflection can be accomplished in one nearest-neighbor communication and a single element shift. The graph of the communication algorithm is shown in Fig. 4. The communication in the CC transform, however, is between processors that are a distance of two away from each other (next-nearest-neighbors). The amount of data times the distance traveled is essentially the same for both orderings. No significant difference in performance is seen.

The distributed FFT method for the three-dimensional transform begins with the array ordered sequentially in spectral space. A CC transform in performed in the $z$ direction. The second and third indices are then switched locally. A CC transform is performed in the $y$ direction. The first and third indices are then switched locally. Finally, a CSR transform is performed in the $x$ direction.

A second method of performing the multi-dimensional FFT on distributed arrays is to rearrange the data so that the vectors in the distributed direction become complete within the memory of the processors. Only local FFTs are necessary. The vectors in another direction become distributed so that the total memory requirements remain constant. This can be accomplished by the transpose.

Figure 5 shows a graph of the block transpose algorithm in which nearest-neighbor parallel communication can be used to transpose a matrix. In the figure,

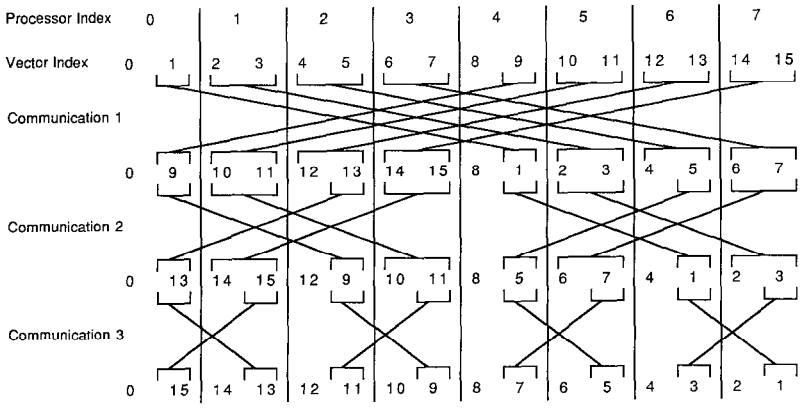


FIG. 3. A graph of the communication algorithm for the reflection operation. $P=8$, $n=16$, sequential ordering. The input and ouput vectors are in order so that Eq. (8) can be performed in an element-by-element fashion.

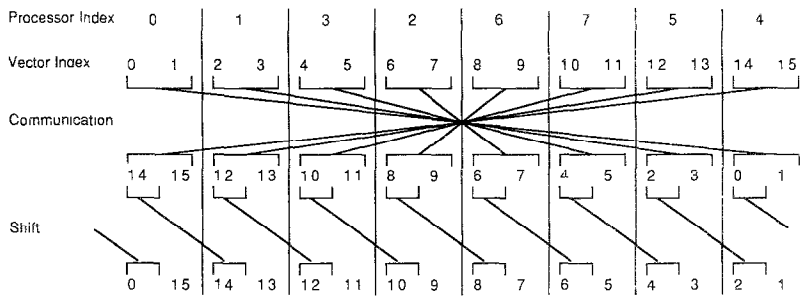


FIG. 4. A graph of the communication algorithm for the reflection operation. $P = 8$, $n = 16$, binary reflected grey code ordering. The input and output vectors are in order so that Eq. (8) can be performed in an element-by-element fashion.

a matrix which is column-distributed is rearranged to be row-distributed. The communication structure is similar to the parallel CC transform. For more information, see McBryan and van de Velde [12]. For three-dimensional problems, the transpose is performed on the last two indices to reduce the number of gather-scatter operations.

