INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 24, 1129–1158 (1997)

HYBRID SPECTRAL FINITE DIFFERENCE SIMULATIONS OF STRATIFIED TURBULENT FLOWS ON DISTRIBUTED MEMORY ARCHITECTURES

RAJAT P. GARG,[1]† JOEL H. FERZIGER[1]* AND STEPHEN G. MONISMITH[2]

[1] *Department of Mechanical Engineering, Stanford University, Stanford, CA 94305, U.S.A.*
[2] *Department of Civil Engineering, Stanford University, Stanford, CA 94305, U.S.A.*

SUMMARY

A method for efficient implementation of a combined spectral finite difference algorithm for computation of incompressible stratified turbulent flows on distributed memory computers is presented. The solution technique is the fractional step method with a semi-implicit time advancement scheme. A single-programme multiple-data abstraction is used in conjunction with a static data-partitioning scheme. The distributed FFTs required in the explicit step are based on the transpose method and the large sets of independent tridiagonal systems of equations arising in the implicit steps are solved using the pipelined Thomas algorithm. A speed-up analysis of a model problem is presented for three partitioning schemes, namely unipartition, multipartition and transpose partition. It is shown that the unipartitioning scheme is best suited for this algorithm. Performance measurements of the overall as well as individual stages of the algorithm are presented for several different grids and are discussed in the context of associated dependency and communication overheads. An unscaled speed-up efficiency of up to 91% on doubling the number of processors and up to 60% on an eightfold increase in the number of processors was obtained on the Intel Paragon and iPSC/860 Hypercube. Absolute performance of the code was evaluated by comparisons with performance on the Cray-YMP. On 128 Paragon processors, performance up to five times that of a single-processor Cray-YMP was obtained. The validation of the method and results of grid refinement studies in stably stratified turbulent channel flows are presented. © 1997 by John Wiley & Sons, Ltd.

Int. J. Numer. Meth. Fluids **24**: 1129–1158, 1997.

No. of Figs: 20. No. of Tables: 7. No. of Refs: 42.

KEY WORDS: spectral finite difference; direct numerical simulation; message-passing computers; data partitioning; fractional step methods

1. INTRODUCTION

The majority of geophysical (oceanic and atmospheric) flows are partially turbulent and density-stratified. Stratification makes stratified turbulent flows markedly different from unstratified ones. One of the most important effects of stratification is on mixing and entrainment, processes that play an important role in the exchange of energy and gases between the atmosphere and the ocean. Consequently, better understanding of this phenomenon is essential for accurate modelling of the

* Correspondence to:. J. H. Ferziger
† Present address: Sun Microsystems, Inc., Mountain View, CA 94043, U.S.A.
Contract grant sponsor: ONR.
Contract grant number: N00014-92-J-1611.

CCC 0271–2091/97/111129–30 $17.50 *Received 22 February 1996*

oceanic mixed layer. The overall goal of this work is to study the effects of stratification on turbulent channel flow via direct numerical simulation (DNS) and large-eddy simulation (LES) based on the incompressible Navier–Stokes and scalar transport equations.

DNS is an important tool in the fundamental study of turbulence; its range of applicability is closely tied to the performance of state-of-the-art supercomputers. Turbulence has been identified as a grand challenge problem that could benefit significantly from the advent of massively parallel machines, which will be key enabling agents in the endeavour to simulate ever more complex flows.

The first parallel simulations of incompressible turbulent flows were performed by Moin and Kim[1] on an SIMD architecture (ILLIAC IV computer). Since then, simulations of viscous incompressible flows have been performed on a variety of massively parallel machines. Pelz[2] implemented the Fourier pseudospectral method for simulation of incompressible isotropic turbulent flows on the NCUBE/1 Hypercube. The only major operations requiring parallelization were the multi-dimensional FFTs. He obtained efficiencies of up to 83% on a 1024-node NCUBE/1 Hypercube. A pseudospectral Navier–Stokes solver for DNS of incompressible homogeneous turbulent flows was also developed by Jackson *et al.*[3] on the iPSC/860 Hypercube. They computed the parallel multidimensional FFTs using a transpose method and obtained speeds comparable with the Cray YMP on a 32-node system. Parallel spatial DNS of a transitional boundary layer on the iPSC/860 Hypercube was reported by Joslin and Zubair;[4] performance of roughly 153 MFLOPS on a 32-node iPSC/860 was obtained. More recently, Hanebutte *et al.*[5] have implemented spatial DNS on the IBM-SP1 and have achieved several-fold improvement in overall performance over the iPSC/860 implementation. Eidson and Erlebacher[6] describe the implementation of a fully load-balanced tridiagonal solver on the Intel Touchstone Delta for DNS of compressible isotropic turbulence.

High-resolution (512^3 grid points) spectral simulations of homogeneous turbulence have been carried out by Chen and Shan[7] on the CM-2. A notable implementation of spectral methodology is the recent high-performance Fourier–Chebyshev simulations of three-dimensional thermal convection on the CM-5 performed by Cortese and Balachandar.[8] Perot[9] and Tafti[10] have reported direct simulations using finite volume and finite difference methods on the CM-5. In a recent review article, Fischer and Patera[11] present a detailed summary of the past and current work in the area of simulation of viscous incompressible flows on massively parallel computers. Additional work on implementation of computational fluid dynamics algorithms on parallel computers may be found in References 12 and 13.

In this paper we present the implementation of a hybrid spectral finite difference algorithm on the Intel iPSC/860 Hypercube and Paragon computers for simulation of incompressible stratified turbulent channel flows. The discretization is based on Fourier spectral methods in the two horizontal directions and finite different methods in the vertical direction. Spectral methods have now become a method of choice for simulation of turbulent flows in simple geometries owing to their high order of accuracy.[14] The high global accuracy of Fourier methods was the main motivation for their use in the present application.

The use of finite difference methods in the vertical direction was motivated mainly by the desire to implement other boundary conditions and to simulate flows with sharp density interfaces, such as two-layer flows which have almost discontinuous interfaces. Almost all oceanic flows and large water bodies exhibit a density profile with a steep gradient called the *thermocline*;[15] spectral methods are ill-suited to such flows as they would exhibit the well-known Gibbs phenomenon[14] around the density interface.

The scheme used in the present work requires distributed two-dimensional FFTs as well as parallel stencil-type operations, which require very different communication patterns. The solution algorithm also leads to different types of data dependencies at various stages and requires different (and mutually conflicting) implementation strategies. The choice of appropriate partitioning scheme is

thus crucial in obtaining optimal performance for all the stages of the algorithm. A performance model is developed for a problem which is representative of the solution algorithm for channel flow. The performance model is then used to evaluate the suitability of various partitioning schemes applied to this problem. The partitioning analysis presented here is general and carries over to other message-passing computers.

One of the objectives of this work was to develop a high-performance scalable code that can be used efficiently for both simulation on small problem sizes (such as the ones used in large-eddy simulations) as well as large grid sizes typical of direct simulations. Accordingly, a scalability study over a large range of problem sizes and processor configurations is presented to establish the efficiency of the implementation. We begin by presenting the numerical method in Section 2, followed by a brief description of the architecture of Intel parallel computers in Section 3. The parallel implementation is presented in Section 4 and the speed-up analysis is given in Section 5. The code validation and grid refinement results are included in Section 6. Performance results and discussions are presented in Section 7. Finally, some conclusions are drawn in Section 8.

2. NUMERICAL METHOD

The flow geometry is indicated in Figure 1; the computational domain is $(0, L_x) \times (0, L_y) \times (-h, h)$ in the x-, y- and z-directions respectively. The governing equations of motion for an incompressible Boussinesq fluid in the channel are the Navier–Stokes and scalar transport equations (in non-dimensional form)

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{1}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i u_j) = -\frac{\partial p}{\partial x_i} + \delta_{i1} + \frac{1}{Re_\tau}\frac{\partial}{\partial x_j}\left(\nu^* \frac{\partial u_i}{\partial x_j}\right) - Ri_\tau(\rho^* - \rho_{\text{lin}})\delta_{i3}, \tag{2}$$

$$\frac{\partial \rho^*}{\partial t} + \frac{\partial}{\partial x_j}(\rho * u_j) = \frac{1}{Re_\tau Pr}\frac{\partial}{\partial x_j}\left(\alpha^* \frac{\partial \rho^*}{\partial x_j}\right), \tag{3}$$

where Re_τ, Pr and Ri_τ are the Reynolds, Prandtl and bulk Richardson numbers respectively. The equations have been cast in the variable viscosity and diffusivity form (ν^* and α^* can be both position- and time-dependent) in order to simulate flows with non-uniform properties. The density field is expressed as

$$\rho = \rho_0 + \rho^*, \tag{4}$$

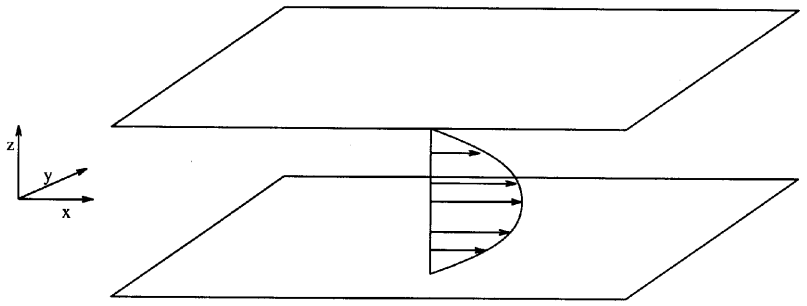

Figure 1. Schematic diagram of channel flow

where ρ_0 is the reference density; $\rho^*(x, y, z, t)$ is further decomposed as $\rho^* = \rho_b(z, t) + \rho_f(x, y, z, t)$. Here $\rho_b = \langle \rho^* \rangle$ is the instantaneous plane-averaged density, from which we can extract a time-independent linear background density ρ_{lin}, defined as $\rho_{lin} + \rho_0 = \rho_B - 0{\cdot}5\Delta\rho(1 + z/h)$, where ρ_B and ρ_T are the fixed densities at the top and bottom walls respectively and $\Delta\rho = \rho_B - \rho_T$ is the initial density difference between the two walls.

Velocities have been scaled by the initial friction velocity u_τ^0, lengths by the channel half-depth h and density perturbations by $\Delta\rho$. With these scalings the definitions of Re_τ, Pr and Ri_τ become

$$Re_\tau = u_\tau^0 h/\nu_m, \qquad Pr = \nu_m/\alpha_m, \qquad Ri_\tau = \Delta\rho g h/\rho_0 (u_\tau^0)^2. \tag{5}$$

The form of the buoyancy term in the momentum equation (2) reflects the fact that the linear portion of the background density field accounts for hydrostatic pressure variation. The flow is driven by a fixed streamwise pressure gradient, accounted for by the term δ_{i1} in the momentum equation (2). For brevity and notational convenience, from here on we shall drop the subscript asterisk on ρ_* and simply denote it as ρ.

The non-linear terms (hereafter represented as h_i) can be expressed in different forms, namely the advection, divergence, rotational or skew-symmetric form.[16] In the present simulations the skew-symmetric form

$$h_i = \frac{1}{2}\left(u_j \frac{\partial u_i}{\partial x_j} + \frac{\partial (u_i u_j)}{\partial x_j} \right), \tag{6}$$

$$h_\rho = \frac{1}{2}\left(u_j \frac{\partial \rho}{\partial x_j} + \frac{\partial (\rho u_j)}{\partial x_j} \right) \tag{7}$$

was used. Zang[17] showed that in incompressible flows this form is more accurate than the rotational form and also preserves conservation properties in the semidiscretized equations.

