Highly Scalable Two- and Three-Dimensional Navier-Stokes Parallel Solvers on MIMD Multiprocessors

A. Averbuch† L. Ioffe‡ M. Israeli‡ L. Vozovoi†
†School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel
‡Faculty of Computer Science, Technion, Haifa 32000, Israel

(Received July 1995; final version accepted February 1997
appeared in J. of Supercomputing, Vol. 11, No. 1, pp. 7-39, 1997)

Abstract

In this paper we present a new parallel algorithm for the solution of the incompressible two- and three-dimensional Navier-Stokes equations. The parallelization is achieved via domain decomposition. The computational region is considered in the form of a 2-D or 3-D periodic box decomposed into parallel strips (slabs). For time discretization we use a third order multistep method of [11]. The time discretization procedure results in solving global elliptic problems of (monotonic) Helmholtz and Poisson types in each time step. For the space discretization we employ the multidomain local Fourier (MDLF) method that was developed in [9, 10, 13]. The discretization in the periodic directions is performed by the standard Fourier method. In the direction across the strips we use the Local Fourier Basis technique which involves the overlapping of the neighboring subdomains and smoothing of local functions across the interior boundaries (interfaces). The matching of the local solutions is performed by adding properly weighted interface Green's functions. Their amplitudes are found in terms of the jumps of the solution and its first derivatives at the interfaces.

The present paper extends the results of our previous works [1, 9, 10, 13] on parallel use of the MDLF method in three-fold aspects:

1. In [1] a model Navier-Stokes type system was considered which does not include the pressure term. Correspondingly, in each time step only the Helmholtz type equations were solved. It was shown that the parallel solution of this equation can be accomplished using only local (neighbor-to-neighbor) communication due to localization properies of the Helmholtz operator.

   We consider the complete Navier-Stokes system including the pressure term. The solution of the Poisson equation for pressure has the potential to degrade the performance and the achieved speedup of a parallel algorithm due to the global nature of this equation that necessitates global communication among the processors. However, we show that only a few lowest harmonics require for the global data trasfer whereas the rest of harmonics can be treated locally. Therefore, most

of the communication that is required for parallelization of the Navier-Stokes solver using the MDLF method is mainly local between adjacent subdomains (processors). Moreover, the percentage of the time spent in global communication reduces as the size of the problem increases. Thus, the present parallel algorithm is highly scalable.

2. In [1] we considered only 2-D equations. In this paper we extend the previous technique to 3-D problems.

3. Previously, the MDLF solver was implemented only on the MEIKO parallel machine. In the this paper the 2-D and 3-D Navier-Stokes solvers are implemented on three MIMD message-passing multiprocessors (a 60-processors IBM SP2, a 20-processors MOSIX[3], and a network of 10 Alpha workstations) and achieve an efficiency of more than 70% to 95%. The same code written with the PVM (parallel virtual machine[7]) software package was executed on all the above distinct computational platforms.

Detailed performance results, which include scalability analysis, are presented.

Keywords: Spectral method, local Fourier basis, Navier-Stokes, Helmholtz and Poisson equations, parallel processing, domain decomposition, local and global communication, SP2, MOSIX.

1 Introduction

The numerical solution of multidimensional nonlinear evolution equations (and especially the particularly difficult case of the incompressible Navier-Stokes equations) is a heavy computational task. At the range of parameters that is of interest in these applications, the resolution requirements in time and space increase enormously. Consequently, a huge number of degrees of freedom must be computed, and memory and CPU requirements increase beyond the capacity of present-day supercomputers.

It is widely recognized that large-scale scientific computing can be realized only on massively parallel computers, especially the flexible MIMD-type message-passing systems, which are considered to be the most promising strategy in supercomputer hardware design. Rapid development of new architectures, processors, and communication networks makes such computers, with their distributed memory, more powerful than shared-memory machines.

The ability to maintain efficiency at a fixed value by simultaneously increasing the number of processors and the size of the problem is the main feature of *scalable* parallel system. The *scalability* of a parallel system measures its capacity to increase speedup in proportion to the number of processors. It reflects a parallel system's ability to use increasing processing resources effectively. Our goal is to demonstrate that the new proposed parallel solvers for the 2-D and 3-D Navier-Stokes equations are scalable in the above sense.

The high performance of parallel machines can be realized only if efficient scalable parallelizing algorithms are supplied. Our experiments were performed on a structured grids; therefore, the geometric domain decomposition (DD) is intrinsically suited for the purposes of parallelization since neighboring subdomains can be allocated to processors with short communication links. The original problem is solved independently in each processor; then

a matching procedure is used to enforce the continuity conditions on the interfaces. The objective of any DD method is to reduce the amount of interprocessor communications and data exchange as much as possible to avoid communication and synchronization bottlenecks that reduce speedup. Currently we are working on a modified algorithm that can be solved on an unstructured grid.

For the long-term integration of PDEs and the resolution of the fine features in subdomains, it is preferable to use high-order methods, in particular spectral methods. However, the *global*[1] nature of spectral methods makes these difficult to parallelize while achieving good performance. In particular, this *global* transfer of data may destroy or degrade the possibility of achieving scalability in the above sense. Numerical parallel algorithms that are based on spectral methods require global data transfers (such as transpose in the application of fast Fourier transform (FFT)) and global communication. For a massively parallel algorithm this may cause communication and synchronization bottlenecks.

Low-communication parallel algorithms based on spectral methods have been developed for the solution of nonlinear time-dependent PDEs[9, 10, 13]. The parallelization is achieved by domain decomposition. Elemental solutions in subdomains are constructed using the local Fourier basis (LFB) method. In this method a global problem is decomposed into smooth local subproblems using a collection of overlapping bell functions. Then, the Fourier method is applied in each local interval with spectral accuracy (without exhibiting the Gibbs phenomenon). It has been shown in [9, 10, 13] that the multidomain local Fourier (MDLF) method is especially efficient for large problems using high resolutions in space. In this case the relative number of operations required to perform the projection is insignificant in comparison with what is needed to execute the FFT. Moreover, this extra work is reduced as the size of the problem grows in each domain[1]. Therefore, for the computation of large problems, which requires many degrees of freedom, the MDLF method is almost as efficient as the conventional Fourier method. The important advantage of the MDLF method, when compared to other multidomain spectral techniques, is that it enables a great simplification of the matching relations for the interface unknowns. It enables each harmonic to be matched separately, thus eliminating the global coupling of the interface unknowns. For problems with constant coefficients and in simple domains, the use of the Fourier basis permits the matching of each harmonic to be done separately, thus eliminating the global coupling of the interface unknowns. Since then, the method has been extended to handle complex geometries [14]. The decomposition strategy of the MDLF method can easily be changed to reflect different resolutions in each subdomain or to treat an unstructured grids, which makes it valuable as an adaptive algorithm.

In previous work [1] the MDLF method was applied to the parallel solution of the model Navier-Stokes problem investigated previously by R. Temam [12]. In the papers cited above the model that was used for the simulation did not contain the pressure term; therefore, the time discretization procedure results in the Helmholtz equation had to be solved in each time step. It was shown by Averbuch et al. [1] that certain elliptic operators have fast decaying Green's functions, which can be used to localize the Helmholtz operator computations, and that the application of the MDLF method to such problems is therefore highly efficient.

[1]Here *global* means that there is a transfer of data among many processors that are not adjacent to each other.

Matching of the solutions in subdomains requires only local communications between neighboring processors. Hence, the algorithm is fully scalable. The algorithm was implemented on the MEIKO multiprocessor. In work by Vozovoi et al. [15] a parallel algorithm, based on the MDLF method for the solution of the 2-D Navier-Stokes equations, was presented.

In the present paper a new algorithm, which is partly based on the algorithm of Vozovoi et al. [15], is devised and applied to the parallel solution of the 2-D and 3-D Navier-Stokes equations with pressure. The pressure term in the Navier-Stokes equations adds the necessity for global data to be transferred in each time step to the parallel solution due to the need to solve the Poisson equation in each time step. The treatment and the solution of the global Poisson equation for pressure may degrade the performance and speedup of the parallel solution. In this paper we show that for an elliptic equations of the Poisson type the size of the required global data transfer can essentially be reduced. The reason, as will be demonstrated later, is that only the lowest harmonics of the pressure are treated and matched globally. Therefore, most of the communication that is required for parallelization of the Navier-Stokes equations is mainly local, between adjacent subdomains (processors). The 2-D and 3-D Navier-Stokes solvers are implemented on three different MIMD message-passing multiprocessors: a 60-processor IBM SP2, a 20-processors MOSIX[3, 4], and a network of 10 Alpha workstations using the PVM (parallel virtual machine) software package[7]. The same program without any modification was run on all these computational platforms, which demonstrates the portability of the algorithms and its software.

The paper has the following structure: In Section 2 we present the governing equations and discuss their discretization procedures in space and time. In Section 2.4 we discuss the influence of the Poisson equation in each time step. The Helmholtz equation is mentioned in Section 3. In Section 3 we describe the computational steps of the algorithm. Details of the implementation are presented in Section 4. Performance and scalability analyses are given in Section 5. In this section we also explain the extrapolation method for problems for which we cannot perform real measurements due to memory limitations.

2 Governing Equations and Numerical Schemes

We are interested in the parallel solution of the Navier-Stokes equations, which governs the incompressible viscous flows with constant properties:

$$\frac{\partial \mathbf{v}}{\partial t} = Re^{-1}\nabla^2\mathbf{v} - \mathbf{N}(\mathbf{v}) - \nabla\Pi + \mathbf{F}, \tag{1}$$

$$\nabla\Pi = \nabla\Pi' + q,\ \Gamma = (\Pi', \mathbf{v}),\ q = \text{const},\ \Gamma(x + 2\pi, y + 2\pi, z + 2\pi) = \Gamma(x, y, z),$$

in the periodic domain $\Omega = [0, 2\pi]^d,\ d = 2, 3$. Here $\mathbf{v}(\mathbf{x}, t) = (u, v, w)$ is the velocity subject to the incompressibility constraint

$$\nabla \cdot \mathbf{v} = 0 \qquad \text{in} \quad \Omega, \tag{2}$$

$\Pi = p + \frac{1}{2}|\mathbf{v}|^2$ is the total pressure (p is the hydrostatic pressure), $\mathbf{F}$ is the external forcing, and Re is the Reynolds number. The nonlinear term is written in rotational form:

$$\mathbf{N}(\mathbf{v}) = \mathbf{v} \times (\nabla \times \mathbf{v}). \tag{3}$$

The numerical solution of the problem eqs. (1)-(3) with periodic boundary conditions requires discretization in both time and space.