Consider decomposition in only two directions and $nx = ny = nz$. The subarray in spectral space with $P_z = 1$ is indexed as

$$\left(1 : \frac{nx/2}{P_x},\ 1 : \frac{ny}{P_y},\ 1 : nz\right). \tag{9}$$

A serial CC transform in the $z$ direction is performed locally. The second and third indices are then transposed yielding:

$$\left(1 : \frac{nx/2}{P_x},\ 1 : \frac{nz}{P_y},\ 1 : ny\right). \tag{10}$$

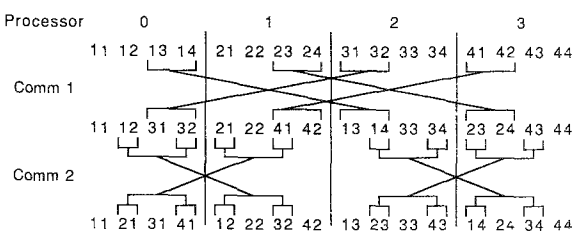


FIG. 5. A graph of the block transpose algorithm. $P = 4$, $4 \times 4$ matrix. The element $ij$ is the element in the $i$th row and $j$th column.

A serial CC transform in the $y$ direction is performed locally. The first and second indices are then switched locally giving

$$\left(1 : \frac{nz}{P_y},\ 1 : \frac{nx/2}{P_x},\ 1 : ny\right). \tag{11}$$

Finally a block transpose is performed making the $x$ direction complete, and a CSR transform on the last index is performed completing the 3D transform.

The methods for the three-dimensional FFT presented above and the steps in (4) represent the parallel Fourier pseudospectral method. In the following section we shall examine the performance on a hypercube computer.

## V. Actual Performance on the Hypercube

In this section we present benchmark results found using the NCUBE/1 computer for the parallel Fourier pseudospectral method.[1] The distributed FFT and block transpose methods are compared. Both single precision (32-bit) and double precision (64-bit) versions are tested. Timings and efficiencies are presented for many cases. A study of how performance varies with subdomain shape is also presented.

The NCUBE/1 computers employed here are a 128-node machine at the CAIP parallel processing laboratory at Rutgers and a 1024-node machine at Sandia National Laboratories in Albuquerque, NM. Each processor (or node) is a 32-bit serial processor specially designed by NCUBE. It is purported to have 300 KFLOPS computational rate, but in reality, we have found that this rate is closer to 100 KFLOPS. Each processor has $\frac{1}{2}$ Mbytes of RAM, 480 Kbytes of which can be used for program and arrays.

The memory constraints produce an upper bound on the mesh size. The algorithm calls for 10 large arrays: 3 velocity, 3 vorticity, and 3 velocity at the previous time level. The maximum number of mesh points per processor is then $2^{13}$. One processor can be used to solve a problem with $32 \times 16^2$, for example.

There is also a 26 Kbyte upper bound on the length of a message that can be passed between processors. For an array that is a power of two, an exchange can be no more than $2^{11}$ 32-bit words. Of course, longer exchanges can be accomplished in two communications, but a serious degradation in performance results.

The maximum mesh sizes that can be handled are $128 \times 64^2$ and $256 \times 128^2$ for the 128- and 1024-node machines, respectively. The latter is a typical mesh on which turbulence databases are generated for current supercomputers.

The particular choice of initial conditions is not crucial to test the parallel performance of the algorithms. The conditions need, however, to be typical, requiring all terms and stages in (4) to be calculated. We have chosen two cases that have solutions that are easily verifiable: the Taylor–Green vortex and the 3-mode Beltrami superposition commonly called the ABC flow (for Arnold, Beltrami, and Childress).

[1] New timings on the NCUBE/2 and Intel i860 can be obtained from the author.

A detailed study of the flow that evolves from the Taylor–Green vortex initial conditions is given by Brachet *et al.* [2]. The initial conditions are

$$\mathbf{u} = (\cos x \cdot \sin y \cdot \cos z, \ -\sin x \cdot \cos y \cdot \cos z, 0), \tag{12}$$

which results in certain symmetries being preserved for all $t > 0$. The resolution of the simulation can be increased by using expansions that also obey these symmetries. In our study, however, none of the spatial symmetries are enforced; all modes are calculated. The Reynolds number is taken low enough so that these symmetries remain naturally. The fact that they are preserved provides a check for the computer code. Results from runs made on the Cyber 205 and the NCUBE compare to within the round-off error for all simulations.