The spatial discretization is a mixed spectral finite differences method with Fourier expansions in the periodic directions (x and y) and second-order finite differences in the vertical direction (z). The mesh is Cartesian and non-uniform in the vertical direction and uniform in the horizontal directions. Hyperbolic tangent stretching[1] is used in the z-direction to cluster more grid points near the solid boundaries in order to resolve the fine-scale structures close to walls. The co-ordinate of the kth vertical grid point is

$$z_k = \frac{1}{a}\tanh(\xi_k \tanh^{-1} a), \quad \xi_k = -1 + \frac{2k}{N_z}, \quad k = 0, \ldots, N_z. \tag{8}$$

In the above, $N_z + 1$ is the number of grid points in the normal direction and $a = 0{\cdot}98346$ was used, following Moin and Kim.[1] A staggered grid (see Figure 2) is used in the wall-normal direction. The

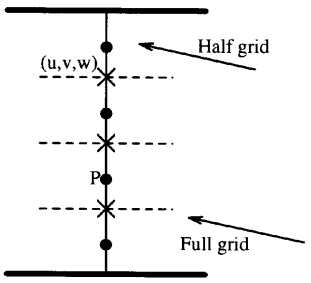

Figure 2. Staggered grid in z-direction

velocity components and scalar are defined at full-grid points while the pressure is defined at half-grid points. The continuity equation is enforced at pressure nodes while the momentum and scalar equations are enforced at velocity nodes or cell edges. Linear interpolation is used to transfer information between the half-grid and full-grid points. The series representation for the primitive variables is

$$u(x, y, z, t) = \sum_{k_x=-N_x/2}^{N_x/2-1} \sum_{k_y=-N_y/2}^{N_y/2-1} \hat{u}_{k_x,k_y}(z, t)\, \mathrm{e}^{\mathrm{i}(\hat{k}_x x+\hat{k}_y y)}, \tag{9}$$

which is the Fourier series representation; $\hat{k}_x = 2\pi k_x/L_x$ and $\hat{k}_y = 2\pi k_y/L_y$ are the wave numbers in the x- and y-directions respectively. The numbers of grid points in the horizontal directions are denoted by N_x and N_y. The pressure and scalar also have similar representations. Henceforth the subscripts k_x and k_y will not be written explicitly unless necessary; a hat will indicate a quantity in Fourier space.

The aliasing errors arising in the pseudospectral treatment of non-linear terms are removed using the $\frac{2}{3}$ rule. While the $\frac{2}{3}$ and $\frac{3}{2}$ rules are equivalent, on distributed memory computers the $\frac{2}{3}$ rule can be implemented locally with no interprocessor communication whereas the $\frac{3}{2}$ rule requires interprocessor communication when distributed FFTs are performed, which is the case in the present implementation (discussed in the next section). Thus on these architectures the $\frac{2}{3}$ rule is preferable for de-aliasing of terms treated pseudospectrally.

The time advancement scheme is of mixed type: Crank–Nicolson for the vertical viscous diffusion arising from the molecular and the horizontally averaged subgrid-scale eddy viscosity* (in large-eddy simulations) and RK3 (Runge–Kutta) for the non-linear, buoyancy (in the vertical momentum equation) and horizontal viscous terms. The pressure term is advanced in time using the backward Euler scheme since this scheme enforces the incompressibility constraint at the new time step. The method is formally second-order-accurate in time (for the velocity and scalar†), but use of the RK3 scheme allows a larger time step than that for second-order explicit schemes. An analysis of the linear advective model problem‡ results in the following condition on the time step for the RK3 scheme:

$$CFL = \max\left\{\Delta t\left[\frac{2}{3}\pi\left(\left|\frac{u}{\Delta x}\right| + \left|\frac{v}{\Delta y}\right|\right) + \left|\frac{w}{\Delta z}\right|\right]\right\}_{x,y,z} < \sqrt{3}. \tag{10}$$

In our simulations the time step was chosen such that CFL was in the range 0·8–1·25. The adequacy of the temporal resolution was ensured by requiring that the time step be several times smaller than the Komogorov time scale at this Reynolds number. This was found necessary by Choi and Moin[19] in order to perform time-accurate computations of turbulent channel flow.

The solution technique is based on the fractional step method originally developed by Chorin[20] and Temam.[21] Since then, many variants have been developed.[18,22–24] The variant used here is similar to the one described in References 22 and 23. In this method, first, intermediate velocities which do not satisfy mass conservation are computed using the momentum equations (without the pressure term) and subsequently a pressure correction is applied to compute the new velocity field that satisfies both the continuity and momentum equations. The boundary conditions for the intermediate velocities are computed using the Leveque–Oliger splitting method. Mathematically the algorithm can be broken

* In the present work the dynamic subgrid-scale model has been used; see Section 6.
† The order of accuracy of pressure is one order lower than that of velocity.[18]
‡ Assuming that the $\frac{2}{3}$ de-aliasing is applied in the horizontal directions.

into three stages: the first stage is the calculation of the terms that will be treated explicitly, i.e. the explicit right-hand side,

$$\mathbf{r}_{\mathbf{v}}^{l} = \gamma_l \Delta t \mathbf{N}_l + \left[1 + \frac{\alpha_l \Delta t}{2}\left(\frac{1}{Re} + \langle \nu_{\mathrm{v}} \rangle\right) L_{\mathrm{v}}\right] \mathbf{v}^{l-1}, \tag{11}$$

the second stage is the velocity step (solution of the Helmholtz equations for the intermediate velocity),

$$\left[1 - \frac{\alpha_l \Delta t}{2}\left(\frac{1}{Re_\tau} + \langle \nu_{\mathrm{v}} \rangle\right) L_{\mathrm{v}}\right] \tilde{\mathbf{v}}^{l} = \mathbf{r}_{\mathbf{v}}^{l}, \tag{12}$$

and the third stage is the solution of the pressure Poisson equation followed by pressure correction,

$$D^{\mathrm{T}} G(\phi^l) = \frac{1}{\alpha_l \Delta t} D^{\mathrm{T}}(\tilde{\mathbf{v}}^l), \tag{13}$$

$$\mathbf{v}^l - \tilde{\mathbf{v}}^l = -\alpha_l \Delta t G(\phi^l), \quad l = 1, 2, 3. \tag{14}$$

The RK3 substeps are denoted by l. At the end of the third RK3 substep the solution at the new time level $n+1$ is $\mathbf{v}^{n+1} = \mathbf{v}^l$. For the first substep we have that $\mathbf{v}^{l-1} = \mathbf{v}^n$. In the above, G, D^{T} and L_{v} are the discrete gradient, divergence and vertical second-derivative operators respectively. ν_{v} is the variable portion of the viscosity and $\langle\ \rangle$ represents averaging over the horizontal planes. ϕ is the pseudo-pressure related to the actual pressure by

$$G(p^l) = G(\phi^l) + \frac{\alpha_l \Delta t}{2}\left(\frac{1}{Re_\tau} + \langle \nu_{\mathrm{v}} \rangle\right) L_{\mathrm{v}} G(\phi^l). \tag{15}$$

The boundary conditions have already been incorporated in the discrete matrix operators.

A low-storage RK3 scheme with the following parameters (for the three stages) has been chosen in the present work:[25]

$$\begin{aligned} &\mathbf{N}_1 = \mathbf{H}_1, && \alpha_1 = \tfrac{1}{3}, && \gamma_1 = \tfrac{1}{3}, \\ &\mathbf{N}_2 = (\mathbf{H}_2 - \tfrac{5}{9}\mathbf{N}_1), && \alpha_2 = \tfrac{5}{12}, && \gamma_2 = \tfrac{15}{16}, . \\ &\mathbf{N}_3 = (\mathbf{H}_3 - \tfrac{153}{128}\mathbf{N}_2), && \alpha_3 = \tfrac{1}{4}, && \gamma_3 = \tfrac{8}{15} \end{aligned} \tag{16}$$

Here $\mathbf{H}$ represents the non-linear, buoyancy and horizontal viscous terms.

Performing the Fourier transform of equations (12)–(14) and using the orthogonality properties of the Fourier polynomials, we obtain a set of one-dimensional equations for the Fourier coefficients (for each wave number pair k_x, k_y). Performing a second-order finite difference discretization of these equations results in the tridiagonal system of linear equations

$$-\alpha_l \Delta t \langle \nu \rangle_k (a_2)_k (\hat{\tilde{\mathbf{v}}})_{k-1} + [1 - \alpha_l \Delta t \langle \nu \rangle_k (b_2)_k](\hat{\tilde{\mathbf{v}}})_k - \alpha_l \Delta t \langle \nu \rangle_k (c_2)_k (\hat{\tilde{\mathbf{v}}})_{k+1} = (\hat{\mathbf{r}}_{\mathbf{v}})_k, \tag{17}$$

which can be inverted efficiently using Gaussian elimination. In the above, $\langle \nu \rangle = 1/Re_\tau + \langle \nu_{\mathrm{v}} \rangle$ is evaluated at time level $l-1$; $(a_2)_k$, $(b_2)_k$ and $(c_2)_k$ are the coefficients (at point z_k) arising from the second-order discretization of L_{v}. The superscript indicating the time level on $(\hat{\tilde{\mathbf{v}}}_{k_x,k_y})_k$ has been dropped.

The Poisson equation for pressure (equation (13)) is derived using the continuity equation. In discrete form the continuity equation in Fourier space is

$$\mathrm{i}\hat{k}_x \hat{u}_{k-1/2} + \mathrm{i}\hat{k}_y \hat{v}_{k-1/2} + \frac{\hat{w}_k - \hat{w}_{k-1}}{\Delta z_k} = 0, \quad k = 1, \ldots, N_z, \tag{18}$$

where $\hat{u}_{k-1/2}$ and $\hat{v}_{k-1/2}$ are related to the values at full-grid points* and $\Delta z_k = z_k - z_{k-1}$. Substitution of the (x, y, z) components of equation (14) in the above equation leads to equation (13). In Fourier space the discrete operators in this equation are as follows (the superscript l indicating the RK3 substep has been dropped):

$$D^{\mathrm{T}}\widehat{G(\phi)} = (d_1)_k \hat{\phi}_{k-3/2} + (d_2)_k \hat{\phi}_{k-1/2} + (d_3)_k \hat{\phi}_{k+1/2}, \tag{19}$$

$$\frac{1}{\alpha_l \Delta t} \widehat{D^{\mathrm{T}}(\tilde{\mathbf{v}})} = \mathrm{i}\hat{k}_x [I_k^{\mathrm{f2h}} \hat{\tilde{u}}_k + (1 - I_k^{\mathrm{f2h}}) \hat{\tilde{u}}_{k-1}] + \mathrm{i}\hat{k}_y [I_k^{\mathrm{f2h}} \hat{\tilde{v}}_k + (1 - I_k^{\mathrm{f2h}}) \hat{\tilde{v}}_{k-1}] + \frac{\hat{\tilde{w}}_k - \hat{\tilde{w}}_{k-1}}{\Delta z_k}. \tag{20}$$

The coefficients are as follows (for $k = 2, \ldots, N_z - 1$) :

$$(d_1)_k = \frac{1}{\Delta z_k \Delta z_{k-1/2}} - (\hat{k}_x^2 + \hat{k}_y^2)(1 - I_k^{\mathrm{f2h}})(1 - I_{k-1}^{\mathrm{h2f}}), \tag{21}$$

$$(d_2)_k = -\frac{1}{\Delta z_k} \left(\frac{1}{\Delta z_{k-1/2}} + \frac{1}{\Delta z_{k+1/2}} \right) - (\hat{k}_x^2 + \hat{k}_y^2)[I_{k-1}^{\mathrm{h2f}}(1 - I_k^{\mathrm{f2h}}) + I_k^{\mathrm{f2h}}(1 - I_k^{\mathrm{h2f}})], \tag{22}$$

$$(d_3)_k = \frac{1}{\Delta z_k \Delta z_{k+1/2}} - (\hat{k}_x^2 + \hat{k}_y^2) I_k^{\mathrm{f2h}} I_k^{\mathrm{h2f}}. \tag{23}$$

At $k = 1$ (lower boundary),

$$(d_1)_k = 0, \tag{24}$$

$$(d_2)_k = -\frac{1}{\Delta z_k \Delta z_{k+1/2}} - (\hat{k}_x^2 + \hat{k}_y^2) I_k^{\mathrm{f2h}} (1 - I_k^{\mathrm{h2f}}), \tag{25}$$

$$(d_3)_k = \frac{1}{\Delta z_k \Delta z_{k+1/2}} - (\hat{k}_x^2 + \hat{k}_y^2) I_k^{\mathrm{f2h}} I_k^{\mathrm{h2f}}, \tag{26}$$

$$\hat{\tilde{w}}_{k-1} = \hat{w}^{n+1}|_{z=-h}; \tag{27}$$

and at $k = N_z$ (upper boundary),

$$(d_1)_k = \frac{1}{\Delta z_k \Delta z_{k-1/2}} - (\hat{k}_x^2 + \hat{k}_y^2)(1 - I_k^{\mathrm{f2h}})(1 - I_{k-1}^{\mathrm{h2f}}), \tag{28}$$

$$(d_2)_k = -\frac{1}{\Delta z_k \Delta z_{k-1/2}} - (\hat{k}_x^2 + \hat{k}_y^2) I_{k-1}^{\mathrm{h2f}} (1 - I_k^{\mathrm{f2h}}), \tag{29}$$

$$(d_3)_k = 0, \tag{30}$$

$$\hat{\tilde{w}}_k = \hat{w}^{n+1}|_{z=h}. \tag{31}$$

In the above equations, $\Delta z_{k-1/2} = z_{k+1/2} - z_{k-1/2}$ and the interpolation operators are defined as

$$I_k^{\mathrm{f2h}} = \frac{z_{k-1/2} - z_{k-1}}{z_k - z_{k-1}}, \quad k = 1, \ldots, N_z, \tag{32}$$

$$I_k^{\mathrm{h2f}} = \frac{z_k - z_{k-1/2}}{z_{k+1/2} - z_{k-1/2}}, \quad k = 1, \ldots, N_z - 1; \tag{33}$$

and at boundaries,

$$I_k^{\mathrm{h2f}} = \frac{z_k - z_{k+1/2}}{z_{k+3/2} - z_{k+1/2}}, \quad k = 0, \qquad I_k^{\mathrm{h2f}} = \frac{z_k - z_{k-3/2}}{z_{k-1/2} - z_{k-3/2}}, \quad k = N_z. \tag{34}$$

* Recall that linear interpolation is used to transfer information from full-grid points to half-grid points and *vice versa*.