2.1 Discretization in Time

The discretization in time is performed using a splitting algorithm investigated by G.E. Karniadakis et al. [11]:

$$\frac{\hat{\mathbf{v}} - \sum_{q=0}^{J_i-1} \alpha_q \mathbf{v}^{n-q}}{\Delta t} = \sum_{q=0}^{J_q-1} \beta_q (\mathbf{N}(\mathbf{v}^{n-q}) - \mathbf{F}^{n-q}), \tag{4}$$

$$\frac{\hat{\hat{\mathbf{v}}} - \hat{\mathbf{v}}}{\Delta t} = -\nabla \Pi^{n+1}, \tag{5}$$

$$\frac{\gamma_0 \mathbf{v}^{n+1} - \hat{\hat{\mathbf{v}}}}{\Delta t} = Re^{-1} \nabla^2 \mathbf{v}^{n+1}. \tag{6}$$

From eqs. (2) and (6) it follows that $\nabla \hat{\hat{\mathbf{v}}} = 0$. Therefore, eq. (5) can be rewritten in the form

$$\nabla^2 \Pi^{n+1} = \frac{1}{\Delta t} \nabla \cdot \hat{\mathbf{v}}. \tag{7}$$

The splitting procedures eqs. (4), (6) and (7) can be implemented with a different order of accuracy, which depends on the numbers J_i and J_q of the time levels. The results presented in this paper are obtained using a third-order scheme with $J_i = J_q = 3$ and the coefficients $\gamma_0 = 11/6, \alpha_0 = 3, \alpha_1 = -3/2, \alpha_2 = 1/3, \beta_0 = 3, \beta_1 = -3, \beta_2 = 1$.

The splitting algorithm consists of an explicit advection step (4), a global pressure adjustment for incompressibility (5), (7), and an implicit viscous step (6). Semi-implicit schemes of this type have much less severe restrictions on the time step, $\Delta t = O(\Delta x)$ than do explicit schemes, where $\Delta t = O(\Delta x^2)$. However, parallelization of such schemes is considerably more difficult as they result in *global* elliptic equations of two types: the (monotonic) Helmholtz equation (6) which can be rewritten in the form

$$\nabla^2 u - \tilde{\lambda}^2 u = f(\mathbf{x}), \tilde{\lambda}^2 = \gamma_0 Re/\Delta t, \tag{8}$$

and the Poisson equation (7)

$$\nabla^2 u = g(\mathbf{x}). \tag{9}$$

Equations (8) and (9) are solved repeatedly in each time step. The discretization procedure and parallelization of these equations are discussed below.

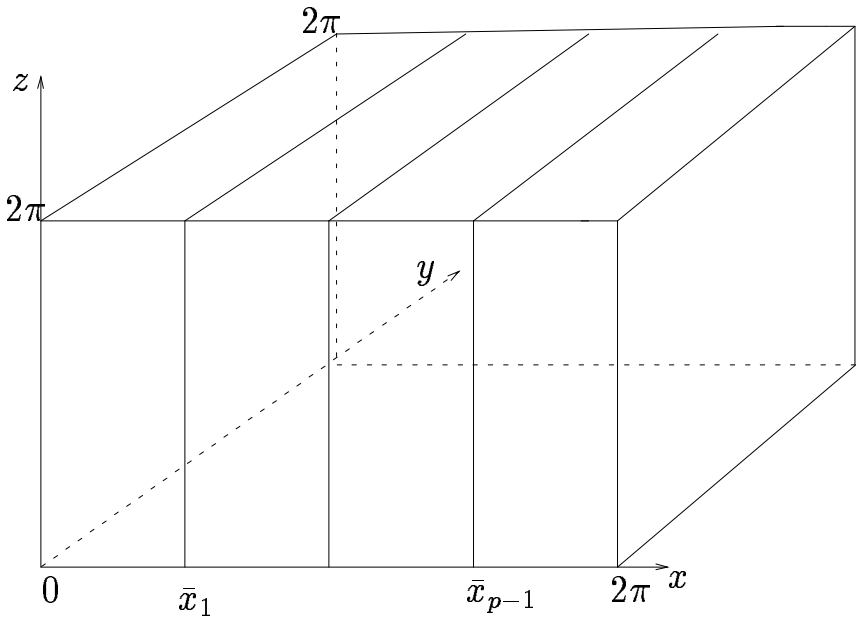

Figure 1: A 2-D domain ($z-x$ plane) and 3-D space ($x-y-z$) decomposed into parallel strips.

2.2 Discretization in Space: Domain Decomposition

Our parallel algorithm is based on domain decomposition technique. In this paper we consider the computational domain $\Omega = [0, 2\pi]^d$, where $d = 2$ or 3 is decomposed into parallel 2-D strips or 3-D slabs (which are called *strips* in the rest of the paper), see Figure 1. The algorithm consists of two main steps. First, we construct the local solutions independently in each subdomain (strip) with some prescribed boundary conditions at the interior boundaries (interfaces). Second, we correct these local solutions to remove discontinuities on the interfaces. This matching step involves global coupling of the interface data due to the the global nature of the elliptic problems (8) and (9).

In eariler work [1, 9, 10, 13] a parallel multidomain local Fourier (MDLF) method was developed (see the Appendix for a brief summary of this technique). It was shown the parallel solution and implementation of the Helmholtz equation can be performed using only local communication between neighboring domains. In this paper, Section 2.4, we show that the parallel solution of the Poisson equation (7) in a periodic domain is mostly composed of local problems since only the lowest harmonics of the pressure needd to be treated globally. Therefore, most of the communication that is needed for the parallelization of the Navier-Stokes equations (1) is mainly local, between adjacent subdomains.

In the following sections we describe briefly the basic MDLF technique for the Helmholtz equation and, in more detail, for the Poisson equation.

2.3 Solution of the Helmholtz Equation

In this section we describe the algorithm as applied to the Helmholtz equation (8). First, discretize the local problems on a uniform grid. After the application of the Fourier transform on the Helmholtz equation in the periodical directions y and z (see Figure 1), we get the collection of uncoupled 1-D equations. In the nth strip we have

$$\frac{d^2\hat{u}^{(n)}}{dx^2} - \lambda^2\hat{u} = \hat{f}^{(n)}(x). \tag{10}$$

where $\hat{u}^{(n)} = \hat{u}_k^{(n)}$, $\hat{f}^{(n)} = \hat{f}_k^{(n)}$, and $\lambda^2 = \tilde{\lambda}^2 + k^2$ in the 2-D case, and $\hat{u}^{(n)} = \hat{u}_{km}^{(n)}$, $\hat{f}^{(n)} = \hat{f}_{km}^{(n)}$, and $\lambda^2 = \tilde{\lambda}^2 + k^2 + m^2$ in the 3-D case. Here $k = -\frac{N_y}{2}, ..., \frac{N_y}{2}$ and $m = -\frac{N_z}{2}, ..., \frac{N_z}{2}$ are the wave numbers in the y and z directions. The MDLF algorithm of [9, 10, 13] for the solution of equations (10) consists of two main steps:

1. **Construction of the local particular solutions.** Eq. 10 is solved independently in each subdomain for $x \in [\bar{x}_{n-1}, \bar{x}_n]$ with some prescribed boundary conditions on the interfaces. In order to obtain smooth local pieces $\hat{f}^{(n)}(x)$ of the source function, we perform extension and "folding" across the boundaries $\bar{x}_{n-1}$ and $\bar{x}_n$ (for additional details see the Appendix).

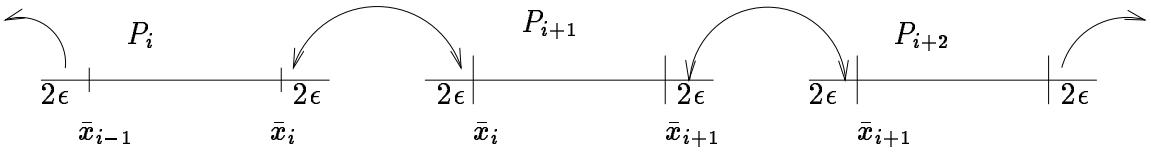

Figure 2: Data transfer between overlapped contiguous subdomains.

The extra pieces of the function $\hat{f}^{(n)}(x)$, required for smoothing, are provided by the overlapping of the neighboring subdomains over a small range 2ϵ. The data transfer between neighboring subdomains is demonstrated in Figure 2. The extended source function coincides with $\hat{f}^{(n)}(x)$ on the interval $x \in [\bar{x}_{n-1}, \bar{x}_n]$ and smoothly decays outwards over the distance 2ϵ from both sides of the interval.

The smooth source function in the right hand side can be represented by rapidly (in fact spectrally) converging trigonometric series. Thus, we integrate the local problems on extended intervals using the discrete cosine transform (DCT). The computational count for this step is $(n + N_\epsilon)\log(n + N_\epsilon)$, where n is the number of collocation points in a subdomain (across the strip) and N_ϵ is the number of points on the extended intervals ϵ. At the assumption that $N_\epsilon \ll n$, the "computational overhead" of the MDLF method, as compared to the conventional Fourier method, can be estimated as $(n + N_\epsilon)\log(n + N_\epsilon)/n\log n = O(N_\epsilon/n)$. This is the cost for having the spectral accuracy when applying the DCT transform to nonperiodic pieces of functions in the subdomains.

The values of the computed particular solutions $u_p^{(n)}$ have jumps on the interfaces and it is demonstrated in Figure 3.

This method preserves the spectral accuracy of the Fourier method and is very efficient for the evaluation of a particular solution in a subdomain.

2. Matching of the local solutions. In this step we correct the local solutions (independently for each harmonic k and m) in order to satisfy the continuity of the solution and its first derivative at the interfaces $x = \bar{x}_n$: $\hat{u}^{(n)} = \hat{u}^{(n+1)}$, $\frac{d}{dx}\hat{u}^{(n)} = \frac{d}{dx}\hat{u}^{(n+1)}$.