Beltrami flows, superpositions of circularly polarized waves, have the distinction of being the only steady Euler solution that can have chaotic streamlines [7]. The simplest of these flows, the ABC flow, is given as

$$\mathbf{u} = (B \cdot \cos y + C \cdot \sin z, \ C \cdot \cos z + A \cdot \sin x, \ A \cdot \cos x + B \cdot \sin y), \tag{13}$$

where $A$, $B$, and $C$ are constants. The Beltrami property, $\boldsymbol{\omega} = \text{constant} \cdot \mathbf{u}$, causes the nonlinear term to vanish. To make a Beltrami flow a steady solution of the Navier–Stokes equations, a body force of the form $\mathbf{f} = \nu k^2 \mathbf{u}$ is added to the equations in order to overcome viscous dissipation. The flow when $t$ is greater than 0 should be steady and have only one nonzero wavenumber mode.

Since the Reynolds numbers is low and the resolution is relatively high, the

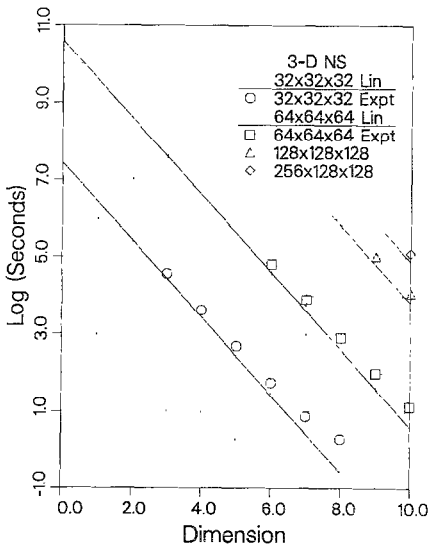


FIG. 6. The wall-clock seconds versus number of processors employed for one timestep of the Fourier pseudospectral, single-precision simulation of the three-dimensional Navier–Stokes equations. A base 2 logarithm scaling is used for both the ordinate and abscissa.

aliased modes are thought to have a negligible effect on the dynamics. They were not removed in the simulations. The neutrally stable timestepping scheme was not stabilized either. These operations, however, could have been done completely in parallel and would have enhanced the parallel performance.

Benchmark timings for simulations of one timestep of the three-dimensional Navier–Stokes equations are presented in Fig. 6. Plotted is the base 2 logarithm of the wall-clock time in seconds versus the base 2 logarithm of the number of processors (dimension). Four cases are shown: $32^3$, $64^3$, $128^3$, and $256 \times 128^2$ mesh sizes. The symbols are the experimental findings and the straight lines represent the linear performances. If $P$ processors are employed for a problem, the linear performance is the single-processor time divided by $P$.

Since none of the cases shown in Fig. 6 can actually be performed on a single processor, the time is estimated by dividing the total number of arithmetic operations by the computational rate. The total number of floating-point operations for one timestep is

$$\text{FLOPs} = \frac{nx}{2} ny \cdot nz \left[ 60 \log_2 \left( \frac{nx}{2} ny \cdot nz \right) + 313 \right]. \tag{14}$$

The first term in square brackets comes from the 12 three-dimensional FFTs, and the last term is the operations in (4). The computational rate is determined experimentally using (14) and the timings for many single-processor runs. The KFLOPs rate was approximately 110 for runs with large mesh sizes.