Note that since the pressure is calculated at the interior points only, no boundary conditions need to be specified for it. The right-hand side of the pressure equation, however, contains the desired boundary conditions on the intermediate velocity and normal velocity (at the end of the full step, (14)). The equations for the pressure are singular for $k_x = k_y = 0$, reflecting the fact that the pressure is determined only to within an arbitrary constant, specified by enforcing the mean value of the pressure next to the bottom wall (i.e. by arbitrarily setting $\hat{\phi}(z_{1/2}, t) = 0$ for $k_x = k_y = 0$).

The scalar equation is also solved in two stages:

$$r_\rho^l = \gamma_l \Delta t N_\rho^l + \left[1 + \frac{\alpha_l \Delta t}{2}\left(\frac{1}{Re_\tau Pr} + \langle \alpha_v^\rho \rangle\right) L_v\right] \rho^{l-1}, \tag{35}$$

$$\left[1 - \frac{\alpha_l \Delta t}{2}\left(\frac{1}{Re_\tau Pr} + \langle \alpha_v^\rho \rangle\right) L_v\right] \rho^l = r_\rho^l, \quad l = 1, 2, 3. \tag{36}$$

Here $\langle \alpha_v^\rho \rangle$ represents the plane-averaged variable scalar diffusivity and N_ρ^l represents the RK3 temporal discretization of the non-linear and horizontal diffusion terms in the ρ equation. After performing Fourier transform and finite differencing, equation (36) also reduces to a tridiagonal system of equations for each horizontal wave number pair. These are inverted in a similar fashion to equations (18). Both Neumann and Dirichlet boundary conditions (at $z = \pm h$) can be implemented in a straightforward manner.

3. ARCHITECTURE OF iPSC/860 AND PARAGON

The Intel parallel computers are based on a distributed memory, message-passing MIMD architecture. The iPSC/860 at Stanford has 32 nodes interconnected by a five-dimensional Hypercube circuit-switched network with e-cube routing. The communication is bidirectional, i.e. two nodes can simultaneously exchange messages in both directions. The peak bandwidth of the network is 2·8 MB s^{-1} and the latency is 67 μs for short messages (less than 100 bytes) and 177 μs for large messages.[26] Each node is based on the 64 bit, 40 MHz i860 microprocessor with 16 MB of memory (RAM). The i860 processor has an 8 kB instruction and an 8 kB two-way set-associative data cache. Additionally, it has two pipelined floating point units, an adder and a multiplier. The floating point units each can produce one result per clock cycle in 32 bit arithmetic for a peak performance of 80 MFLOPS. For double-precision arithmetic the adder needs one cycle while the multiplier needs two cycles to produce one result for a peak performance of 60 MFLOPS.

The Intel Paragon at San Diego Supercomputer Center (SDSC) has 400 nodes interconnected by a 2D worm-hole routed mesh network. Each node is a general-purpose (GP) node which has dual 50 MHz i860XP microprocessors—one as an application processor and the other as message coprocessor (MCP). The architecture of the i860XP is very similar to that of the i860 chip except for a higher clock rate and a larger data cache (16 kB for the i860XP). The peak floating point rates for the i860XP microprocessor are 100 MFLOPS in single precision and 75 MFLOPS in double precision. The network has (for OSF 1.2) a peak transfer rate of 90 MB s^{-1} and latencies of the order of 50 μs with the MCP enabled (the MCP can be used whenever computation and communication can be overlapped; typically this can be achieved by using asynchronous message-passing calls).

The most commonly used programming model for Intel parallel computers is message passing, whereby the user explicitly performs data partitioning and inserts communication calls to exchange data between the processors. The most commonly used languages are C and Fortran77, with message passing done using either NX or MPI libraries.

4. PARALLEL IMPLEMENTATION

The fractional step algorithm naturally divides into three stages (equations (11)–(14)). By analysing each stage, we can identify the communication and data dependence requirements of the algorithm and develop a parallel implementation that optimizes these requirements. The first stage of the algorithm involves computation of the partial derivatives (in the three directions) of the advective and diffusive terms using known velocity and density data. This is a 'parallel by point' type of calculation. In these calculations the computation at a grid point requires only values at the previous time step at one or more grid points. Thus the only overhead is communication of the non-local data, which can be performed any time before the computation. In the two horizontal directions, derivatives are computed using Fourier spectral methods. These require global long-distance communication to perform the FFTs. On the other hand, in the vertical direction, stencil-type operations are performed which require nearest-neighbour communication. The cost of performing the communication in the FFTs is much higher than that of nearest-neighbour communication and the primary parallelization requirement of this stage is efficient implementation of the distributed FFTs.

In the second stage a tridiagonal matrix needs to be inverted for each wave number pair. The computations in this stage are 'parallel by line'.* The parallelization requirements of this stage are the solution of a large number of independent systems of distributed tridiagonal equations. The matrix is diagonally dominant (so no pivoting is required in Gaussian elimination) and also identical for each wave number pair (it only depends on the vertical grid spacing and $\langle v_v(z, t)\rangle$). Thus it can be factored once and then used to solve for each right-hand side successively at each time step. This, along with the fact that the matrix is asymmetric, makes the use of iterative methods such as conjugate gradient algorithms unattractive and direct inversion method is preferable. The requirements of the third stage (equation (14)) are similar to those of the velocity step and finally the pressure correction step (equation (14)) is an almost 'data-parallel' computation.†

The above considerations guided the implementation strategy that was used in the code. This is described next.

For parallel implementation the programming model is based on a node-only single-programme multiple-data (SPMD) abstraction.[27] This node-only model was chosen over the host node model as it is scalable to large problem sizes and a large number of processors. A static 2D unipartitioning scheme[28] is used for data decomposition, i.e. the three-dimensional computational domain is divided into P (total number of processors) non-overlapping subdomains (with equal numbers of points), each of which resides in a separate processor (see Figure 3). The number P is factored into $P_x \times P_z$, the numbers of processors used in the partitioning in the x (streamwise) and z (vertical) directions

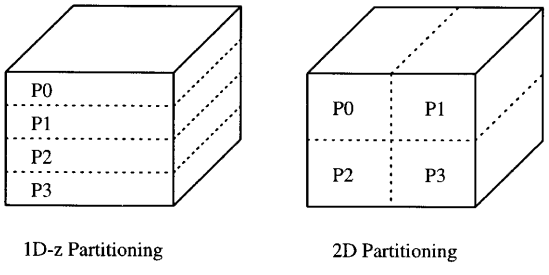

Figure 3. 1D and 2D unipartition schemes

* Each grid point is coupled to the other grid points in a fixed direction.
† The only communication required is in evaluating the z-derivative of ϕ in the correction of vertical velocity.

respectively. The processors exchange information at domain boundaries via message passing. 1D data partitioning (either the vertical or the streamwise direction distributed) can be implemented simply by setting the partition size in one of the directions to unity. All these schemes map efficiently on both the Hypercube topology of the iPSC/860 and the mesh topology of the Paragon XP/S. The motivations for choosing the unipartitioning scheme are discussed in the next section. The code is written in F77; the NX library is used for message passing. Conversion of NX calls to MPI calls is straightforward.

Parallel 2D FFTs are required in the explicit stage of the algorithm for the 2D unipartitioning scheme; we use the transpose method. In this method the FFTs are performed on memory-resident data; data in the distributed direction are brought into the local memory of a processor by performing a distributed matrix transpose (we use the multiphase complete exchange algorithm[26]). An advantage of this method is that it allows the use of efficient single-processor FFT library routines and separates the computation and communication phases.

Based on the timing model for the complete exchange algorithm presented in Reference 26, we can derive a simple speed-up model for the 2D parallel FFTs:

$$S_{\mathrm{p}} = \left(\frac{1}{P} + \frac{2(P-1)[\lambda + N^2 P^{-2} b\beta + \delta P(\log_2 P)/2(P-1)]}{6N^2(\log_2 N)t_{\mathrm{c}}} \right)^{-1}. \tag{37}$$

Here λ is the message-passing latency, b is the number of bytes per word of data, β is the message-passing bandwidth in words per second, δ is the distance impact in seconds per dimension on the Hypercube[26] and t_{c} is the computation cost per floating point operation. A comparison of this model with measured timings on the iPSC/860 for 512∗512 FFT is shown in Figure 4. The predictions of the model are in good agreement with the measured timings; we see that low speed-up efficiencies (less than 50%) are obtained with this algorithm when compared with the ideal (linear) speed-up curve. The complete exchange algorithm requires an all-to-all communication between the processors, which is the densest communication exchange between a group of processors. Sivasubramaniam *et al.*[29] show that on a Hypercube architecture, link bandwidths higher than 6 MB s^{-1} are required on $P > 4$ processors to keep the network overheads below 30% for distributed FFTs. On the iPSC/860, which has a peak bandwidth of 2·8 MB s^{-1}, communication overheads result in 50% or lower efficiencies. Clearly, it is preferable to perform in-place FFTs as would be the case in 1D vertical decomposition. As will be shown later, 2D partitioning is used in order to increase the parallelism in the vertical direction operations. Note that on the Paragon (under OSF 1·2) the bandwidths are in the range 35–40 MB s^{-1} and better scalability for FFTs is achieved with this algorithm.[29]

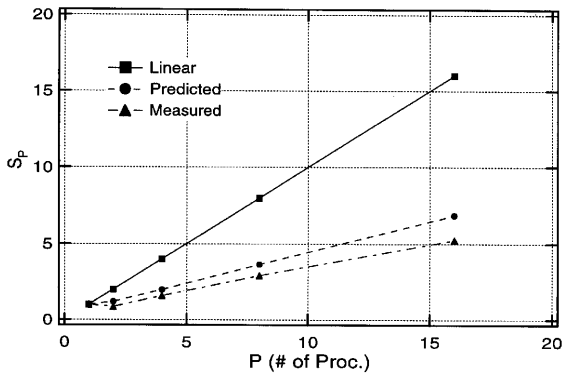

Figure 4. Performance of 2D parallel FFT on iPSC/860

The tridiagonal systems of equations arising in both the second and third steps are solved using the pipelined Thomas algorithm. A high degree of parallelism can be obtained from the Thomas algorithm when several independent solves are performed. This is achieved by pipelining the forward elimination (and back substitution) across the processors so that several solves are overlapped. The message start-up and pipelining overhead delay trade-offs determine the optimum number of line solves to be performed before a message is transmitted to the neighbouring processor.

A simple timing model can be derived for this algorithm which can be used to obtain an expression for l_{opt}, the optimum number of line solves per message:[28]

$$T_{\text{ovr}} = 8P_z\left\lceil \frac{N_z}{P_z} \right\rceil l_{\text{p}} T_{\text{c}} + P_z(\lambda + 4l_{\text{p}} b\beta), \tag{38}$$

$$T_{\text{pip}} = \frac{N_x N_y}{P_x l_{\text{p}}}\left\lceil \frac{N_z}{P_z} \right\rceil 8l_{\text{p}} t_{\text{c}} + \frac{N_x N_y}{P_x l_{\text{p}}}(\lambda + 4l_{\text{p}} b\beta), \tag{39}$$

$$T_{\text{tot}} = T_{\text{ovr}} + T_{\text{pip}} \tag{40}$$

$$\Rightarrow l_{\text{opt}} = \sqrt{\left(\frac{N_x N_y \lambda}{P_x(8N_z t_{\text{c}} + 4b\beta P_z)}\right)}. \tag{41}$$

Here T_{ovr}, T_{pip} and l_{p} denote the cost in the overhead stage, the cost in the pipelined stage and the packing factor (i.e. the number of line solves performed before a message is sent) respectively. In this model we assume five floating point operations in the forward elimination and three in the back substitution stage of the Gaussian elimination algorithm. Similarly, three words of data per solve are transmitted in the forward elimination and one in the back substitution stage. Note that in this model the local memory access times for loads and stores of various variables have been ignored.