The matching procedure makes use of the properly weighted interface Green's functions which are the singular solutions of the homogeneous form of eq. (10). For each n, this is a pair of exponential functions

$$h_+^{(n)} = e^{\lambda(x-x_n)}, \quad h_-^{(n)} = e^{-\lambda(x-x_n)} \tag{11}$$

"attached" to the interface x_n and decaying away from $x = x_n$ on both side (see Figure 4). The smooth global solution is sought as a linear combination

$$\hat{u} = \bigcup_{n=1}^{P} \hat{u}^{(n)}, \quad \hat{u}^{(n)} = \hat{u}_p^{(n)} + \sum_{m=1}^{P} \left(A_m h_+^{(m)} + B_m h_-^{(m)} \right). \tag{12}$$

where P is the number of subdomains (strips).

The amplitudes A_n and B_n are computed locally, in each subdomain, in terms of the jumps of $\hat{u}_p^{(n)}$, $d\hat{u}_p^{(n)}/dx$ at the interfaces $x = \bar{x}_n$,

$$A_n = -\alpha_{n-1}, \qquad B_n = -\beta_{n-1} \tag{13}$$

where

$$\alpha_n = \frac{1}{2}\left(\frac{\delta_n'}{\lambda} + \delta_n\right), \quad \beta_n = \frac{1}{2}\left(\frac{\delta_n'}{\lambda} - \delta_n\right), \tag{14}$$

$$\delta_n = \hat{u}_p^{(n)}(x_{n+1}) - \hat{u}_p^{(n+1)}(x_{n+1}), \quad \delta_n' = \frac{\hat{u}_p^{(n)}}{dx}(x_{n+1}) - \frac{\hat{u}_p^{(n+1)}}{dx}(x_{n+1}),$$

$\hat{u}_p^{(n)}$ and $\frac{\hat{u}_p^{(n+1)}}{dx}$ are the interface values of the particular solution and its derivative.

The above matching procedure is *exact*. Indeed, by matching the function and its first derivative at the interfaces, the higher derivatives will be matched automatically due to eq. (10).

We observe that if $\lambda l \gg 1$, where $l = 2\pi/P$ is the size of a subdomain across the strip, then the exponential functions $h_\pm^{(n)} = e^{\pm\lambda|x-x_n|}$ decay rapidly away from the interface x_n so that they are well localized on the size of a subdomain. In effect, instead of (12) we arrive at the local matching relation

$$\hat{u}^{(n)} = \hat{u}_p^{(n)} + A_{n+1} h_+^{(n+1)} + B_n h_-^{(n)}. \tag{15}$$

The accuracy of the local matching is estimated as $O(e^{-\lambda l})$. The computational count of this step is $O(n)$.

In the case of the monotonic Helmholtz equation,

$$\lambda^2 = \tilde{\lambda}^2 + k^2 + m^2, \qquad \tilde{\lambda}^2 \propto Re\Delta t^{-1}. \tag{16}$$

the condition of locality $\lambda l \gg 1$ holds even for $k = m = 0$ provided that $Re/\Delta t$ is large enough. In such a case the local matching procedure (12) can be used, which requires only neighbor-to-neighbor communication.

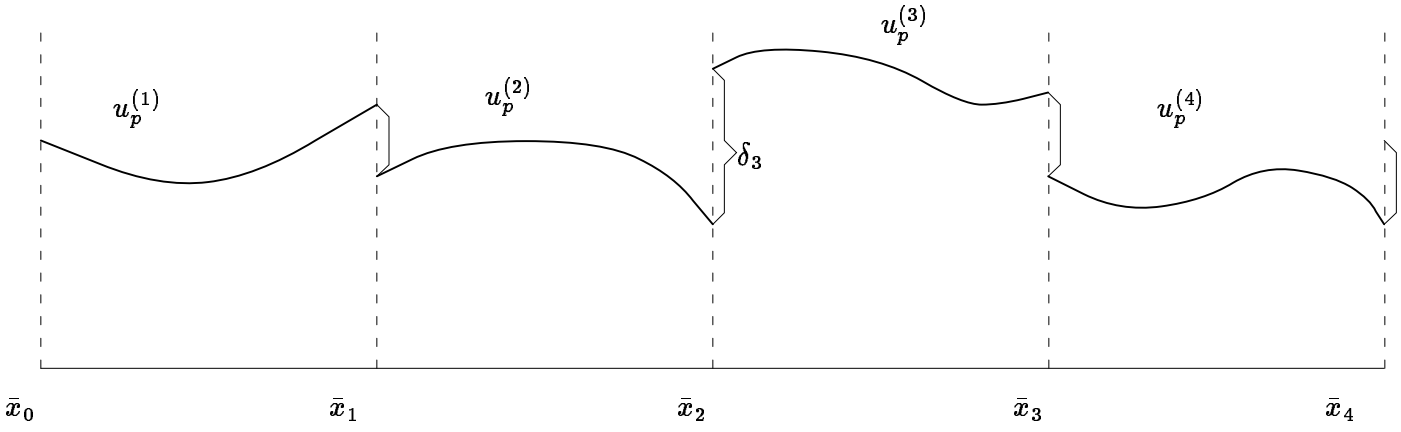

Figure 3: The jumps between the solution in each subdomain.

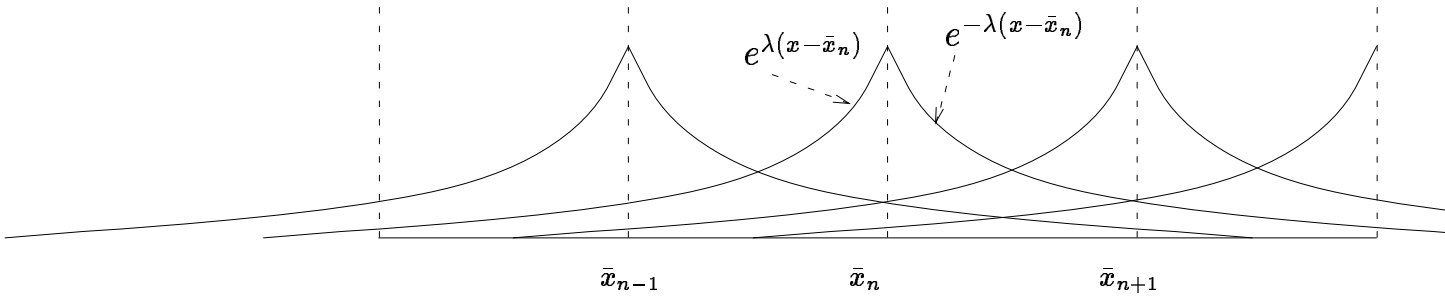

Figure 4: Decaying Green's functions on the interfaces.

To conclude, the performance of the MDLF method with the local matching of elemental solutions in subdomains is almost as good as that of the conventional Fourier method viewing accuracy, computational count and memory requirements. From the parallel performance point of view, this method is also advantageous as it requires only local interaction between neighboring subdiomains. Thus, the corresponding parallel algorithm is scalable.

2.4 Solution of the Poisson Equation

The solutions of the Poisson equation (9) that describe equilibrium processes are determined, in principle, at each location by the combined effect of all nonhomogeneous forcing terms and boundary conditions. One may conclude, therefore, that these solutions must be obtained by global procedures requiring repeated global communications. The homogeneous solutions

of eq. (9) decay slowly as a function of the distance, so that the contribution of all remote interfaces seems to be necessary for the computation of the matching parameters in each location. Nevertheless, even in this "tough" case, a substantial reduction can be achieved in the magnitude of the interprocessor communication while preserving the desired accuracy.

The Fourier transform of eq. (9) along the strips results in equation (10) where $\lambda^2 = k^2$ in the 2-D case ($\lambda^2 = k^2 + m^2$ in the 3-D case); k and m are wave numbers in the y and the z directions. The analysis below is related to the 2-D case and the 3-D case can be treated similarly.

For $\lambda \neq 0$, the homogeneous solutions of (10) decay exponentially as a function of x. The rate of decay becomes faster as k increases. Thus, a global matching procedure is required only for the long waves $\lambda \leq k_*$, whereas the short waves $\lambda \geq k^*$ can be treated by using local matching on the interfaces. For the intermediate wavenumbers $k_* < \lambda < k^*$ the number of connected subdomains (processors) can be controlled with the prescribed accuracy.

Thus, the solution of the Poisson equation (9) can be obtained by the MDLF method using either global or local matching depending on the value of k.