By dividing (14) by this rate, the linear performances in Fig. 6 are found. The

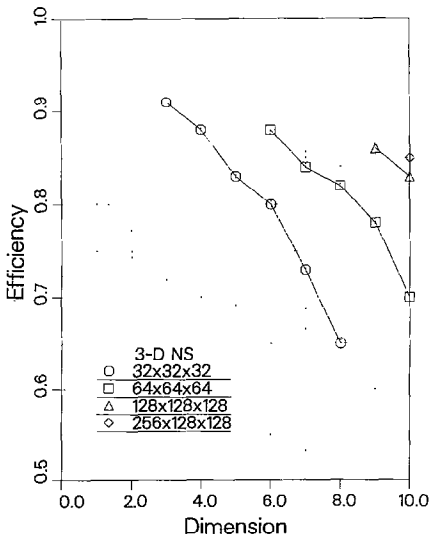


FIG. 7. The efficiency versus processor dimension for the Fourier pseudospectral, single-precision simulation of the three-dimensional Navier–Stokes equations.

experimental results appear to asymptote to the linear performance lines as the dimension decreases; thus, the single-processor estimate is close to the actual time.

One timestep for a $128^3$ mesh on 1024 processors is accomplished in less than 17 s of wall-clock time. This case is typical for direct simulations of turbulence run on supercomputers. With assembly-coded FFTs and fully vectorized codes, the CPU seconds per timestep are 16 and 23 for the CRAY XMP and 2-pipe CYBER 205, respectively. We should like to stress that the codes running on the NCUBE have been written entirely in FORTRAN 77. No attempt has been made to optimize it in any way.

Figure 7 shows the efficiency versus dimension for the cases shown in Fig. 6. Efficiency is defined as

$$\text{Efficiency} = \frac{T_1/P}{\langle T_P \rangle}, \tag{15}$$

where $T_1$ is the time to complete one timestep when only one processor is employed, and $\langle T_P \rangle$ is the average of times reported by the $P$ processors which are employed on the problem. This is a measure of how close the experimental timings are to the linear performance. The value of $T_1$ is calculated using (14). The standard deviation of the set $\{T_P\}$ from the $P$ processors varied less than 1% indicating a balanced computational load.

The efficiency decreases in an approximately linear fashion for low dimension and decreases more severely for high dimension. The latter behavior is due to the communication startup time becoming a significant part of the communication time.

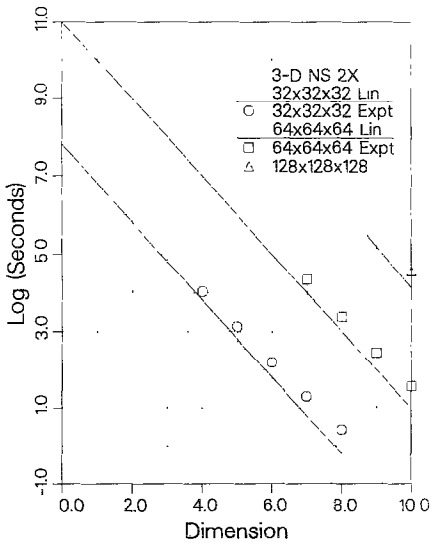


FIG. 8. The wall-clock seconds versus processor dimension for one timestep of the Fourier pseudospectral, double-precision simulation of the three-dimensional Navier–Stokes equations.

A simple model of efficiency illustrates this point. Neglecting processor idle time, the total time is

$$T_P = T_P^{\text{calc}} + T_P^{\text{comm}}. \tag{16}$$

Assuming there are no redundant calculations, $T_P^{\text{calc}}$ can be written as $T_P^{\text{calc}} = T_1/P$. The efficiency is then

$$\text{Efficiency} \approx 1 - T_P^{\text{comm}}/T_P^{\text{calc}}. \tag{17}$$

Using (14) for the calculation time and taking the time for a communication exchange to be $\alpha + \beta \cdot (\text{message length})$, Eq. (17) becomes

$$\text{Efficiency} \approx 1 - \frac{12d(\alpha(2P/nx \cdot ny \cdot nz) + \beta)}{\gamma[60 \log_2((nx/2)\, ny \cdot nz) + 313]}, \tag{18}$$

where $\gamma$ is the MFLOPS rate, $d$ is the dimension, $\alpha$ is the startup communication time in microseconds and $\beta$ is the message passing rate in microseconds per 64-bit word.