The effect of the packing factor on the measured execution time for solving a system of tridiagonal equations is shown in Figure 5 for a representative case on the iPSC/860 ($N_x \times N_y$ is the number of independent systems to be solved, each of size $N_z \times N_z$). We see that the execution time drops rapidly as l_{p} is increased up to 4; further increase decreases it slowly (for very high values of l_{p} the time will start to increase), with the optimal value being $l_{\text{p}} = 16$ or 32. These are the values used in subsequent timing measurements. This type of behaviour was also observed for other values of N_x, N_y and N_z, with optimal values increasing with increasing grid. The value of l_{opt} predicted by the model (for the case shown in Figure 5) is 30, which is quite close to the value determined on the basis of measurements. For computational convenience and to ensure an integer number of line solves, values

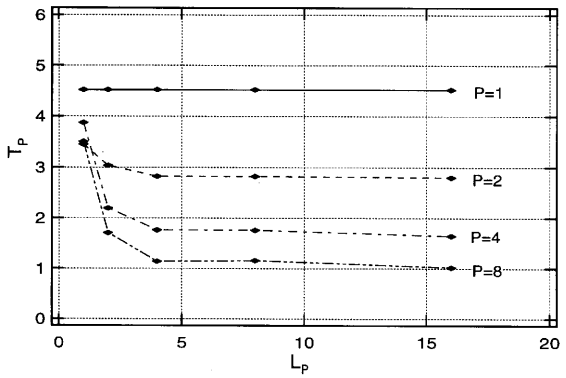

Figure 5. Time (in seconds) versus l_p (packing factor) for pipelined tridiagonal Thomas algorithm on iPSC/860; $N_x = N_y = N_z = 64$. P is the number of processors

of l_p which are powers of 2 were used in actual simulations and timing measurements. In Figure 5 we also see that an efficiency of roughly 81% is obtained when the number of processors is doubled (for $l_p = 4$); this drops to 50% for eight times as many processors. The timing model predicts somewhat higher efficiencies.

5. SPEED-UP ANALYSIS OF MODEL PROBLEM

In order to evaluate the optimal partitioning scheme for this problem, a speed-up analysis of a simpler model problem, namely the scalar transport equation (equation (3)), was performed. For a given velocity field this equation is solved in two stages (equations (35) and (36)). The computational characteristics of these stages are identical with those of the first two stages (equations (11) and (12)) of the fractional step algorithm. In the present spatial discretization method the solution of the pressure Poisson equation (stage 3, equation (14)) is the least expensive step for serial (and parallel) implementation and the total computational cost is dominated by the first two stages of the fractional step algorithm. This stage requires inverting a tridiagonal system of equations for each wave number pair to obtain the Fourier coefficients of pressure. Once the Fourier coefficients have been obtained, they can be used to enforce discrete incompressibility directly in the wave number space (equation (14)). Thus the pressure need not be computed in the physical space except for purposes of computing statistical quantities involving the pressure (in turbulent flow simulations). Hence the performance analysis of the scalar transport equation is representative of and relevant to the choice of partitioning scheme for the fractional step algorithm.

We analysed three schemes, namely unipartitioning, multipartitioning and transpose partitioning (see Figure 6), applied to the model problem. The distributed FFTs required in the unipartition and multipartition schemes are assumed to be performed using the transpose method. The distributed tridiagonal system of equations arising in the unipartition scheme (in the implicit stage) is assumed to be solved using the pipelined Thomas algorithm. Note that for the multipartition scheme the Thomas algorithm can be applied in a straightforward fashion without any load imbalance. In the transpose partition both stages are performed in-place and a global data transpose is used between stages to bring all the data required for a given stage to the local memory of the processor.

The speed-ups and speed-up efficiencies for the different schemes on a Hypercube architecture* can be expressed as

$$S_p = 1/(1/P + O_s), \tag{42}$$

$$S_\eta = S_p/P, \tag{43}$$

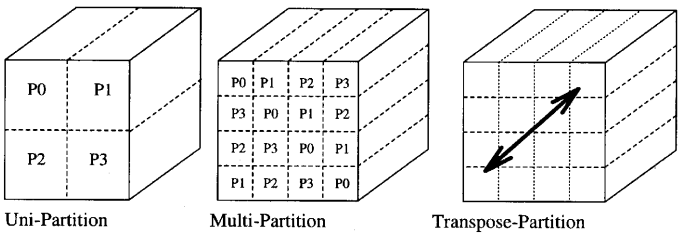

Figure 6. Unipartition, multipartition and transpose partition schemes

* A Hypercube architecture is assumed to perform the complexity analysis of the global exchange algorithm for the distributed matrix transpose. Analyses for other network topologies such as the mesh network can be done in a similar fashion.

where $O_s = Num_s/Den$ represents the parallelization overhead for each data-mapping scheme. The expressions for Num_s and Den are given below

$$Num_{\text{uni}} = 8(P_x - 1)\left\lceil \frac{N_z}{P_z} \right\rceil \left(\lambda + \frac{N_x N_y}{P_x^2} b\beta + \delta \frac{(\log_2 P_x)P_x}{2(P_x - 1)} \right) + \left(\frac{N_x N_y}{P_x l_p} + P_z + 4 \right) \lambda + \left(\frac{8N_x N_y}{P_x} + 4P_z l_p \right) b\beta + 8N_z l_p t_c, \tag{44}$$

$$Num_{\text{mul}} = 8(P - 1)N_z \left(\lambda + \frac{N_x N_y}{P^2} b\beta + \delta \frac{(\log_2 P)P}{2(P - 1)} \right) + 3(P - 1)\lambda + 6(P - 1)\frac{N_x N_y}{P} b\beta, \tag{45}$$

$$Num_{\text{trp}} = 2(P - 1)\left(\lambda + \frac{N_x N_y N_z}{P^2} b\beta + \delta \frac{(\log_2 P)P}{2(P - 1)} \right) + 4(\lambda + N_x N_y b\beta), \tag{46}$$

$$Den = \frac{N_x N_y N_z}{P} [40 + 6(\log_2 N_x + \log_2 N_y)] t_c. \tag{47}$$

Here N_x, N_y and N_z are the numbers of grid points in the x-, y- and z-direction respectively and the other parameters are as before (see Section 2). Note that, as was the case in the analysis of the pipelined Thomas algorithm, memory access times have not been included. The subscripts 'uni', 'mul' and 'trp' denote the unipartitioning, multipartitioning and transpose partitioning respectively.

The S_η predictions for the model problem using the machine parameter values for the iPSC/860* are shown in Figure 7. The smaller grid is the typical grid used in large-eddy simulations and the larger grid is typical of the ones in direct simulations. The results clearly indicate that multipartitioning leads to very low efficiencies compared with the unipartition and transpose schemes. The reason for this is that while multipartitioning eliminates the data dependence in the solution of tridiagonal systems of equations, it significantly increases the amount and stages of

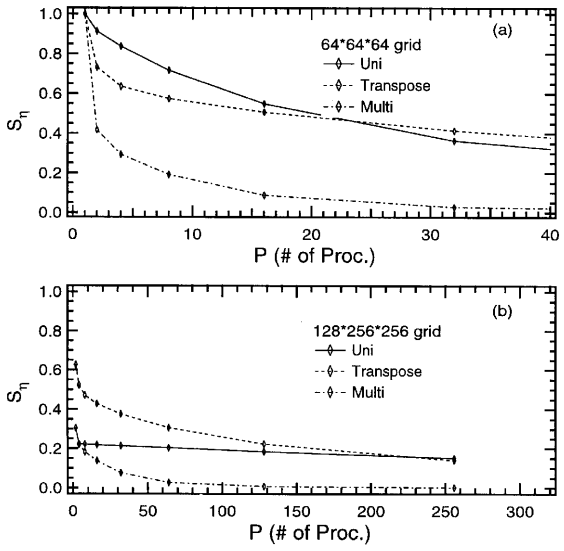

Figure 7. S_η predictions for model problem on iPSC/860 for coarse and fine grids

* We have used the values presented in Reference 26.

communication required to perform the FFTs. Furthermore, it serializes the inherent parallelism in the FFTs that is exposed by the 1D vertical unipartition and transpose schemes (groups of horizontal planes reside in each processor and in-place FFTs can be performed in each such processor independently).

For the two grids considered in Figure 7, the memory requirements of the Navier–Stokes solver are such that a minimum of four processors are required for the smaller grid and a minimum of 64 processors are required for the larger grid (for 16 MB/node computers). Thus we compare the two schemes for numbers greater than this minimum. For the smaller grid the unipartitioning shows a higher efficiency because a 1D-z partition is used and the FFT's are performed locally, with communication and data dependence delays occurring only in the vertical operations. The transpose partitioning, on the other hand, requires global communication in the transpose phase.

In the case of the larger grid a 2D unipartitioning* has to be used (see Section 6.2) in order to improve the efficiency of the tridiagonal solver. Thus, for unipartitioning, distributed FFTs are required in stage 1, leading to increased communication overhead. However, in the unipartition scheme the matrix transpose (for FFTs) is performed only across P_x processors whereas in the transpose scheme it is performed across P processors. Thus for the unipartition scheme the cost of FFTs is fixed with increasing P (P_x fixed) and the nearly linear decrease in S_η is occurring owing to increased overhead in the tridiagonal solver. For the transpose partitioning the total number of stages ($P-1$ on a Hypercube), as well as the data to be communicated, is higher in the global exchange algorithm than for the unipartition scheme, resulting in poorer scalability for P greater than 64 (Figure 7). Furthermore, on a 2D mesh (i.e. on the Paragon) the global exchange would suffer greater contention and require more stages than on a Hypercube.[30]

Thus, based on these predictions, over the range of problem and processor sizes that are of interest to us for stratified turbulent channel flow simulations, we conclude that the unipartitioning scheme should be superior to the transpose partition scheme and the former was therefore used in the parallel implementation.

6. CODE VALIDATION AND GRID REFINEMENT STUDIES

The accuracy of the code was verified by computing the instantaneous growth rates and phase speeds of small-amplitude Tollmien–Schlichting (TS) waves in an unstratified flow and comparing them with the theoretical values. The results are presented in Table I. The 2D cases were run on a $4 \times 4 \times 256$ ($x \times y \times z$) and the 3D cases on a $8 \times 8 \times 128$ grid. The box size was $2\pi h \times 2\pi h \times 2h$, where h is the channel half-depth. The growth rates in Table I are within 5% of the theoretical values. The error in the perturbation energy (after one time period) as a function of the number of grid points

Table I. Comparison of computed and theoretical growth rates and frequencies of TS waves in unstratified channel flow

	Wave number		Theoretical		Computed	
Re_{cl}	k_x	k_y	Growth rate	Frequency	Growth rate	Frequency
7500	1	0	0·002235	0·24989	0·002144	0·25004
7500	1	1	−0·02178	0·29097	−0·02229	0·30777
5000	1	0	−0·0017503	0·26813	−0·0017542	0·26838
5000	1	1	−0·019902	0·32264	−0·019399	0·32538

* $P_x = 4$ gave the best results.

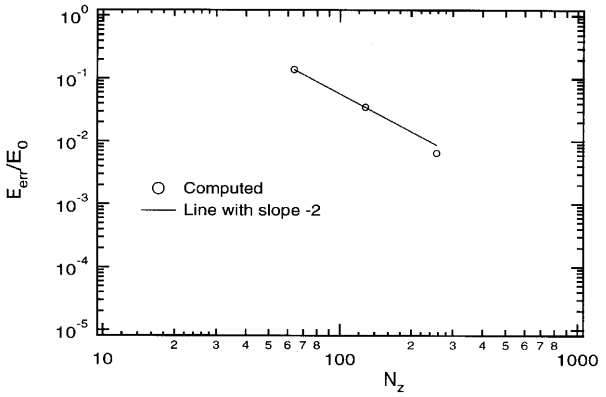

Figure 8. Accuracy of code in computing growth of (1, 0) TS mode in channel flow; $Re_{cl} = 7500$, $T = 25{\cdot}13$

in the vertical for the 2D (1, 0) Tollmien–Schlichting mode is shown in Figure 8. As expected, the error is inversely proportional to the square of the number of grid points in the vertical (for comparison a line of slope -2 is plotted on the same graph). In the x-direction four points were used; these are sufficient to exactly resolve this mode using Fourier functions.