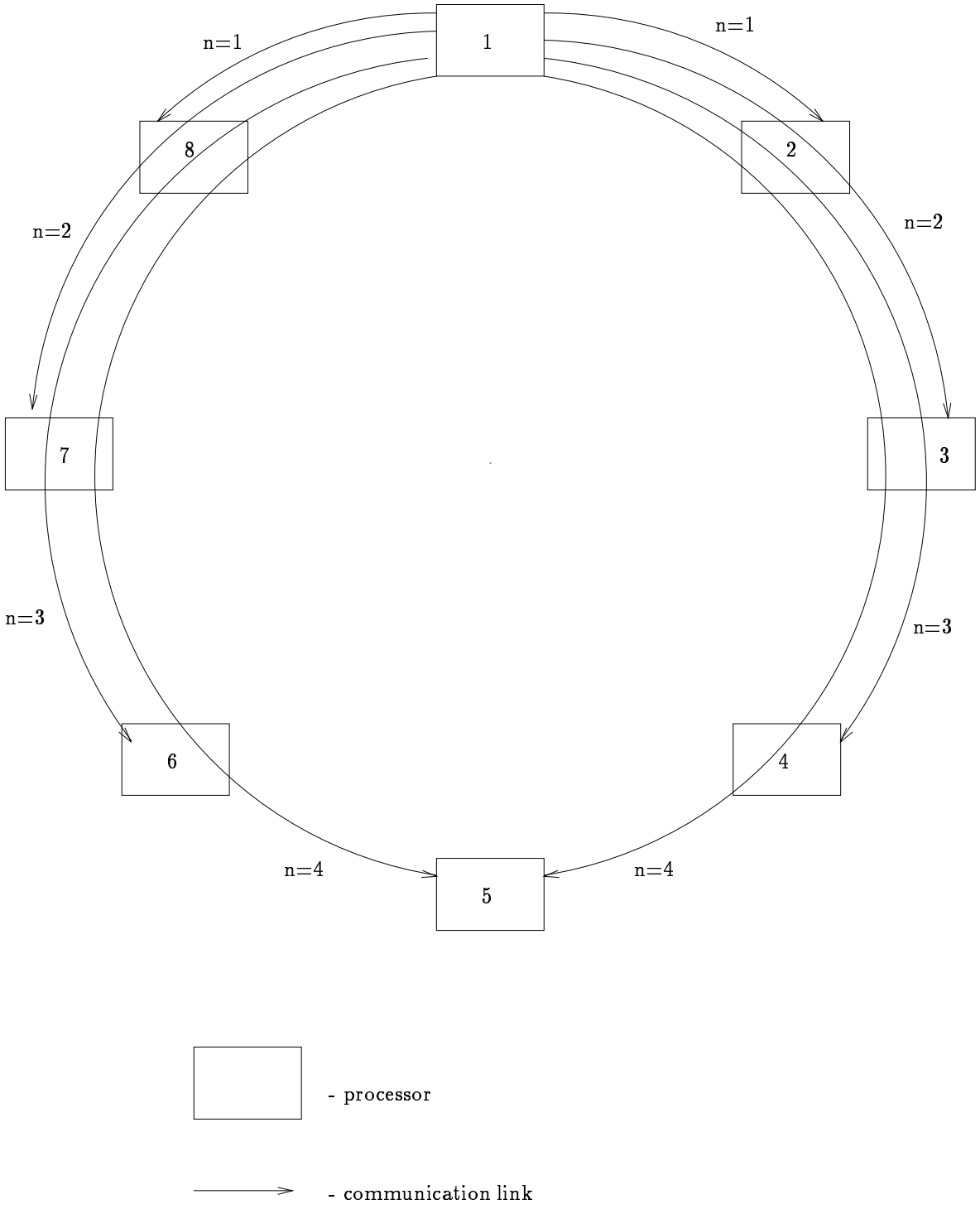

Figure 5: Different levels of communication for eight processors ($n = 1$ corresponds to local matching and $n = 4$ to global matching).

In order to evaluate the potential of this approach we present the following numerical test. We solve eq. (10) with the right-hand-side (RHS) corresponding to the solution $u_{ex} =$

Table 1: The numerical error in the solution of Poisson's equation in the cases of 1 to 4 connected subdomains; k is the wavenumber in the periodic direction and $P = 8$.

1	5.5 (-2)	3.2 (-2)	2.2 (-2)	3.5 (-12)
2	1.6 (-2)	3.9 (-3)	1.0 (-3)	1.4 (-12)
3	5.4 (-3)	5.4 (-4)	5.8 (-5)	9.1 (-13)
4	1.8 (-3)	7.8 (-5)	3.6 (-6)	6.8 (-13)
5	5.7 (-4)	1.1 (-5)	2.3 (-7)	5.4 (-13)
6	1.8 (-4)	1.7 (-6)	1.5 (-8)	4.6 (-13)
7	6.0 (-5)	2.4 (-7)	1.0 (-9)	4.0 (-13)
8	1.9 (-5)	3.6 (-8)	6.8 (-11)	3.5 (-13)
9	6.3 (-6)	5.4 (-9)	4.6 (-12)	3.2 (-13)
10	2.1 (-6)	8.1 (-10)	3.2 (-13)	2.9 (-13)

$\cos ky \sum_{m=1}^{M} \cos(mx + \alpha_m)/m$. Table 1 presents the maximum relative error $\varepsilon_n = |u - u_{ex}|_{max}/u_{max}$ at $P = 8$ (P is the number of strips), $N_l = N_x/P = 64$, $M = 64$ and phases α_m are chosen randomly. The subscript n in ε_n indicates the number of connected subdomains from each side (forming clusters of $2n + 1$ connected subdomains (see Figure 5). Note that for $n = P/2$ all subdomains are connected.

If we are satisfied, for example, with the accuracy $\varepsilon \leq 10^{-5}$, then global matching is needed only for $k = 1, 2, 3$; for $k = 4, 5$ we can restrict the communication to $n = 3$ neighboring processors; for $k = 6, 7, 8$ the communication can be reduced to $n = 2$; and, at last, for all $k \geq k^* = 9$ local communication ensures the desired accuracy. If the number of collocation points in the y direction is $N_y = N_x = 64 \times 8 = 512$, then the percentage $\eta = 2k^*/N_y \cdot 100\%$ of eq. (10), which is treated locally, amounts to more than 96% (here we take into account that each harmonic k is represented by a couple of Fourier coefficients). The results in Table 1 depend only slightly on the particular form of the source function $g(x)$ in (10).

The decomposition into "local" and "global" waves depends on the number of subdomains P, which determines the local length $l = 2\pi/P$, and thus affects the factor of locality $\mathcal{L} = e^{-kl}$. We now estimate the dependence of η on the number of processors P for a given accuracy ε. The processors are assumed to have a fixed memory size, so that the total number of collocation points in each subdomain $N_s = (N_x/P) \times N_y$ is independent of P. We define γ by $N_y/N_x = P^\gamma$. Then the following relations hold:

$$N_s = \frac{N_x}{P} \cdot P^\gamma N_x = N_x^2 P^{\gamma-1},$$

$$N_x \propto P^{\frac{1-\gamma}{2}}, \quad N_y \propto P^{\frac{1+\gamma}{2}}.$$

In order to keep the locality parameter $\mathcal{L}$ constant (to ensure the prescribed accuracy ε), the cutoff wavenumber k^* must be proportional to P, so that

$$\eta \propto k^*/N_y \propto P/P^{\frac{1+\gamma}{2}} = P^{\frac{1-\gamma}{2}}.$$

For $\gamma = 0$ we have $N_x = N_y \propto \sqrt{P}$ and $\eta \propto \sqrt{P}$ (e.g., if the number of processors $P = 128$, then according to Table 1, $\eta = 4\% \times \sqrt{128/8} = 16\%$). Another extreme case, $\gamma = 1$, corresponds to a constant N_x and $N_y \propto P$ (the resolution is changed only in the direction of the strips, as in the plane $x - z$ of Figure 1.) In this case η is independent of P; that is, the algorithm is fully scalable.

3 Computational Algorithm for the 2-D and 3-D Navier-Stokes Parallel Solver

In this section we present all the computational steps of the algorithm that was used to solve the 2-D and 3-D Navier-Stokes equations with strip topology. Each subdomain (strip) is mapped into a different processor. Initially, we define the velocity $\mathbf{U}_d(\mathbf{x}, t)$ and the pressure $P_d(\mathbf{x})$ distributions in space (see Section 5.1). Then we compute the "forcing" $\mathbf{F}$ using eq. (1). By using a time marching algorithm (eqs. (4)–(6)) we compute the velocity evolution $\mathbf{v}(\mathbf{x}, t)$ under the "forcing" $\mathbf{F}$ where the initial condition is $\mathbf{U}_d(\mathbf{x}, t)$. It is clear that in this case the deviation of the computed solution $\mathbf{v}(\mathbf{x}, t)$ from $\mathbf{U}_d(\mathbf{x}, t)$ in each time step has to be small and depends on the time-stepping increment Δt and the number of collocation points. (For example, in our 3-D computations the maximum relative error is $\epsilon \sim 10^{-10}$ for double precision at $\Delta t = 10^{-3}$ where the number of collocation points $n = 480, N_y = 32$, and $N_z = 32$. n is the number of points in a subdomain across the strip where N_x and N_z are number of points in the y and z periodic direction (along the strips), respectively; see Section 5.)

As mentioned previously, all harmonics from the solution of eqs. (4)–(6) in the y and z directions using strip topology are uncoupled. From now on, all the algorithmic work in the y and z directions will take place using specified harmonics except for the computation of the nonlinear terms $\mathbf{N}(\mathbf{v})$, which will be discussed later.

Our algorithm consists of eight main steps. After each step 2 to 6 we exchange data between the processors. However, only two data exchanges, before and after step 5, involve global data transfer (all processors transfer the data to one processor). All other data exchanges (communication) are local, that is they take place only between neighboring subdomains. In step 5, we match the particular solutions for the pressure Π and its derivative $\partial\Pi/\partial x$ in the whole domain. According to the previous analysis, only the lowest few harmonics have to be matched globally whereas higher harmonics are matched locally.

The time marching is performed in the Fourier domain independently for each harmonic k. The transformation to the physical domain is required only to compute the nonlinear term.

Step 1: Initialization

- Generation of the bell function for the RHS folding of eqs. (4)–(6) [9, 10] (see Appendix). It is needed for the construction of the elemental particular solutions of the Helmholtz equation in each subdomain (processor).

- Computation of the spatial part of the "exact" solution $\mathbf{U}_d(\mathbf{x}, t)$ in each subdomain (for details see Section 5.1).
- Calculation of the spatial parts of the forcing $\mathbf{F}$ components using eq. (1).
- Compute the initial velocity fields $\mathbf{v}^n$ for n=0,-1,-2 (correspondingly at $t = 0, -\Delta t, -2\Delta t$) as required by the $3d$ order time discretization scheme (4).
- Transform $\mathbf{v}^n$ into the Fourier space by applying the FFT in the periodic directions.
- Compute the nonlinear term $\mathbf{N}(\mathbf{v}^n)$, $n = 0, -1, -2$ according to eq. (3) using the pseudo-spectral technique (compute derivatives in the Fourier domain and multiply functions in the physical domain). This is performed in the following way: apply the inverse Fourier transform on $\nabla \times \mathbf{v}$ and $\mathbf{v}$. In physical space we compute $\mathbf{N}(\mathbf{v}) = \mathbf{v} \times (\nabla \times \mathbf{v})$ and return to Fourier space.

Step 2: Begin the time marching: Explicit advection step :

- Compute the velocity $\hat{\mathbf{v}}$ according to eq. (4).

Step 3: Compute the RHS of the Poisson equation :

Compute the function $\frac{1}{\Delta t}\nabla \cdot \hat{\mathbf{v}}$ in the RHS of (7). In order to compute the derivative $\partial\hat{u}/\partial x$ across the strips we have to extend and "fold" the velocity $\hat{u}$. Extension is provided by the overlapping of adjacent subdomains. Thus, the data exchange is taking place between neighboring processors (see Appendix).