For a fixed problem size and large granularity ($\alpha/\beta \ll nx \cdot ny \cdot nz/2P$), the efficiency decreases linearly with dimension. When the granularity becomes small cnough for the startup time to be a significant part of the total communication time, then the efficiency decreases as $P \log_2 P$.

The values of $\alpha$ and $\beta$ for an exchange were experimentally determined to be 750 $\mu$s and 21 $\mu$s per 64-bit word. These values were estimated by taking the total communication times for each run and fitting them to the function $12d[\alpha + \beta(\text{number of 64-bit words in each communication})]$. For a $32^3$ mesh and $P = 128$, and for a $64^3$ mesh and $P = 1024$, the startup time takes about 36% of the total communication time. In Fig. 7 transition fom linear to a more severe decrease occurs at approximatcly these values of processor numbers.

Benchmark timings for codes written in double precision are presented in Fig. 8. Three cases are shown: $32^3$, $64^3$, and $128^3$. The rate for double precision arithmetic operations is 15% less than the rate for single precision. A word is twice as long

TABLE I

A Comparison between the Transpose and the Distributed FFT Methods

| *nx* | *ny* | *nz* | *P* | Transpose | Dist FFT |
|---|---|---|---|---|---|
| 32 | 32 | 32 | 8 | 32.1 | 23.6 |
| 32 | 32 | 32 | 16 | 16.7 | 12.2 |
| 32 | 32 | 32 | 32 | 8.67 | 6.44 |
| 64 | 64 | 64 | 64 | 39.5 | 28.2 |

*Note.* The seconds per timestep required for the cases described in columns 1–4 are in columns 5 and 6.

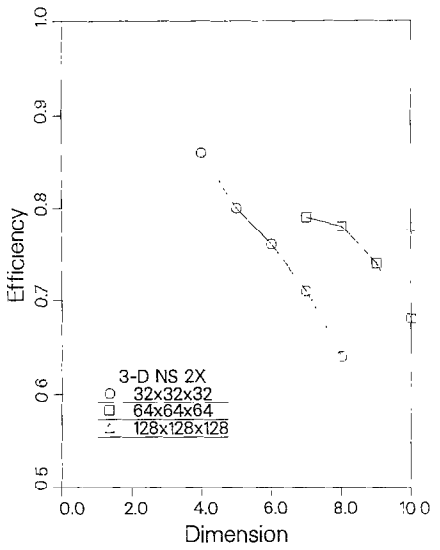


FIG. 9. The efficiency versus processor dimension for the Fourier pseudospectral, double-precision simulation of the three-dimensional Navier–Stokes equations.

in double precision; hence, the length of communicated data doubles. Compared to single-precision, a larger percentage of total time is taken by the communication; thus, a lower parallel performance results. Despite this, Fig. 8 shows that the timings are still relatively close to the linear performances and that one timestep on a $128^3$ mesh with 1024 processors took about 23 s.

Efficiency for the double precision runs is shown in Fig. 9. The efficiency for a $128^3$ mesh with 1024 processors has dropped to 78% from the single precision value of 83%. The linear behavior is again observed. The data point corresponding to $64^3$ mesh and dimension 7 is somewhat out of alignment, however.