We also performed large-eddy simulations of a pressure-driven (fixed pressure gradient) turbulent channel flow with passive scalar (temperature) at Reynolds number 180 (based on friction velocity and channel half-depth) and Prandtl number 0·71 (heated air). Two different grids were used: (i) coarse grid $64 \times 64 \times 65$ and 32 processors; (ii) fine grid $128 \times 128 \times 97$ and 64 processors.* The simulations were conducted on the Intel Paragon at San Diego Supercomputer Center. In the present simulations the dynamic subgrid-scale model[31] with Lilly's least squares modification[32] is used to parametrize the SGS stresses and the extension to scalar transport equations developed by Moin *et al.*[33] is used to model the SGS heat flux. Isothermal boundary conditions for the temperature† were used in the simulations—a hot top wall (non-dimensional temperature 0·5) and a cold bottom wall (non-dimensional temperature $-0{\cdot}5$). The simulation results were compared with the direct numerical simulations of Kim *et al.*[34] and Kim and Moin.[35] The direct simulations were conducted on a $192 \times 160 \times 129$ grid (after de-aliasing) using a fully spectral (Fourier–Chebyshev) method; the validity of the DNS results was established by comparisons with previous experimental results.[34]

The computational statistics of the LES runs are shown in Table II; the CPU time listed in the table is the overall CPU time (this includes the time in the flow solver, the scalar solver, the subgrid-scale model, the routines for computing run time stastistics of various turbulence quantities and the file I/O modules).

In Figure 9 the mean velocity and temperature profiles are shown as a function of the distance from the wall in local wall co-ordinates. The computed results show very good agreement with the DNS results and are also in accord with the linear profile in the viscous sublayer and the log profile in the core region of the channel. The good agreement between the LES and DNS results indicates that the LES (even on the coarse grid) is resolving the mean quantities accurately. The resolved turbulence intensities are compared with the DNS results in Figure 10. The large-eddy simulations predict a somewhat higher u intensity and lower v and w intensities, but the overall agreement is good. This behaviour has also been observed by others.[36] The maximum deviation between the LES and DNS

* Note that these are the grid sizes before de-aliasing is applied.
† We have assumed a linear equation of state relating the density ρ^* to the temperature T.

Table II. Computational statistics for LES runs of unstratified channel flow

Grid	Integration time (h/u_τ units)	Δt (h/u_τ units)	Time steps	CPU time (h)
$64 \times 64 \times 65$	14·0	$2{\cdot}5 \times 10^{-3}$	5600	15·0
$128 \times 128 \times 97$	8·5	$1{\cdot}25 \times 10^{-3}$	6800	35·0

results occurs in the vertical turbulence intensity; this is expected because the vertical component w, which largely depends on small scales (particularly close to the walls), is most sensitive to grid resolution. From the figure we can also infer the improvements as a result of grid refinement and can conclude that the numerical results are converging (to the exact result) as the grid resolution is increased.

The code has been used to perform simulations of stably stratified turbulent closed (both surfaces are no-slip) and open (free-slip top surface) channel flows to study the effects of stratification on near-wall and near-free-surface turbulence. Simulations with various Richardson and Reynolds numbers and temperature (density) boundary conditions have been performed. The results show consistent trends as to the effects of stable stratification on channel turbulence. A large database has been generated which we are currently studying in greater detail to develop better understanding of stratified flows and to evaluate various turbulence closures. Here we shall present some results of grid refinement studies which we were performed to ascertain the adequacy of grid resolution.

The simulations for the closed channel were conducted at $Re_\tau = 180$ and $Pr = 0{\cdot}71$ (heated air); stable stratification was maintained by fixing the wall temperatures (heated top wall and cold bottom wall). Large-eddy simulations over a range of Ri_τ (0–60) were performed on a $64 \times 64 \times 65$ grid. A fine LES ($128 \times 128 \times 97$ grid, 64 processors, $Ri_\tau = 18$) and a DNS ($128 \times 256 \times 160$ grid, 64 processors, $Ri_\tau = 45$) were also performed to study the accuracy of the coarse grid runs and to accurately compute higher-order statistical quantities. The $Ri_\tau = 18$ flow is a moderately stratified flow in which turbulence is active but suppressed compared with the unstratified case, while in the

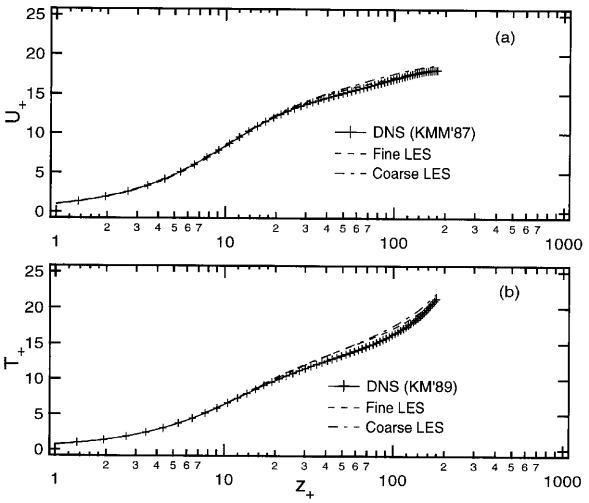

Figure 9. Comparison of LES results with DNS results: (a) mean velocity, comparisons with DNS of Kim *et al.*;[34] (b) mean temperature, comparisons with DNS of Kim and Moin[35]

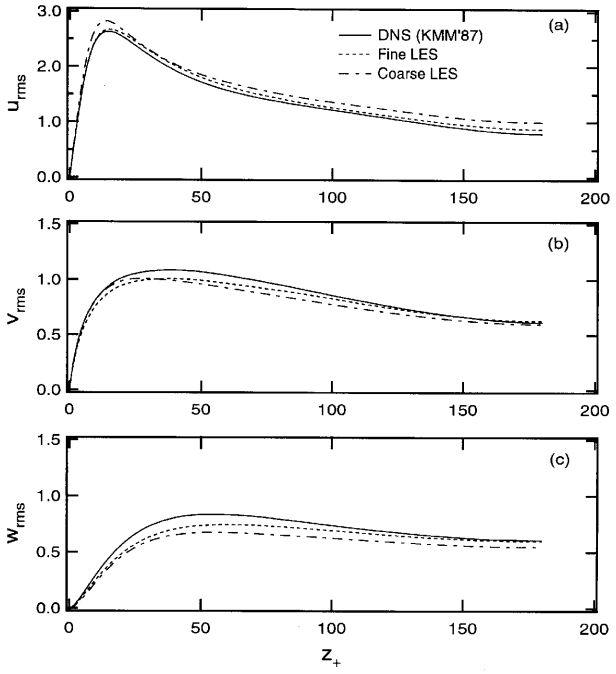

Figure 10. Comparison of turbulent intensities with DNS of Kim *et al.*[34]

$Ri_\tau = 45$ case the flow relaminarizes and the turbulence is completely damped out. The simulations were all started with an initial flow field obtained from simulation of the passively heated (i.e. unstratified) turbulent channel flow.

The one-dimensional energy spectra of vertical kinetic energy for $Ri_\tau = 0$ and 18 flows are shown in Figure 11 at $z_+ = 37$ (in the buffer layer) and $z_+ = 144$ (close to the channel centre). Stable stratification directly reduces the vertical kinetic energy (owing to its conversion to potential energy), making the vertical turbulent fluctuations most sensitive to grid resolution. We note that in the buffer

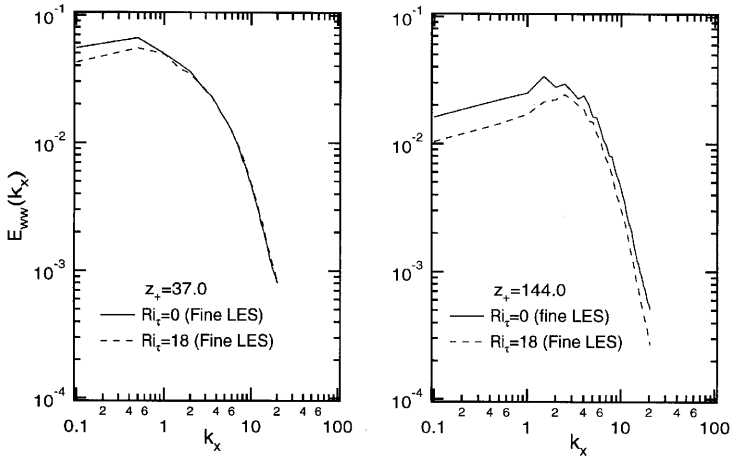

Figure 11. Comparison of 1D spectra of vertical kinetic energy (at two vertical locations) between unstratified and stably stratified channel flow LES; $Re_\tau = 180$, $Pr = 0{\cdot}71$

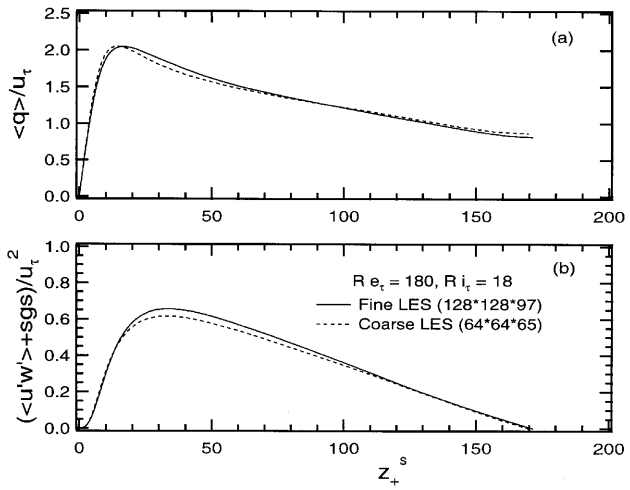

Figure 12. Distribution of (a) RMS resolved TKE and (b) Reynolds shear stress (resolved plus subgrid-scale) on coarse and fine grids in LES of stably stratified channel flow; $Re_\tau = 180, Ri_\tau = 18, Pr = 0{\cdot}71$

layer only the larger vertical scales are suppressed, while near the channel centre all the vertical scales are suppressed. For both the stratified and unstratified cases the spectra show adequate resolution, as indicated by the fact that the energy at low wave numbers is two to three decades higher than at high wave numbers and there is no evidence of energy pile-up at high wave numbers. The distributions of RMS resolved TKE and Reynolds shear stress are shown in Figure 12; again, good agreement between the two grids is evident. The individual components of turbulence intensity show the same trends as observed in the unstratified case: u_{rms} is slightly overpredicted while v_{rms} and w_{rms} are underpredicted on the coarse grid. As expected, the coarse grid also slightly underpredicts $\langle u'w' \rangle$, but the overall agreement with the fine grid result is good. The unstratified case yields similar comparisons between the fine and coarse grid simulations (see Figure 10).

The comparison between the DNS and LES results for the strongly stratified flow is presented in Figure 13. The two quantities shown (both have been non-dimensionalized) are (i) the instantaneous plane-averaged wall shear and (ii) the available potential energy function, which is defined as $APEF = -\frac{1}{2} Ri_\tau \langle \rho'^2 \rangle / (\mathrm{d}\langle \rho \rangle / \mathrm{d}z)$. Here $\langle \rho'^2 \rangle$ is the instantaneous density variance and $\langle \rho \rangle$ is the instantaneous mean density. *APEF* is an important concept in stratified flows and represents the potential energy of density fluctuations available for conversion to kinetic energy (by the process of adiabatic redistribution of mass to a statically stable state[37]). Under very strong stratification both these quantities decay rapidly as the flow relaminarizes. The results in Figure 13(a) show very good agreement between LES and DNS at all times. Note that the initial values are slightly different as the initial fields for the coarse and fine grids are instantaneously different (both were obtained from corresponding unstratified runs).

The decay of *APEF*, which is more sensitive to grid resolution than is wall shear, also shows good agreement between the LES and DNS results for $t < 2{\cdot}5$; for $t > 2{\cdot}5$ the agreement is not so good. This is probably due to the fact that at later times a significant restratification process, marked by a gravity-induced reconversion of potential energy into vertical kinetic energy and counter-gradient buoyancy fluxes, takes place. Analysis of DNS data[38] shows that the largest scales of motion are strongly suppressed (at later times) and the peak of buoyancy flux co-spectra shifts to smaller scales. Since these scales are not resolved in the LES calculation, discrepancies in the decay rate of *APEF* occur.