- Computation of the derivatives $\partial\hat{u}/\partial x$, $\partial\hat{v}/\partial y$ and $\partial\hat{w}/\partial z$. To calculate $\partial\hat{u}/\partial x$ we extend and "fold" the velocity component $\hat{u}$ for the application of the Fourier transform in the x direction.
- Calculation of the RHS of the Poisson equation.

Step 4: Solve the Poisson equation :

We solve the Poisson equation (7) and compute the pressure Π^{n+1} and its derivatives for the calculations of the RHS in eq. (5). To accomplish this we have to exchange the overlapped areas of the RHS of eq. (7) in the neighboring subdomains.

- Extend the RHS function in each subdomain. In order to do this, exchange the data from the overlapped (neighboring) subdomains.
- Compute the particular solution Π_p^{n+1} of the Poisson equation by the MDLF method: fold the RHS, apply the Discrete Cosine Transform (DCT) on the extended interval and integrate in the Fourier domain).
- Compute the first derivative $\partial\Pi_p^{n+1}/\partial x$ by the MDLF method.

Step 5: Matching of Π_p^{n+1} and $\partial\Pi_p^{n+1}/\partial x$ at the interfaces (local-global matching)

This describes the partial global matching for the particular solutions of the Poisson equation. Globally matched harmonics are processed in one processor after all the data have been channeled to it. For locally matched harmonics we need data exchange between neighboring subdomains.

- Exchange the boundary values of Π_p^{n+1} and $\partial\Pi_p^{n+1}/\partial x$ between adjacent subdomains (processors). We compute the coefficients A_n and B_n, using eq. (13).
- *Global matching ($k < k_*$):* Exchange the computed values A_n, B_n between the processors so that each processor has the complete set of the coefficients. Correct the particular solutions using Green's functions from all the interfaces as in (12).
- *Local matching ($k > k_*$):* Correct the particular solutions using only local Green's functions as in (15).

Step 6: Compute the velocity $\hat{\hat{\mathbf{v}}}$

- Compute the functions Π^{n+1} and $\partial\Pi^{n+1}/\partial x$ in every subdomain, using eq. (12).
- Compute derivatives of Π^{n+1} in the periodic directions, $\partial\Pi^{n+1}/\partial y$ and $\partial\Pi^{n+1}/\partial z$.
- Evaluate $\hat{\hat{\mathbf{v}}}$ using eq. (5)
- Evaluate the RHS of eq. (6).

Step 7: Implicit viscous step

Solve the Helmholtz equation (6) for $\mathbf{v}_p^{n+1}$ using the MDLF method. This involves the following steps:

- Compute the source term in the RHS, $Re\hat{\hat{\mathbf{v}}}$
- Extend the RHS by exchanging data between the neighboring subdomains (processors).
- Fold the RHS function.
- Apply the DCT (across the strips) on the extended intervals and integrate in the Fourier domain.

Step 8: Matching step of the particluar solutions $\mathbf{v}_p^{n+1}$: Completion of the time step

To calculate the velocity $\mathbf{v}^{n+1}$ we match the particular solutions $\mathbf{v}_p^{n+1}$ and their derivatives $\partial\mathbf{v}_p^{n+1}/\partial x$. This requires the exchange of boundary values of these quantities between neighboring subdomains.

- Match the particular solutions $\mathbf{v}_p^{n+1}$ using local matching procedure as in step 5.
- Compute the derivatives $\partial\mathbf{v}_p^{n+1}/\partial x$ across the strips.

This completes one marching step. To continue marching in time, go to step 2.

The accuracy and stability of the above MDLF method were tested in our previous papers [9, 10, 13, 15]. In [2] this method was applied for long time numerical simulation of turbulent flows. In this paper we concentrate on the implementation of the MDLF method on several parallel machines and the performance analysis.

4 Implementation

The proposed algorithms are appropriate for scalable distributed-memory MIMD multiprocessors.

To implement the above algorithms we used the PVM[7] software package, which currently is one of the most popular software packages for parallel processing. Under PVM the user defines a collection of serial, parallel, and vector computers that appears as one large distributed-memory computer. We will refer to this computer as a *virtual machine* and the term *host* will be used to designate one of the member computers. The virtual machine is scalable so that it can grow with the size of a problem. PVM supplies the functions to automatically start up tasks on the virtual machine and allows tasks to communicate and synchronize with each other. A *task* is defined as a unit of computation in PVM that is analogous to a UNIX process.

Effective implementation of the code on a distributed-memory parallel computer requires the development of an efficient concurrent algorithm, which should minimizate communication costs and maintain a proper load balance. The parallel implementation strategy adopted here is based upon domain decomposition (DD) method. According to this method each domain is assigned to a specific processor.

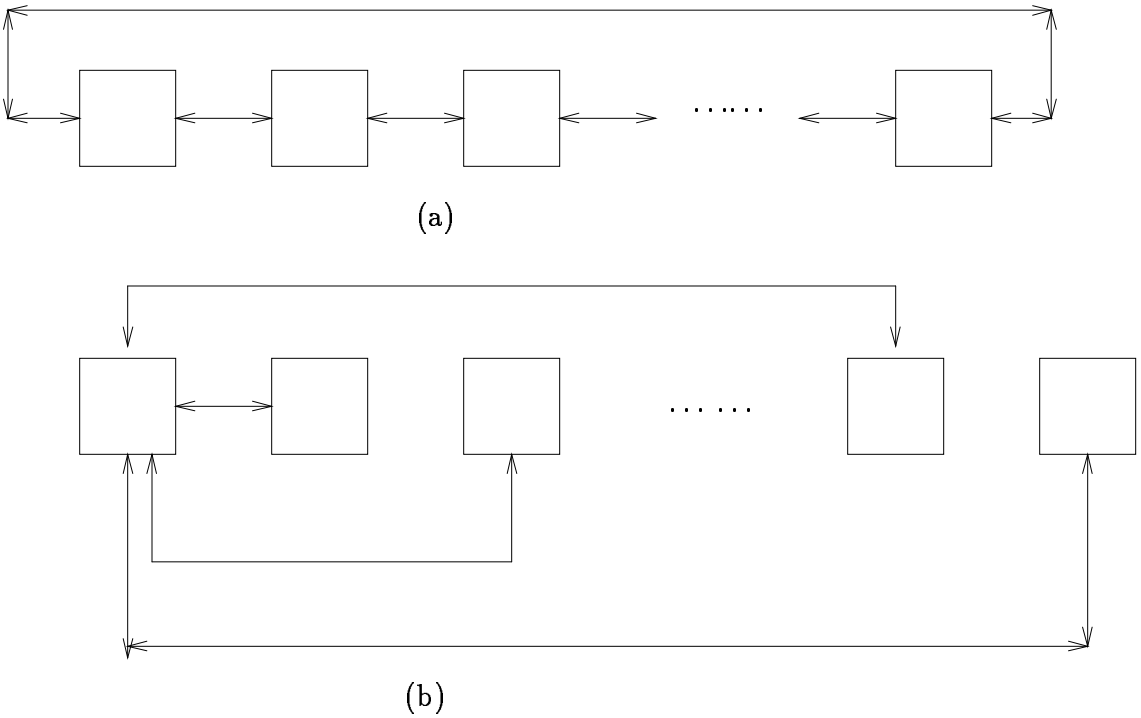

Figure 6: Two topologies for strip computation.

In our computations we used a multiprocessor topology, as described in Figure 6. During the computations there are several local data exchanges between neighboring processors, according to the parabolic domain decomposition (PDD) methodology (Figure 6(a)) and two partial global data exchanges for the matching of particular solutions for the Poisson equation (Figure 6(b)). The 2-D and 3-D Navier-Stokes solvers are implemented on three different MIMD message-passing multiprocessors: a 60-processor IBM SP2, a 20-processor MOSIX, and a network of 10 Alpha workstations using PVM software package. The same program without any modifications using PVM was running on all multiprocessors. The next section, 4.1, is a description of the three machines.

4.1 The Multiprocessors

The network of Alpha workstations (Alpha farm) used in our study is a distributed-memory farm of a DEC 3000 Model 400/400S AXP system. Each node in the farm is a high-performance desktop system that uses a Digital DECchip 21064 RISC microprocessor running at 133.33 MHz. The processor is based on Digital's Alpha AXP architecture and has about 75Mbytes of memory space available to the user. The Alpha farm consists of 10 computers connected through the Fiber Distributed Data Interface (FDDI). Effective data and message transfers through this interface are realized via the *Giga Switch*, with an average measured data transfer rate of 3-4 Mbytes

Due to memory limitation 75Mbytes is the upper limit for the size of the permitted number of collocation points in each node of the Alpha farm. For the computation

of the achieved speedup and efficiency, however, we need measurements of the serial algorithms taken on a number of collocation points N_x greater than what is available on the Alpha farm. In order to get a time estimate for N_x that is bigger than the available memory we extrapolate some results for the serial measurements.

MOSIX is an enhancement of UNIX that allows distributed-memory multicomputers, including a LAN-connected network of workstations (NOW), to share their resources by supporting preemptive process migration and dynamic load balancing among homogeneous subsets of nodes.

The NOW MOSIX is designed to run on configurations that include personal workstations, file servers and CPU servers that are connected by LANs, a shared bus, or fast interconnection networks. In these configurations each workstation (node) is an independent computer, with one or more (shared-memory) processors, local memory, and communication and I/O devices. Users interact with the multicomputer through their own workstations.

Parallel applications can be executed on MOSIX by simply creating many processes, just like a single-machine environment. The operating system automatically attempts to optimize the process allocations and balance the load.

The system image model of MOSIX is a NOW, in which all the user's processes seem to run at the user's "home" workstation. Each new process is created at the same site (or sites) as its parent process. All the processes of each user have the execution environment of the user's home workstation. Processes that migrate to other (remote) workstations use local resources if possible, but interact with the user's environment through the user's workstation.

The NOW MOSIX is compatible with BSDI's BSD/OS Release 2.0, which is based on Release 4.4BSD-Lite from the Computer Systems Research Group at UC Berkeley. The current implementation is for Intel's X86/Pentium-based workstations and PCs.

IBM SP2 has 64 processors, 62 of which are of the type Thin Node 2; each node includes 128 Mbytes memory and a 4.5-Gbyte internal disk. The two remaining processors are of type Thin Node 2, in which each node has 512 Mbytes of memory and 4.5-Gbyte internal disk. Each processor has a POWER2 architecture RS/6000 processor running at 66.7 MHz. The processors are interconnected with a multistage switch made up of bidirectional 4×4 crossbars. The SP2 is theoretically capable of 266 Mflops and it contains the AIX Windows Environment/6000.