TABLE II

Total and Communication time (Seconds per Timestep) for $128^3$ Mesh and 1024 Processors

| $128 \times 128 \times 128$ mesh, 1024 processors | | | | |
|---|---|---|---|---|
| $P_x$ | $P_y$ | $P_z$ | Seconds per timestep | Communication time |
| 1 | 32 | 32 | 16.9 | 2.69 |
| 2 | 32 | 16 | 18.1 | 2.99 |
| 4 | 16 | 16 | 18.4 | 3.30 |
| 8 | 16 | 8 | 18.7 | 3.55 |
| 16 | 8 | 8 | 19.1 | 3.86 |
| 32 | 8 | 4 | 19.6 | 4.09 |

*Note.* Timings for various subdomain are given. $P_x$, $P_y$, and $P_z$ are the number of processors employed in the $x$, $y$, and $z$ directions, respectively.

TABLE III

Total and Communication Time (Seconds per Timestep) for $64 \times 256^2$ Mesh and 1024 Processors

| $64 \times 256 \times 256$ mesh, 1024 processors | | | |
|---|---|---|---|
| $P_y$ | $P_z$ | Seconds per timestep | Communication time |
| 32 | 32 | 34.0 | 5.20 |
| 64 | 16 | 34.0 | 5.19 |
| 128 | 8 | 34.1 | 5.20 |
| 16 | 64 | 34.0 | 5.21 |
| 8 | 128 | 34.1 | 5.26 |

*Note.* Timings for subdomain shapes that vary in the $y$ and $z$ directions are given, $P_x = 1$.

Table I shows some timings of typical cases comparing the transpose method and the distributed FFT method. In each case $P_x = 1$ so that the double transpose (11) was not done. The $x$-transform was performed locally on the first index. Only the two-dimensional FFT was distributed. One timestep using the distributed FFT method takes consistently about 27% less time than one timestep using the transpose method. The number of arithmetic operations for the two methods is the same. The amount of communication for the transpose method is similar to that of the distributed FFT method.

The reason for the difference in the two methods is that in the block transpose, $d-1$ stages require a time consuming gather-scatter operation. This operation is indicated by the double bracket in Fig. 5. A better compiler or assembly-coded gather-scatter routines [9] will tend to decrease the time required for this operation. The transpose method will then be competitive with the distributed FFT method.

Table II shows the total time and communication time for one timestep of the distributed FFT method, single precision, $128^3$ mesh, and 1024 processors. Each row corresponds to a different subdomain partitioning. As a larger percentage of processors are employed in the $x$ direction, the times increase. There is a large jump between $P_x = 0$ and 1, due partly to a second local transpose that must be performed in the latter case. The increase in both total and communication times is due to the inefficiencies in the parallel RCS transform.

Table III shows results similar to those in Table II, except that only the $y$ and $z$ directions are distributed, and the mesh size is $64 \times 256^2$. The changes in partition of the $y$–$z$ plane have little effect on the amount of time required for one timestep.

## VI. Conclusions

The goal of this work was to assess the parallelization of the Fourier pseudospectral method for Navier–Stokes simulations. It was found that while the amount of

data to be communicated is large and scales with the dimension of the problem, the actual performance on problems of current interest is remarkably good. For a problem with a mesh of $128^3$ points, the efficiency on a 1024-node hypercube is about 83%, and one timestep takes 17 s of wall-clock time. The code was written entirely in FORTRAN with no optimization, assembler coding, or vectorization.

The high efficiencies are due to the fact that the only operation that requires communication is the FFT and that the parallel FFT is load-balanced and efficient for multi-dimensional transforms. The block transpose method is attractive for the multi-dimensional transform; however, more time is required than the distributed FFT method due to the inefficient gather–scatter operation. Efficiency of double precision runs is lower than that of single precision because the amount of communication is double in the former. Subdomain shape effects performance only through the degree of distribution in the $x$ direction. The parallel real-to-conjugate symmetric transform, used in the $x$ direction, is less efficient that the parallel complex-to-complex FFT that is performed in the other directions.

In practice, each processor of the NCUBE/2 is about 5 times more powerful than that of the NCUBE/1. A realistic estimate of elapsed time per timestep for a $1024^3$ mesh run on 8192 processors with 8 Mbytes of memory is about 5 min for an all FORTRAN code.