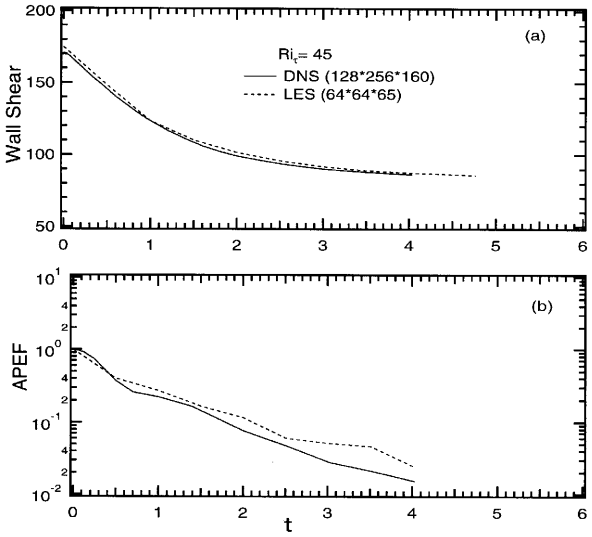

Figure 13. Comparison of (a) wall shear and (b) available potential energy function between DNS and LES in strongly stratified channel flow; $Re_\tau = 180$, $Pr = 0{\cdot}71$

7. PERFORMANCE RESULTS

In this section the results of timing measurements of the parallel implementation are presented. These measurements were made on a 32-node iPSC/860 Hypercube computer at Stanford and the 400-node Paragon XP/S at SDSC. The results were obtained by advancing the solution for 50 time steps (for each run) to obtain averaged execution times. All the times presented are for direct numerical simulation of a homogeneous fluid and do not include the time required for solution of the scalar transport equation and the subgrid-scale model (in large-eddy simulations).* The runs were performed in dedicated batch queues and the longest of the elapsed times was taken as the parallel execution time.† The velocity field was initialized as low-amplitude (5% of mean) divergence-free random perturbations superposed on a laminar flow profile. All the times are for 64 bit arithmetic (double-precision) unless otherwise stated. We first present overall performance results, then detailed timings.

7.1. Overall performance

The algorithm was evaluated both in terms of scalability as well as in terms of absolute performance, i.e. execution times.‡ Constant problem size (or unscaled speed-up) scaling was used to measure the scaling characteristics of the algorithm. This was motivated by the fact that the problem size (grid) requirements for LES and DNS computations are dictated by the fluid dynamic

* The time required for solution of the scalar transport equation is approximately equal to one-third of the times required in the first and second stages of the fractional step algorithm. The time required in the subgrid-scale model depends on the model used to parametrize the subgrid-scale stresses.

† Since there is no global clock on the iPSC/860 and Paragon, the time on each processor was measured separately and the maximum value chosen; owing to slight load imbalances and different clocks, the measured times differ on all processors.

‡ Since the solution method is based on a direct solver, the execution time and rate of execution (MFLOPS) are equivalent measures of absolute performance. Note that this is not the case for iterative solvers, where the rate of convergence (or the number of iterations required to reach a certain tolerance level) also needs to be considered in evaluation the absolute performance.

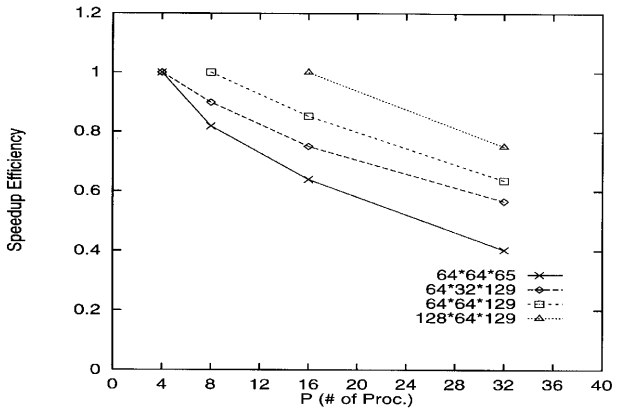

Figure 14. Speed-up efficiency S_η versus number of processors for iPSC/860 Hypercube

parameters, i.e. the Reynolds, Prandtl and Richardson numbers. Constant problem size scaling is thus the appropriate metric to evaluate the efficiency of parallel implementation and determine the problem granularity§ for the grid sizes chosen based on the resolution requirements of the simulations.

The measured relative speed-up efficiency S_η versus the number of processors, P, is shown in Figures 14 and 15 on iPSC/860 and Paragon respectively for different grids. S_η is defined as $(T_{\min}/T_P) \times (P_{\min}/P)$, where $P_{\min}$ is the minimum number of nodes on which a given problem size fits, $T_{\min}$ is the CPU time for $P_{\min}$ processors and T_P is the measured time for P processors. Since the problem sizes are too large to run on a single node, the speed-up is measured with respect to the minimum number of processors on which the problem can be run. An estimate of the single-processor time can be made (for the iPSC/860) using the procedure outlined in Reference 39, but it requires accurate measurements of the communication, computation and idle time of each processor. This is a

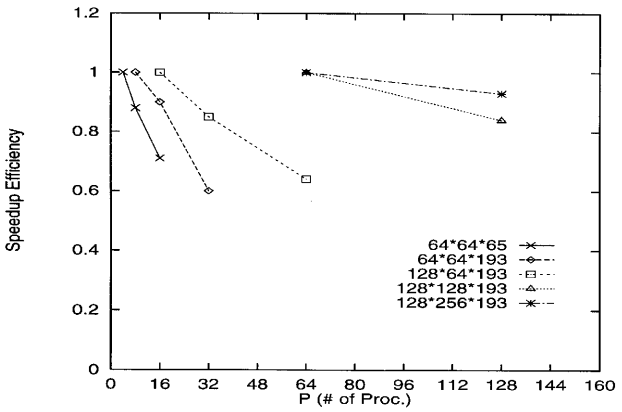

Figure 15. Speed-up efficiency S_η versus number of processors for Paragon XP/S (OSF 1·2)

§ For a given problem size, owing to the algorithmic and hardware scalability limits, there is a limit to which we can increase the number of processors.

complicated and error-prone task for the algorithm considered here. The relative speed-up efficiency is easier to compute and gives an accurate indication of the scalability of the algorithm for the problem size of interest.

For a fixed number of processors, as the grid size is increased, the efficiency also increases, indicating that the ratio of communication to computation decreases with increasing grid size and the algorithm scales with the grid size. From the figures we also see that as the number of processors is increased, the efficiency drops faster for smaller-sized problems. This is because the communication time increases faster than the execution time decreases. We note that an efficiency of up to 91% on doubling the number of processors and up to 60% on an eightfold increase in the number of processors is obtained. Also as expected, the efficiency is higher on the Paragon because the communication on it is faster than on the iPSC/860, while the computation is nearly identical for the two machines (compare the $64 \times 64 \times 65$ grid results in Figures 14 and 15). The scaled speed-up of the algorithm can also be inferred from the plots; we see that S_η decreases by only a small amount when the number of processors is increased as the grid size is increased, implying a nearly fixed execution time as required by the fixed execution time (i.e. scaled speed-up) scaling. The results in Figure 16 confirm that a good fixed execution time scaling is obtained in the present implementation.

The algorithm was also implemented and vectorized on a Cray-YMP. The performance on a single-processor Cray-YMP is shown in Table III. As expected, the performance increases with the

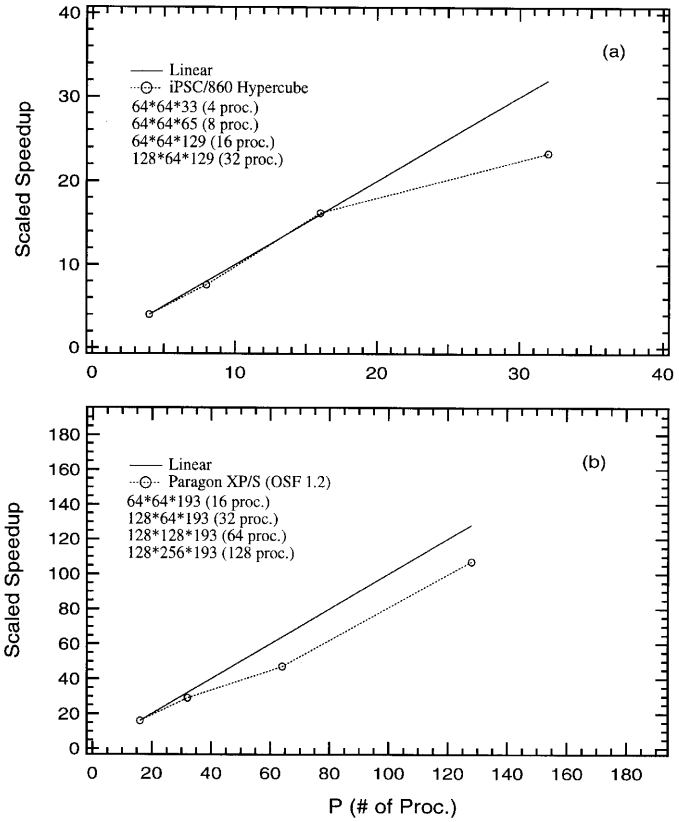

Figure 16. Scaled speed-up on (a) iPSC/860 Hypercube and (b) Paragon XP/S

Table III. Measured CPU time per time step per grid point and MFLOPS on CRAY-YMP (single processor) for different grids

Grid	CPU_n (μs)	MFLOPS
$64 \times 64 \times 65$	14·94	153
$128 \times 64 \times 97$	14·67	163
$128 \times 64 \times 193$	15·48	154
$128 \times 128 \times 193$	18·42	174
$128 \times 256 \times 97$	18·38	185

grid size (increasing vector length) and peaks at 185 MFLOPS (or about 56% of theoretical peak performance) on the grids that were tested. Even for the smallest grid a performance of 153 MFLOPS is obtained. Note that as the grid size increases, CPU_n (the CPUtime per time step per grid point) is increasing as well; this is because for FFTs the operation count per grid point is $O(\log_2 N_x + \log_2 N_y)$, so the number of operations increases logarithmically with the grid size. However, with increasing grid size the performance also increases and as a result the values of CPU_n for $128 \times 128 \times 193$ and $128 \times 256 \times 193$ grids are almost identical. On a Cray C-90 these timings will be approximately a factor of two to three faster.*

A comparison between Cray-YMP and Paragon CPU_n timings is presented in Table IV for several grids. This is a good measure of performance, since in this way the performance of the algorithm can be measured for different grids. From this table it is clear that the performance on the Paragon is strongly dependent on the problem size, in contrast with the YMP where it is fairly independent of the grid size. We see that the performance on approximately 32 processors of the Paragon is equivalent to that of a single-processor YMP. Furthermore, as expected, for a fixed number of processors, CPU_n decreases as the grid size is increased (e.g. compare CPU_n for 64 processors) and the best times are

Table IV. Performance comparison between Paragon (OSF 1·2) and Cray-YMP/1 timings

Grid	Cray-YMP $CPU_n(\mu s)$	Paragon		Ratio to YMP/1
		P	$CPU_n(\mu s)$	
$64 \times 64 \times 65$	14·94	4	81·0	0·18
		8	46·0	0·32
		16	28·60	0·52
$64 \times 64 \times 193$	15·0	8	41·2	0·37
		16	23·0	0·65
		32	17·1	0·88
$128 \times 64 \times 193$	15·50	16	21·7	0·71
		32	12·7	1·22
		64	8·23	1·88
$128 \times 128 \times 193$	18·40	64	7·83	2·35
		128	4·64	3·97
$128 \times 256 \times 193$	18·40	64	6·38	2·88
		128	3·45	5·33

* The speed-up will be dependent on the problem size; for larger problem sizes for which the average vector length is greater than 128 (the asymptotic vector length for the Cray C-90) the speed-up will be a factor of three.

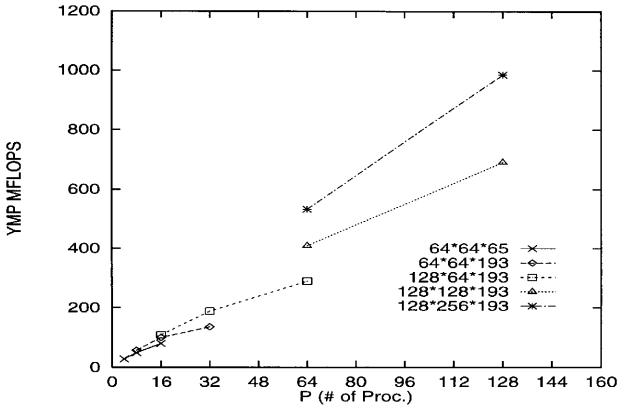

Figure 17. Cray-YMP/1 equivalent MFLOPS on Paragon (OSF 1·2)

obtained for the largest grid sizes. For the grid sizes considered here, with 128 processors, the parallel code runs up to five times faster than the Cray-YMP implementation.