5 Performance Analysis

We now present the performance analysis of our algorithm for the solution of the 2-D and 3-D Navier-Stokes equations (eq. (1)). The algorithm was tested according to the methodology described in Section 3. We identify the evolution of the solution under an external force that has complicated behavior in space and time. This algorithm can be easily adapted to the solution of other real problems of turbulence, such as turbulence decay [6] and the long-term

evolution of unstable exact solutions of Navier-Stokes equation [5]. Only the "forcing" term has to be modified for this application.
We use the following notation:

p	Number of strips (or processors)
n	Number of points in a subdomain across the strip
$N_x = np$	Global number of points in the x direction (across strips)
N_y	Number of points in the y (periodic) direction (along strips)
N_z	Number of points in the z (periodic) direction (along strips)
$2N_\epsilon$	Number of points in the overlapping region (see Figure 2; also Figure 13 in the Appendix)
$N_{LCT} = n + 2N_\epsilon$	Length of the local Fourier basis (LFB)
T_s	Serial time (for a single processor)
T_p	Parallel time (for p processors)
$S = T_s/T_p$	Speedup
$E = S/p$	Efficiency

5.1 The 2-D and 3-D Navier-Stokes Model Problems

To simulate the solution with "complex" behavior in space and time we take the model suggested by R. Temam [12] for the investigation of the efficiency of the multilevel nonlinear Galerkin method. The general methodology and the theoretical results in this presentation are valid for wide range of problems and they are independent of the specific example below.

Different initial distributions of velocity and pressure were used to test the 2-D and 3-D algorithms. The results presented in this section are obtained for highly nonlinear distributions, which enable us to simulate the complex behavior of flow in space and time.

The initial distributions of velocity and pressure for the simulations are given below.

For the 2-D Navier-Stokes equations we have

$$v_x = \frac{\partial \psi}{\partial y}, v_y = -\frac{\partial \psi}{\partial x},$$

$$\psi(x,y) = g(t) e^{(\cos kx)(\cos ky)}, \tag{17}$$

$$g(t) = \frac{1}{40}\left(\frac{1}{10} e^{(sin2t - 3sin2\pi t)} + cos(2\sqrt{2}t)\right) + \frac{0,375}{5},$$

$Re = 100, k = 5$. We take the external pressure distribution to be

$$\Pi_d = \cos(2kx)\cos ky. \tag{18}$$

Components of the "forcing" term $\mathbf{F}$ were derived from eq. (1) using the distributions of eqs. (17) and (18).

For the 3-D Navier-Stokes equations we have

$$\mathbf{v} = \nabla \times \mathbf{\Psi},$$

$$\Psi_x = g(t) \cdot e^{(\cos ky)(\cos kz)},$$
$$\Psi_y = g(t) \cdot e^{(\cos kx)(\cos kz)},$$
$$\Psi_z = g(t) \cdot e^{(\cos kx)(\cos ky)},$$

$$g(t) = \frac{1}{40}\left(\frac{1}{10}e^{(sin2t-3sin2\pi t)} + cos(2\sqrt{2}t)\right) + \frac{0,375}{5}, \tag{19}$$

$Re = 100, k = 5$. The pressure distribution is given by

$$\Pi_d = \cos(2kx)\cos ky \cos kz. \tag{20}$$

By using eqs. (19) and (20) we compute the components of the forcing term $\mathbf{F}$ in eq. (1).

All the results presented below were obtained with a single-precision (float) option. The timing T_s and T_p are measured for 10 iterations in time (time-marching steps). The accuracy of the measured time results is about 4% on the SP2, 6% on MOSIX, and 10% on the Alpha farm.

5.2 Serial Timing

To obtain a realistic performance analysis we need timing results of serial programs with very large numbers of points. Due to memory limitations there is an upper limit for the number of collocation points per processor. For a meaningful comparison of the achieved speedup we have to obtain timing results for very large serial programs that are the counterparts of large parallel programs. Therefore, we use extrapolation methods to estimate the timing of serial programs that do not fit into a single processor.

To compute and estimate the parallel speedup of the 2-D and 3-D Navier-Stokes solvers we have to get the timing of a serial algorithm on a single processor using the same global resolution $N_x \times N_y$. The serial algorithm is based on the pseudospectral Fourier method in space using global FFT in both directions. Both serial and parallel algorithms use the same time-stepping scheme. In tables 2 and 3 we present the best achievable serial solver for the 2-D and 3-D Navier-Stokes equations, respectively; the corresponding plots are shown in Figures 7 and 8.

Table 2: Serial timing (in seconds) for the 2-D Navier-Stokes solver: $N_y = 1024$; the number of time steps is 10.

N_x	Alpha	MOSIX	SP2
128	53.8	80.9	44.7
256	128.4	199.1	94.2
512	346.9	450.3	220.9
1024	763.9	1004.1	542.7
2048	1736.9	2638.3	1345.5
4096	—	6202.4	2747.1

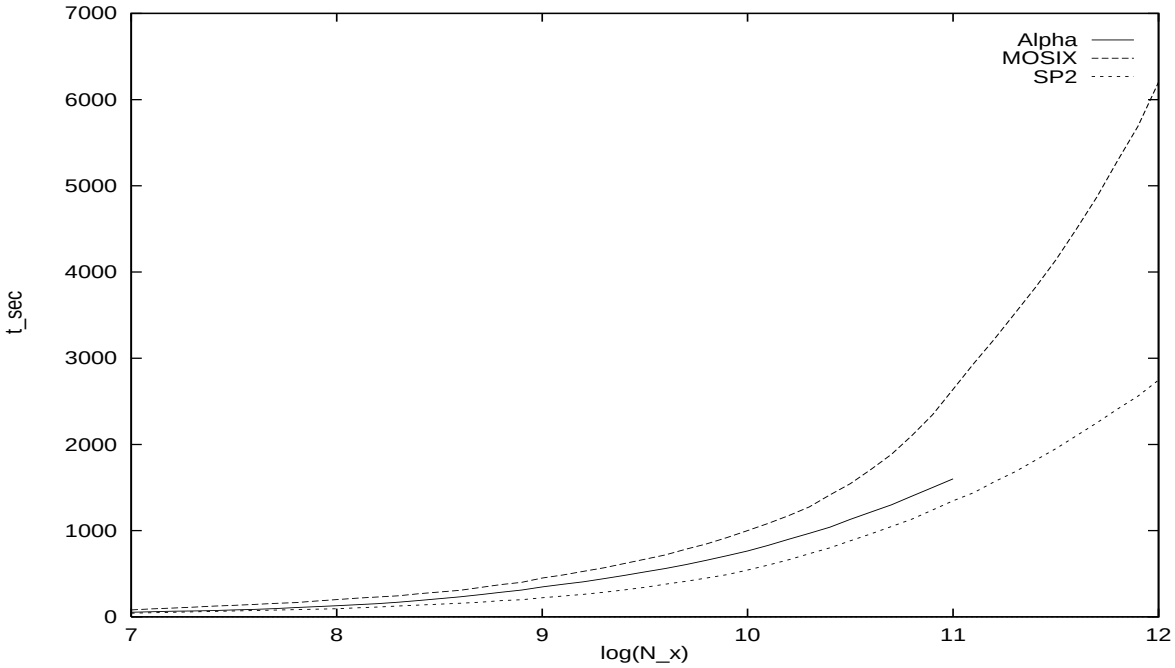

Figure 7: Serial timing for the 2-D Navier-Stokes solver using the pseudo-spectral Fourier method in space.

Table 3: Serial timing (in seconds) for the 3-D Navier-Stokes solver.

	$N_y = 32$ $N_z = 32$		
N_x	Alpha	MOSIX	SP2
128	125.3	141.6	43.6
256	272.3	384.7	84.4
512	553.1	788.1	172.5
1024	1229.3	1924.3	494.6
2048	2683.1	4079.5	1019.4
4096	—	—	2829.8

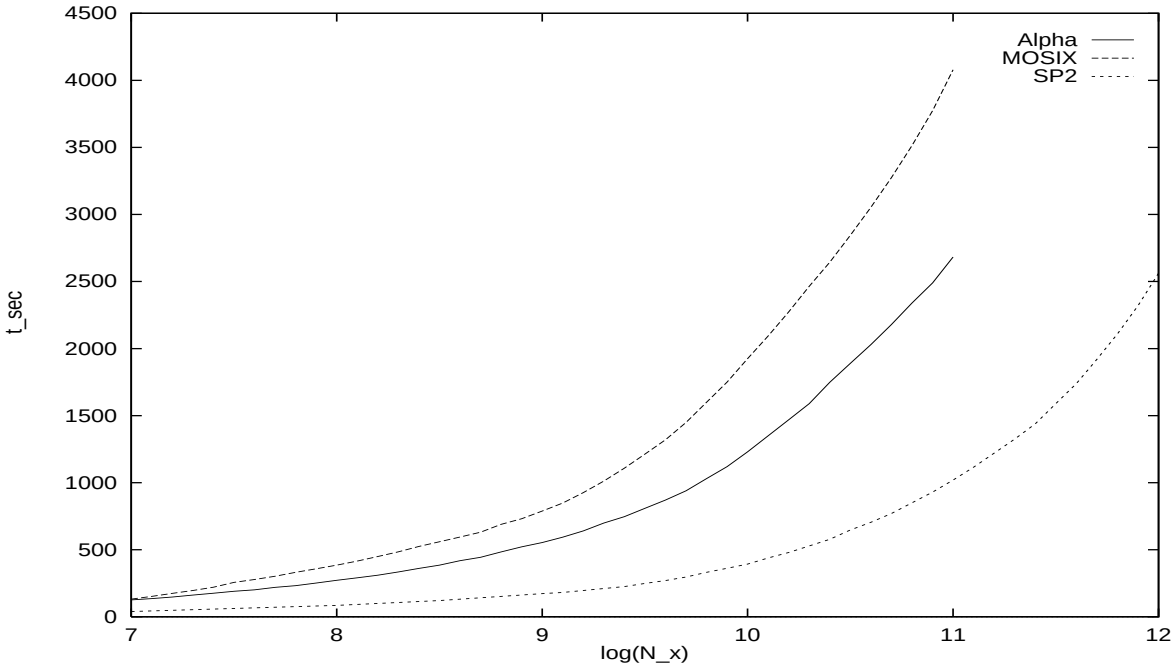

Figure 8: Serial timing for the 3-D Navier-Stokes solver using the pseudospectral Fourier method in space.