## ACKNOWLEDGMENTS

I thank R. Peskin and I. Nelken for helpful suggestions. I am grateful to R. Benner for help in accessing the hypercube at Sandia. The work was supported by the National Science Foundation under Grant EET 88-08780, by Office of Naval Research under Grant N00014-89-J 1320, and by CAIP under Grant P89-11. The tests were performed at the CAIP Parallel Processing Laboratory and at Sandia National Laboratories in Albuquerque, NM. CAIP, Center for Computer Aids for Industrial Productivity is an Advanced Technology Center of the New Jersey Commission of Science and Technology at Rutgers University.

## REFERENCES

1. S. Bonazzola and J. A. Marck, *J. Comput. Phys.* (1988).
2. M. E. Brachet, D. I. Meiron, S. A. Orszag, B. G. Nickel, R. H. Morf, and U. Frisch, *J. Fluid Mech.* **130**, 411 (1983).
3. C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang, *Spectral Methods in Fluid Dynamics* (Springer-Verlag, Berlin, 1987).
4. R. M. Chamberlain, *Parallel Comput.* **6**, 225 (1988).
5. C. Y. Chu, Cornell University Department of Computer Science Report No. TR-87-850, 1987 (unpublished).
6. J. W. Cooley, P. A. W. Lewis, and P. D. Welch, *J. Sound Vib.* **12**, 315 (1970).
7. T. Dombre, U. Frisch, J. M. Green, M. Henon, A. Mehr, and A. M. Soward, *J. Fluid Mech.* **167**, 353 (1986).
8. C. T. Gordon and W. F. Stern, *Month. Weather Rev.* **110**, 625 (1982).
9. J. L. Gustafson, G. R. Montry, and R. E. Benner, *SIAM J. Sci. Stat. Comput.* **9**, 609 (1988).
10. R. W. Hockney and C. R. Jesshope, *Parallel Computers 2* (Hilger, Bristol, 1988).

11. J. LEORAT, A. POUQUET, AND J. P. POYET, in *Problems of Collapse and Numerical Relativity 134*, edited by D. Bancel and M. Signore (Reidel, Toulouse, 1983), p. 287.
12. O. A. MCBRYAN AND E. F. VAN DE VELDE, *SIAM J. Sci. Stat. Comput.* **8** (1986).
13. D. I. MEIRON, S. A. ORSZAG, AND M. J. SHELLEY, "A Numerical Study of Vortex Reconnection," in *Mathematical Aspects of Vortex Dynamics*, edited by R. Caflisch (SIAM, Philadelphia, 1989).
14. M. V. MELANDER, "Close Interactions of 3D-Vortices in Incompressible Flows," in *Mathematical Aspects of Vortex Dynamics*, edited by R. Caflisch (SIAM, Philadelphia, 1989).
15. R. W. METCALFE, S. A. ORSZAG, M. E. BRACHET, S. MENON, AND J. J. RILEY, *J. Fluid Mech.* **184**, 207 (1987).
16. S. A. ORSZAG, *Stud. Appl. Math.* **50**, 293 (1971).
17. R. B. PELZ, "Hypercube Algorithms for Turbulence Simulation," in *11th International Conference on Numerical Methods in Fluid Mechanics*, edited by D. L. Dwoyer, M. Y. Hussaini, and R. G. Voigt, *Lecture Notes in Physics*, Vol. 323 (Springer-Verlag, Berlin, 1989), p. 462.
18. R. S. ROGALLO AND P. MOIN, *Annu. Rev. Fluid Mech.* 16, 99 (1984).
19. P. N. SWARZTRAUBER, *Parallel Comput.* **5**, 197 (1987).
20. E. TADMOR, *SIAM J. Numer. Anal.* **23**, 1 (1986).