In Figure 17 the MFLOP ratings of the YMP have been used to compute YMP equivalent ratings of the Paragon. Speeds of up to 1 GFLOPS were obtained on 128 processors of the Paragon. From the plot it is also clear that a consistent scalability over the range of problem and processor sizes considered is obtained. An indication of this is given by the slopes of the curves (in Figure 17), which are not decreasing or levelling out as the problem and processor sizes are increased.

The MFLOPS/node obtained on the Paragon are shown in Figure 18. This is another measure of scalability,[6] because if, over a certain range of processor numbers, it reaches a constant value, then the total performance scales linearly over that range. The tendency of the curves to flatten out can be seen clearly in Figure 18. The performances vary between 3 and 9 MFLOPS/node, i.e. 8%–10% of theoretical peak performance* of the Paragon. In contrast, a performance of over 50% of theoretical peak performance was obtained on the CRAY-YMP/1. Such low performances (in terms of theoretical peak performance) are not uncommon on present-generation massively parallel computers for CFD coses.[4,40] The primary reason for the large discrepancy between peak and actual performance is the large difference between the floating point rate of the i860XP microprocessor and the memory bandwidth—both on-chip, i.e. CPU local memory bandwidth,[41] as well as remote, i.e. communication bandwidth. The small size of the on-chip data cache and the long memory access times (compared with floating point operations) significantly limit the single-node performance on the Paragon. Realizable communication bandwidths of 45–50 MB s^{-1} (compared with the hardware maximum of 200 MB s^{-1}) and latencies of the order of 50 μs under OSF 1.2[42] limit the achievable scalability of the applications. As will be shown later, the single-node performance is the main reason for the poor performance on smaller grids, whereas for larger grids the communication performance also significantly affects the overall performance.

The CPU time per time step per grid point, CPU_n, is shown in Table V for the iPSC/860 and Paragon. The timings on the Paragon are on average 30%–50% faster than those on the iPSC/860, with the higher end improvements observed mostly for problems where 2D partitioning was used. The reason for this is presented in the next subsection along with results of detailed timing measurements.

* The peak performance of the i860XP chip is 76 MFLOPS for 64 bit arithmetic.

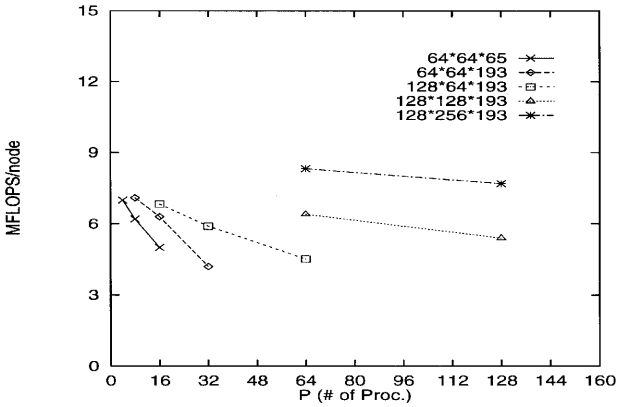

Figure 18. MFLOPS/node on Paragon (OSF 1·2)

7.2. Detailed performance

In Section 3 the speed-up analysis of the model problem showed that the type of data partitioning affects the extractable parallelism, dependence overhead and communication costs and requirements. The results showed that the unipartition scheme gives better efficiencies than the multipartition and transpose partition schemes. In this subsection we shall present comparisons for the three types of unipartitionings, namely the 1D-*x*, 1D-*z* and 2D schemes, and also analyse the performance of the individual stages of the fractional step algorithm.

In the computation of 2D FFTs by the transpose method the communication cost scales as $(P_x - 1)N_xN_y/P_x^2$, which is approximately the total number of words of data transferred (in $P_x - 1$ transmissions) between each processor in the *x*-direction. On the other hand, the communication cost for finite difference operations in the vertical direction scales as $O(N_xN_y/P_x)$. Clearly, more communication is required in spectral operations than in finite difference operations. Furthermore, while only near-neighbour communication is needed for vertical operations, the global exchange algorithm requires long-distance data exchange. Thus, despite dependence delays in vertical decomposition, the communication in horizontal partitioning makes it more costly than vertical partitioning. For the same number of grids (with $N_x \approx N_y \approx N_z$) and the same number of processors, 1D-*x* partitioning is significantly inferior to the other two decompositions.

From the analysis presented in Section 3, the communication times for evaluation of the explicit right-hand side for the model problem can be estimated as

$$(T_{\text{comm}}^{\text{2d}})_{\text{rhs}} \approx 8(P_x - 1)\left\lceil \frac{N_z}{P_z} \right\rceil \left(\lambda + \frac{N_xN_y}{P_x^2} b\beta\right) + 4\left(\lambda + \frac{N_xN_y}{P_x} b\beta\right) \tag{48}$$

Table V. Measured CPU times (in seconds) on iPSC/860 (I) and Paragon (P) for different grids and processors (denoted *P*)

Grid	*P*	CPU_n(I)	CPU_n(P)
$32 \times 32 \times 33$	2	$2{\cdot}48 \times 10^{-4}$	$1{\cdot}85 \times 10^{-4}$
	4	$1{\cdot}51 \times 10^{-4}$	$1{\cdot}41 \times 10^{-4}$
	8	$1{\cdot}07 \times 10^{-4}$	$1{\cdot}33 \times 10^{-4}$
$64 \times 64 \times 65$	4	$1{\cdot}26 \times 10^{-4}$	$8{\cdot}1 \times 10^{-5}$
	8	$7{\cdot}7 \times 10^{-5}$	$4{\cdot}6 \times 10^{-5}$
	16	$4{\cdot}90 \times 10^{-5}$	$2{\cdot}86 \times 10^{-5}$

and

$$(T^{1dz}_{comm})_{rhs} \approx 4(\lambda + N_x N_y b\beta) \tag{49}$$

for the 2D and 1D-z unipartitionings respectively. The first term in equation (48) represents the communication time in FFTs and the second term is the communication time in finite difference derivatives. It is easy to see that for typical problem sizes, such as the ones presented in Section 6.1, $(T^{2d}_{comm})_{rhs} > (T^{1dz}_{comm})_{rhs}$, while the computation times are almost identical. Consequently, the 1D-z scheme is expected to yield a better overall time as well as speed-up for the first stage of the fractional step algorithm.

For the second stage (i.e. the solution of the distributed tridiagonal system of equations) the communication times for the model problem are

$$(T^{2d}_{comm})_{tri} \approx \left(\frac{N_x N_y}{P_x l_p} + P_z\right)\lambda + \left(\frac{4N_x N_y}{P_x} + 4P_z l_p\right) b\beta, \tag{50}$$

$$(T^{1dz}_{comm})_{tri} \approx \left(\frac{N_x N_y}{l_p} + P\right)\lambda + (4N_x N_y + 4Pl_p) b\beta, \tag{51}$$

$$\Rightarrow (T^{2d}_{comm})_{tri} = \frac{(T^{1dz}_{comm})_{tri}}{P_x}. \tag{52}$$

The above equations indicate that for this stage the 1D-z partitioning is inferior to the 2D scheme and will give poorer scalability in comparison with the 2D scheme as the number of processors (partitions in x) is increased. For the computation of tridiagonal solves the 2D partitioning exposes a higher explicit parallelism (P_x solves can go completely in parallel) and decreases the amount as well as frequency of data each processor transmits to its nearest north/south neighbour, thereby reducing both the transfer and message start-up times in the communication.

In Figure 19 the CPU time per step is shown for 1D-z and 2D partitioning* ($P_x = 4$) for different grids. The results in Figure 19 indicate that up to 16 processors the 1D-z scheme gives better timings than the 2D scheme. However, for a fixed problem size the 2D partitioning has better scalability and for 32 processors (and higher on the Paragon) it yields lower times compared with 1D-z partitioning. The figure also shows that for both partitionings a nearly linear scaled speed-up is obtained when N_z is doubled, i.e. the execution time changes little when both the problem size and the number of processors are doubled. In fact, the times for grids 2 and 3 are nearly identical, indicating a higher contribution of communication start-up overhead (latency cost) to the total communication cost for grid 2. Our results on these (and other) grids also showed that 1D-z partitioning is usually better when N_z/P_z is relatively high (see equations (48) and (49)).

The results for the individual stages of the fractional step algorithm are shown in Figure 20 (on the iPSC/860). We can see clearly that for the right-hand-side calculation the 1D-z scheme has a lower execution time and better speed-up than the 2D scheme. However, as expected (based on the discussion presented above), for the velocity and pressure steps it shows inferior speed-up compared with the 2D scheme. In fact, for 32 processors the velocity step becomes the most expensive of the three stages when 1D-z partitioning is used (Figure 20). For the grids on which we performed the simulations, for $P > 32$ in all cases the 2D scheme was used as it gave better performance (lower execution time) than the 1D-z scheme. From Figure 20 we also see that the pressure step is much less

* The 2D partitioning was evaluated for several different values of P_x and P_z. On the 32-node iPSC/860, $P_z = 4$ was found to give the best results.

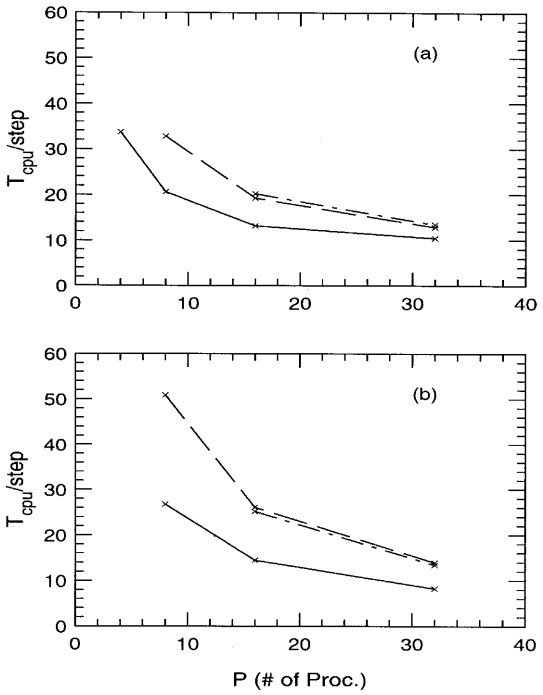

Figure 19. CPU time (in seconds) per time step versus number of processors on iPSC/860 for different grids: ————, $64 \times 64 \times 64$; -----, $64 \times 64 \times 128$, —·—·—, $128 \times 64 \times 128$; (a) 1D-z partitioning; (b) 2D partitioning, $P_x = 4$

expensive than the first two stages of the algorithm, thus validating the assumption made for the analysis presented in Section 4.

A detailed breakdown of the various stages is shown in Table VI. For both grids, as expected, increasing P_z (with P_x fixed) increases the time in the second and third stages. The second stage requires 42% of the total time with just four processors (for the smaller grid) and an increase in P_z increases this time to 57% of the total time (for 16 processors) with a corresponding decrease in the time for the first stage. For the larger grid the first stage is the dominant stage, as distributed FFTs are required. However, the second stage still takes a substantial percentage of the total time. These results show consistency, as the characteristics of the algorithm (expressed by the amount of time each stage requires) are preserved by the parallel implementation, when the grid sizes and processor sizes are varied over a large range (such as those in Table VI).

On parallel machines it is often advantageous to use single-precision (32 bit) as opposed to double-precision (64 bit) arithmetic. With 32 bit operands the amount of data to be transferred and the memory requirements are reduced by half, leading to better performance and scalability. In many algorithms this may not be possible, but the present algorithms is numerically well-conditioned and 32 bit precision is adequate. The effect of word length on performance is shown in Table VII. For the smaller grid the performance improvement by using 32 bit arithmetic is primarily due to increased computational speed, as for this grid the communication workload is significantly less than the computation workload. For the larger grid we see almost 50% improvements; for this case the communication time is a larger fraction of the total time and consequently the use of single-precision arithmetic has a bigger effect on performance compared with the smaller grid.