If the local resolution n (where n is the number of points in the subdomain across the strip) is fixed and the number of processors p grows, then the resolution $N_x \times N_y$, $N_x = np$, may become too large, so that the total memory required for the serial solver is beyond the capacity of a single processor. Nevertheless, we can estimate the serial timing by extrapolating the data in Table 2 for large N_x. In our serial algorithm the most time-consuming

Table 4: The values of $\tilde{\mu}_1$ and $\tilde{\mu}_2$ in eq. (22) for the 2-D ($N_y = 1024$) and 3-D ($N_y = N_z = 32$) cases, respectively.

	2-D			3-D		
	Alpha	MOSIX	SP2	Alpha	MOSIX	SP2
$\tilde{\mu}_1$	0.0631	0.1030	0.05330	0.0701	0.1742	0.0509
$\tilde{\mu}_2$	0.1010	0.0221	0.00701	0.5220	0.0121	0.0102

operation is the FFT. Other operations such as the copying of arrays or pointwise arithmetic operations have a computation count proportional to the number of grid nodes.

In order to evaluate the achieved speedup by the various algorithms serial timing has to be measured and analyzed. The dependence of the serial time T_s (Tables 2 and 3) on the number of points N_y (2-D case) and N_y and N_z (3-D case) is not smooth (not even monotonic). The reason is that we are using Temperton's FFT algorithm, which depends on how the number N_x is factorized into its primitive multipliers: $N = 2^i\ 3^j\ 5^k$ where i, j, k are integers. In other words, the performance of the algorithm is different for numbers that are powers of $2, 3$, or 5 only or that contain a mix of different multipliers. However, if the size of the FFT is based on one factor only, say $N = 2^i$, then the dependence of T_s on N is depicted by a smooth curve. For further estimation of the speedup, the serial timing for an arbitrary N (not necessarily based only on $2, 3$, and 5 separately) is required. We estimate it by the interpolation of known values, provided the dependence is smooth. Correspondingly, the serial time can be estimated as follows:

$$\begin{aligned} T_s &= \mu_1 N_x N_y (\log N_x + \log N_y) + \mu_2 N_x N_y \qquad \text{(2-D case)}, \\ T_s &= \mu_1 N_x N_y N_z (\log N_x + \log N_y + \log N_z) + \mu_2 N_x N_y N_z \qquad \text{(3-D case)}, \end{aligned} \tag{21}$$

where the first part is the time for performing the FFT and the second part is proportional to the resolution. Since in our implementations only N_x is changed and N_y is fixed for the 2-D case and N_y and N_z are fixed for the 3-D case, we can rewrite eq. 21 for both cases as

$$T_s = N_x(\tilde{\mu}_1 \log N_x + \tilde{\mu}_2). \tag{22}$$

The parameters $\tilde{\mu}_1$ and $\tilde{\mu}_2$ are evaluated by the least squares method. The values for each multiprocessor are for the 2-D and 3-D cases, and they are presented in Table 4 for $N_y = 1024$ and $N_y = N_x = 32$, respectively. Table 4 is a specific example but the fitting of the general data by eq. (22) approximates well all the other cases of the data.

The coefficients μ_1 and μ_2 (eq. (21)) in the "pure" FFT algorithm are independent of the total number of points $N_T = N_x N_y$ in the region. However, because of the presence of other factors in the program and also some particular features of the multiprocessors such as the size of the cache, this factor depends slightly on N_x and N_y. Since μ_1 and μ_2 increase monotonically with the numbers N_x and N_y, the expression in eq. (21) and the data in Tables 2 and 3 are used to estimate the minimal serial time through eq. (22).

This model used to extrapolate serial timings of the algorithm to larger problems which do not fit into the size of the available memory. Figures 9 and 10 show that they fit well the data and they are about 2-4% from the actual measurements that appeared in tables 2 and 3.

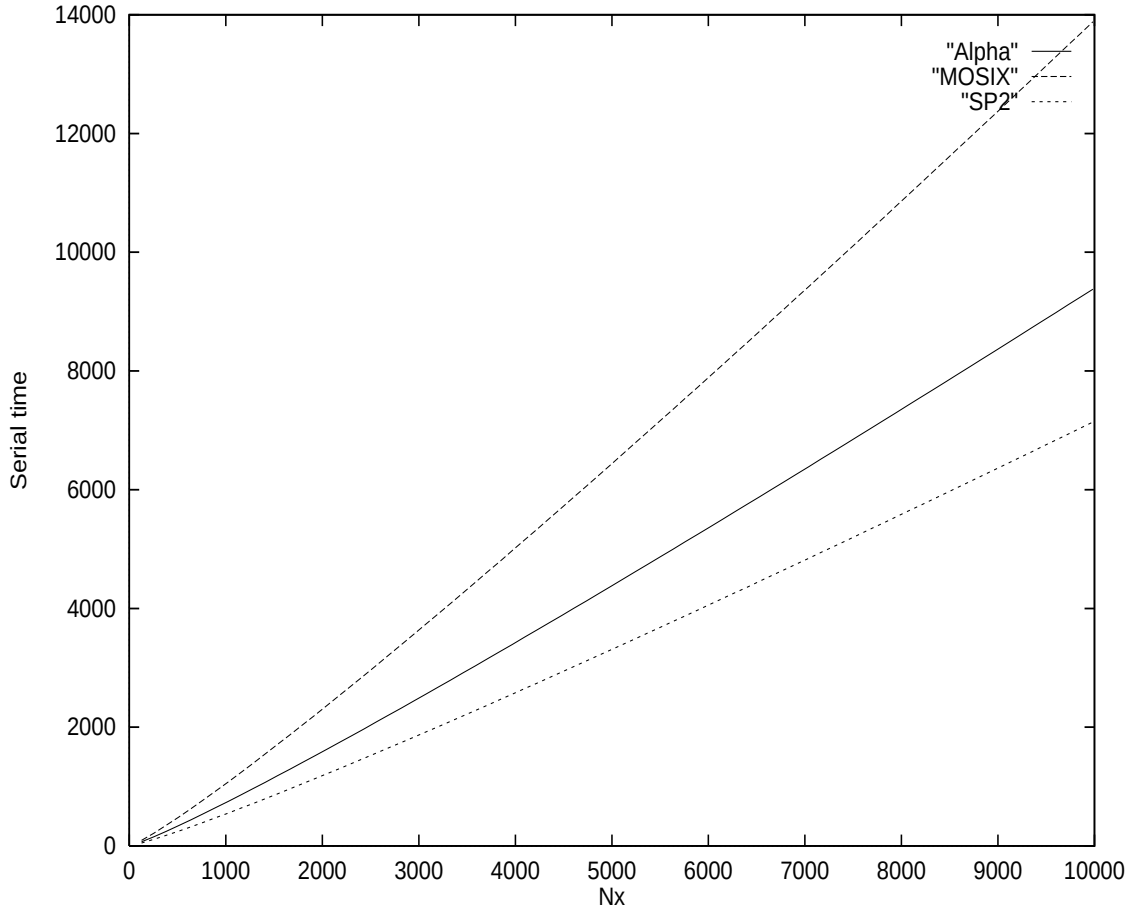

Figure 9: Plot of the 2-D serial time estimate based upon eq. (22). They fit well the numbers in table 2.

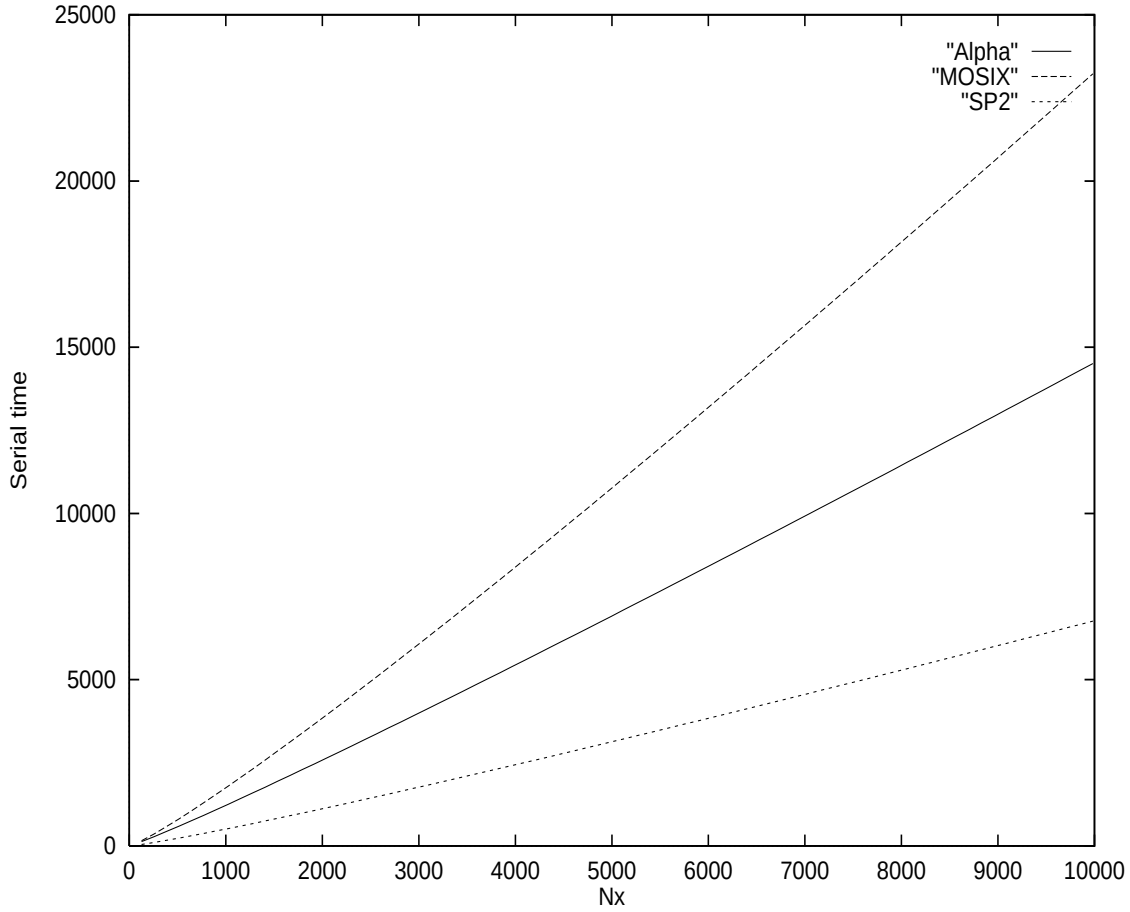

Figure 10: Plot of the 3-D serial time estimate based upon eq. (22). They fit well the numbers in table 3.