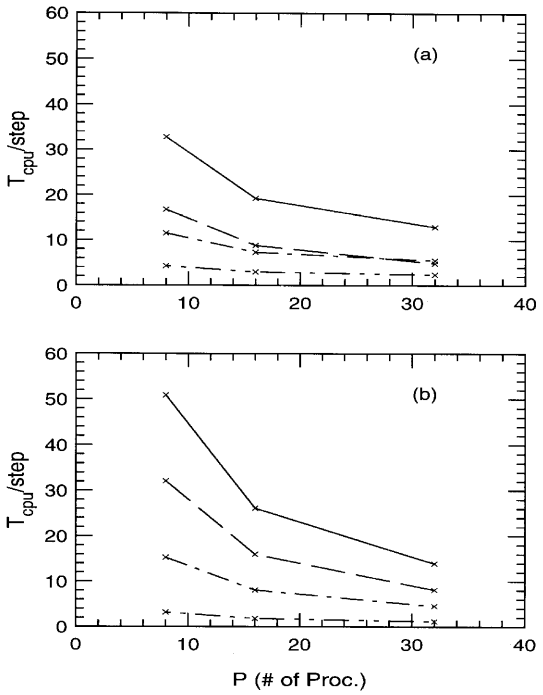

Figure 20. CPU time (in seconds) per time step versus number of processors on iPSC/860 for different stages: 64 × 64 × 128 grid; ————, total; -----, RHS; —·—·—, velocity step, —··—, pressure step; (a) 1D-*z* partitioning; (b) 2D partitioning, $P_x = 4$

Table VI. Breakdown of different stages as percentage of total execution time on Paragon (OSF 1·2)

Grid	P	RHS (%)	VT (%)	PT (%)
64 × 64 × 65	4	45·0	42·0	13·0
$P_x = 1$	8	38·0	49·0	13·0
	16	29·0	57·0	14·0
128 × 256 × 193	64	54·0	35·0	11·0
$P_x = 4$	128	51·0	37·0	12·0

Table VII. Comparison of 32 bit versus 64 bit performance on Paragon (OSF 1·2)

Grid	P	T(64 bit)/T(32 bit)
64 × 64 × 65	4	1·22
$P_x = 1$		

8. CONCLUSIONS

A scalable parallel algorithm for spectral finite difference simulation of turbulent stratified flows was implemented and its performance was evaluated. The performance was found to depend strongly on the problem size and the type of data decomposition. Three types of partitioning were investigated for a model problem and the analysis showed that multipartition schemes are ill-suited for this algorithm. While here the analysis was used to evaluate the partition schemes on the iPSC/860, it is general and applicable to any distributed memory message-passing architecture. It thus provides an analytical framework to evaluate the communication overheads of different schemes on different architectures and thus obtain useful information on the suitability or non-suitability of a particular decomposition strategy on a given machine. This procedure is very useful, as comparison based on actual implementation of the different schemes is non-trivial, particularly for large codes such as the present one. Based on the analysis, we concluded that unipartitioning schemes (1D and 2D) are optimal for this problem and unscaled speed-up efficiencies of up to 91% were obtained in actual implementation. The scalability was evaluated for several different grids over a range of 8–128 processors and consistent speed-up efficiencies were achieved. The performance was compared with a well-optimized implementation on the Cray–YMP; on 128 processors of the Paragon, speeds up to five times faster than on a single processor of the Cray-YMP were obtained. Performance between 8% and 10% of theoretical peak performance was obtained on the Paragon. The comparison between the iPSC/860 and Paragon showed 30%–50% faster timings on the Paragon.

Three different unipartitioning schemes were investigated and the performance of the individual stages of the algorithm was studied in detail. It was found that while 1D-z partitioning was better for the first stage, 2D partitioning was better for the second and third stages of the algorithm. The speed-up analysis presented for the model problem was used to provide explanations for these observations. The effect of word length on performance was also investigated and the results were found to be consistent with the computation and communication characteristics of the method. A careful study of the numerical accuracy and adequacy of the grid resolution in simulations of stably stratified turbulent channel flows was also performed. The results established the validity and correctness of the code.

On parallel machines the mapping of the communication pattern of the algorithm to the processor topology is important and performance metrics that are used for sequential algorithms (total number of floating point and memory operations) need to be altered. The data dependencies in the algorithm, communication and synchronization cost and load balancing need to be considered in the evaluation of a parallel algorithm. The different stages of the algorithm may require different partitioning schemes and trade-offs between these schemes have to be analysed. Finally, while it is hard to quantify, the programming aspect of various implementation strategies also needs to be considered in the development (porting) of application codes on (to) parallel machines.

ACKNOWLEDGEMENTS

The authors extend thanks to Professor R. W. Dutton for providing access to the iPSC/860 Hypercube operated by his group, to Dan Yergeau for his help with getting us started on the iPSC/860, to Dr. J. P. Singh of Stanford Computer Science department, Ken Steube of SDSC, Brad Rullman of Intel SSD and Dr. Rob F. Van Der Wijngaart of NASA Ames Research Center for many helpful discussions on parallel implementation. The support of SDSC for providing computer time on the Intel Paragon is also gratefully acknowledged. The computer time on the Cray YMP was provided by the Central Computing Facility at Los Alamos National Laboratory. Financial support was provided by ONR under grant N00014-92-J-1611.

REFERENCES

1. P. Moin and J. Kim, 'Numerical investigation of turbulent channel flow', *J. Fluid Mech.*, **118**, 341 (1982).
2. R. B. Pelz, 'The parallel Fourier pseudospectral method', *J. Comput. Phys.*, **92**, 296 (1991).
3. E. Jackson, Z. She and S. A. Orszag, 'A case study in parallel computing: I. Homogeneous turbulence on a Hypercube', *J. Sci. Comput.*, **6**, 27 (1991).
4. R. D. Joslin and M. Zubair, 'Parallel spatial direct numerical simulations on the Intel iPSC/860 Hypercube', *ICASE Rep. 93–53*, ICASE, NASA Langley Research Center, 1993.
5. U. R. Hanebutte, R. D. Joslin and M. Zubair, 'Scalability study of parallel spatial direct numerical simulation code on IBM-SP1 parallel supercomputer', *ICASE Rep. 94–80*, ICASE, NASA Langley Research Center, 1994.
6. T. M. Eidson and G. Erlebacher, 'Implementation of a fully load-balanced periodic tridiagonal solver on a parallel distributed memory architecture', *ICASE Rep 94–37*, ICASE, NASA Langley Research Center, 1994.
7. S. Chen and X. Shan, 'High-resolution turbulent simulations using the Connection Machine 2', *Comput. Phys.*, **6**, 643 (1992).
8. T. A. Cortese and S. Balachandar, 'High performance spectral simulation of turbulent flows in massively parallel-machines with distributed memory', *TAM Rep. 765*, University of Illinois at Urbana-Champaign, 1994.
9. J. B. Perot, 'Direct numerical simulation of turbulence on the Connection Machines', *Proc. Conf. on Parallel CFD*, Princeton, NJ, 1992.
10. D. Tafti, 'Features and implementation issues for a high order finite-difference algorithm for direct and large eddy simulations of incompressible turbulence on CM-5f', *NCSA Rep. P047*, NCSA, University of Illinois at Urbana-Champaign, 1994.
11. P. F. Fischer and A. Patera, 'Parallel simulation of viscous incompressible flows', *Ann. Rev. Fluid Mech.*, **26**, 483 (1994).
12. P. F. Fischer, L. W. Ho, G. E. Karniadakis, E. M. Ronquist and A. T. Patera, 'Recent advances in parallel spectral-element simulation of unsteady incompressible flows', *Comput. Struct.*, **30**, 217 (1988).
13. R. K. Agarwal and J. C. Lewis, 'Computational fluid dynamics on parallel processors', *Comput. Sys. Eng.*, **3**, 251 (1992).
14. C. Canuto, M. Y. Hussaini, A. Quarteroni and T. A. Zang, *Spectral Methods in Fluid Dynamics*, Springer, Berlin, 1988.
15. W. M. Washington and C. L. Parkinson, *An Introduction to Three-Dimensional Climate Modeling*, University Science Books, Mill Valley, CA, 1986.
16. P. M. Gresho, 'Incompressible fluid dynamics: some fundamental formulation issues', *Ann. Rev. Fluid Mech.*, **23**, 413 (1991).
17. T. A. Zang, 'On the rotational and skew-symmetric forms for incompressible flow simulations', *Appl. Numer. Math.*, **7**, 27 (1991).
18. J. B. Perot, 'An analysis of the fractional step method', *J. Comput. Phys.*, **108**, 51 (1993).
19. H. Choi and P. Moin, 'Effects of computational time step on numerical solutions of turbulent flow', *J. Comput. Phys.*, **113**, 1 (1994).
20. A. J. Chorin, 'Numerical solution of the Navier–Stokes equations', *Math. Comput.*, **22**, 745 (1968).
21. R. Temam, *Arch. Rat. Mech. Anal.*, **32**, 377 (1969).
22. J. Kim and P. Moin, 'Application of a fractional step method to incompressible Navier–Stokes equations', *J. Comput. Phys.*, **59**, 308 (1985).
23. T. Z. Zang and M. Y. Hussaini, 'On spectral multigrid methods for the time-dependent Navier–Stokes equations', *Appl. Math. Comput.*, **19**, 359 (1986).
24. J. K. Dukowicz and A. S. Dvinsky, 'Approximate factorization as a high order splitting for the implicit incompressible flow equations', *J. Comput. Phys.*, **102**, 336 (1992).
25. T. A. Zang and M. Y. Hussaini, 'A three-dimensional spectral algorithm for simulations of transition and turbulence', *ICASE Rep. 85–19*, ICASE, NASA Langley Research Center, 1985.
26. S. H. Bokhari, 'Multiphase complete exchange on a circuit switched hypercube', *ICASE Rep. 91–5*, ICASE NASA Langley Research Center, 1991.
27. F. Darema, D. A. George, V. A. Norton and G. F. Pfister, 'A single-program multiple-data computational model for EPEX/FORTRAN', *Parallel Comput.*, **7**, 11 (1988).
28. N. H. Naik, V. K. Naik and M. Nicoules, 'Parallelization of a class of implicit finite difference schemes in computational fluid dynamics', *Int. J. High Speed Comput.*, **5**, (1993).
29. A. Sivasubramaniam, A. Singla, U. Ramachandran and H. Venkateswaran, 'On characterizing bandwidth requirements of parallel applications', *Proc. ACM SIGMETRICS '95*, Ottawa, 1995, p. 198.
30. D. Scott, 'Efficient all-to-all communication patterns in hypercube and mesh topologies', *Proc. Sixth Distributed Memory Computing Conf.*, 1991, p. 398.
31. M. Germano, U. Piomelli, P. Moin and W. H. Cabot, 'A dynamic subgrid-scale eddy viscosity model', *Phys. Fluids A*, **3**, 1760 (1991).
32. D. K. Lilly, 'A proposed modification of the Germano subgrid-scale closure method', *Phys. Fluids A*, **4**, (1992).
33. P. Moin, K. Squires, W. H. Cabot and S. Lee, 'A dynamic subgrid-scale model for compressible turbulence and scalar transport', *Phys. Fluids A*, **3**, (1991).
34. J. Kim, P. Moin and R. D. Moser, 'Turbulence statistics in fully developed channel flow at low Reynolds number', *J. Fluid Mech.*, **177**, 133 (1987).
35. J. Kim and P. Moin, 'Transport of passive scalars in a turbulent channel flow', in J. C. André, J. Cousteix, F. Durst, B. E. Launder, F. W. Schmidt and J. H. Whitelaw (eds), *Turbulent Shear Flows 6*, Springer, New York, 1989, p. 86.

36. U. Piomelli, J. H. Ferziger and P. Moin, 'Models for large eddy simulations of turbulent channel flows including transpiration', *Department of Mechanical Engineering Rep. TF-32*, Stanford University, 1988.
37. A. E. Gill, *Atmosphere–Ocean Dynamics*, Academic, San Diego, CA, 1982.
38. R. P. Garg, 'Physics and modeling of stratified turbulent channel flows', *Ph.D. Thesis*, Stanford University, 1996.
39. J. L. Gustafson, G. R. Montry and R. E. Benner, 'Development of parallel methods for a 1024–processor Hypercube', *SIAM J. Sci. Stat. Comput.*, **9**, 609 (1988).
40. R. A. Fatoohi, 'Adapting a Navier–Stokes solver for three parallel machines', *J. Supercomputing*, **8**, 91 (1994).
41. S. A. Moyer, 'Performance of iPSC/860 node architecture', *Rep. IPC-TR-91-008*, Insitutue for Parallel Computation, University of Virginia, Charlottesville, VA, 1991.
42. S. Saini and H. D. Simon, 'Applications performance under OSF/1 AD and SUNMOS on Intel Paragon XP/S-15', *Proc. Supercomputing '94*, Washington, DC, 1994, p. 580.