5.3 Parallel Performance

5.3.1 Parallel Timing

For the 2-D case $N_\epsilon = 12$ and for the 3-D case $N_\epsilon = 16$ where N_ϵ (see Figure 2 and the Appendix, Figure 13).

The parallel time for the 2-D problem is given in Table 5 for several resolutions n across the strips. The FFT is used in the periodic direction (along strips) and the DCT is applied (extended and periodized) to the local functions across the strips. The parallel time for the 3-D problem is given in Table 6

After completing the extension and folding of the MDLF (see Section 2.3, Appendix and references [9, 10]) we apply the Fourier transform to even functions in the subdomains. We can use the fact that the folding procedure, as explained in Section 2.3 and Appendix, is based on the symmetry of the bell functions. Thus, the DCT should be used rather than the FFT (as was done by Averbuch et al. [1]). In the 2-D case the time that the FFT and DCT use constitutes about 55% of the total parallel time on the Alpha farm, 53% on MOSIX, and 46% on the SP2. In the 3-D case the time that the FFT and DCT use constitutes about 50% to 70% used of the performance time. From now on, the DCT is used instead of the

Table 5: Two-dimensional parallel timing (in seconds) on three MIMD machines: $N_\epsilon = 12$, $N_y = 1024$; the number of time steps is 10.

	$n = 488$			$n = 1000$
p	Alpha	MOSIX	SP2	SP2
3	668.2	864.0	291.1	623.4
6	654.4	1028.6	290.5	638.9
10	708.1	1053.2	292.5	641.6
20	—	1129.3	293.1	651.1
30	—	—	294.2	646.3
40	—	—	293.6	669.4
50	—	—	294.4	679.6
60	—	—	298.6	677.4

Table 6: Three-dimensional parallel timing (in seconds) on three MIMD machines: $N_\epsilon = 16$, $N_y = N_z = 32$; the number of time steps is 10.

p	$n = 96$			$n = 224$			$n = 480$
	Alpha	MOSIX	SP2	Alpha	MOSIX	SP2	SP2
3	182.6	354.6	64.2	399.4	726.2	135.1	271.1
6	187.3	340.1	63.9	390.9	713.7	134.7	267.2
10	180.3	348.2	63.6	380.1	716.3	137.8	260.5
20	—	358.2	63.2	—	719.2	136.4	280.6
30	—	—	66.1	—	—	135.5	260.4
40	—	—	67.6	—	—	136.0	264.4
50	—	—	67.8	—	—	130.5	260.7
60	—	—	68.8	—	—	139.9	266.7

Table 7: Communication rates.

	Alpha	MOSIX	SP2
Communication rate (Mbit/s)	24	100	280
Latency (μs)	100	250	40

FFT in the pseudospectral phase of the algorithm. Therefore, in the following tables the total parallel time of the algorithm in each processor is measured with the assumption that the kernel of the pseudospectral computation, which is applied across the strips, is based on the DCT.

5.3.2 Communication Timing

The communication characteristics of the three multiprocessors are given in Table 7. Our results indicate that the total computation time strongly depends on the data transfer rate between processors. In each time step we are solving the Helmholtz and Poisson equations. The parallel Helmholtz solution necessitates only "local" transfer of data while the Poisson parallel solver dictates "global" transfer of data. Hence, there are two "types" of communication:

1. **Communication between neighboring processors (local).** In Section 2.4 and reference [1] we demonstrated that the solution of the Helmholtz equation is *local*, which means that only neighboring processors (according to the topology design) communicate and transfer data. In "local" communication $2N_\epsilon N_y$ data are transferred between neighboring processors. The transferred data is independent of n.

2. **Communication between all processors (global).** All the nodes send their data to a designated processor. Then they block on waiting for its return messages. In the "global" phase the boundary conditions on the interface are transferred.

   In Section 2.4 we saw that for 2-D and 3-D Poisson solvers only a few (the lowest) harmonics of the pressure in the directions along the strips have to be matched globally. The number of globally matched harmonics depends on the required accuracy of the solution. In this section we verify this observation on a highly nonlinear solution (eq. (19)) with complex dynamics in space and time. To investigate the advantage of using local matching of partial pressure harmonics we take another extreme case, in comparison with the results in Tables 10–11. For this extreme case, we take the maximum number of harmonics (the only constraint is the memory size) that are being matched along strips. The results of these computations are presented in Table 8. The notations are the same as in Section 2.4. All the results reflected local matching of partial pressure harmonics at distinct N_g to produce the same order of accuracy ($\epsilon \sim 10^{-4}$) as was achieved by using complete global matching of pressure harmonics. From this table we see that for the solution of eq. (19) we need only the three lowest harmonics and the mean value of the pressure to be matched globally. This results in about a 20% savings in the executable time in comparison with the complete (full version) of the global algorithm. We get almost the same savings when the number of

Table 8: Timing (in ticks) and speedup $S_{g/l} = T_g/T_{g/l}$ for parallel algorithms for 10 iterations: $n = 8,\ N_x = n \cdot p,\ N_y = N_z = 128,\ N_\epsilon = 4$.

p	N_g	$T^c_{g/l}$	$T_{g/l}$	T^c_g	T_g	$S_{g/l}$
10	4	600	681	694	792	1.16
4	32	568	660	621	726	1.10
6	32	546	637	632	726	1.14
8	32	559	652	680	762	1.17
10	32	562	651	694	792	1.21

Table 9: Two-dimensional problem: global communication time (in seconds) for 10 time steps. It is performed 20 times and is independent of n.

p	Global transfer (Mbytes)	Alpha	MOSIX	SP2
3	0.50	0.2	0.2	0.1
6	1.15	0.3	0.3	0.3
10	2.17	1.3	0.9	0.8
20	4.32	—	1.9	2.4
30	6.67	—	—	4.0
40	8.92	—	—	4.8
50	11.27	—	—	6.0
60	13.36	—	—	8.3

globally matched harmonics is increased (row 2 at $N_g = 32$ in Table 8). This is due to the fact that a packet size in the Alpha farm is 16Kbytes. Therefore, the transfer time of 4 and 32 harmonics is almost the same. From Table 8 we see that the algorithm with local matching of partial pressure harmonics is scalable. The execution time of the complete global (full version) algorithm increases with the number of processors. Global exchange of data in global algorithms can degrade the performance. Therefore, the speedup $S_{g/l}$ increases with machine size (see Table 8). Thus, we show that local matching of partial pressure harmonics leads to a reduction in executable time while preserving the accuracy of the solution

Actually, from Tables 10 and 12 we can see that the communication time varies from 2% to 4% of the overall computational time and it depends upon the problem size.

These results also indicate that the communication overhead related to "local" and "global" communication times is negligible. This is due to the fact that, according to the MDLF method, the time $T_c \sim N_y \cdot N_\epsilon$. Thus, as the local problem size increases, the communication/computation ratio decreases. Therefore, the amount of communication overhead decreases.

5.3.3 Speedup and Efficiency

From Tables 13-14 we will see that the achieved speedup grows almost linearly with an increase in the number of processors p. This linear growth is explained by the fact that the

Table 10: Two-dimensional problem: total (global + local) communication time (in seconds) for 10 time steps. Local transfer of bytes is 12Mbytes. It is performed 60 times and is independent of n.

p	Alpha	MOSIX	SP2
3	2.9	4.3	2.0
6	3.4	5.4	3.1
10	4.3	5.5	4.7
20	—	6.8	8.1
30	—	—	10.0
40	—	—	12.3
50	—	—	16.5
60	—	—	17.8

Table 11: Three-dimensional problem: global communication time (in seconds) for 10 time steps. It is performed 20 times and is independent of n.

p	Global	$N_y = 32,\ N_z = 32$		
	transfer(Mbytes)	Alpha	MOSIX	SP2
3	0.6	0.2	0.1	0.3
6	1.4	0.4	0.3	0.3
10	2.5	0.5	0.9	0.8
20	5.3	—	1.4	2.3
30	8.1	—	—	4.6
40	10.9	—	—	5.9
50	13.7	—	—	7.5
60	16.5	—	—	9.2

Table 12: Three-dimensional problem: total communication time (global + local) (in seconds). Local transfer of bytes is 19Mbytes. It is performed 60 times and is independent of n.

p	Alpha	MOSIX	SP2
3	3.3	6.0	2.2
6	5.0	7.9	2.6
10	4.9	8.5	3.7
20	—	10.4	6.4
30	—	—	9.9
40	—	—	12.7
50	—	—	16.3
60	—	—	17.3

Table 13: Two-dimensional problem: speedup and efficiency (in parentheses) of the 2-D parallel solver of the Navier-Stokes equations.

	$n = 488$			$n = 1000$
p	Alpha	MOSIX	SP2	SP2
3	1.5 (0.50)	1.7 (0.57)	2.1 (0.70)	2.2 (0.77)
6	4.1 (0.68)	4.2 (0.70)	4.7 (0.79)	5.0 (0.85)
10	6.8 (0.68)	7.3 (0.73)	8.3 (0.83)	9.2 (0.95)
20	—	16.2 (0.81)	19.1 (0.95)	20.1 (1.00)
30	—	—	32.2 (1.07)	33.5 (1.11)
40	—	—	43.8 (1.09)	46.1 (1.15)
50	—	—	57.8 (1.15)	60.2 (1.20)
60	—	—	69.0 (1.18)	73.8 (1.23)

algorithm is highly scalable.

The ability to maintain efficiency at a fixed value by simultaneously increasing the number of processors and the size of the problem is the main feature of a scalable parallel system. The scalability of a parallel system is a measure of its capacity to increase speedup in proportion to the number of processors. It reflects a parallel system's ability to use increasing processing resources effectively. Our goal is to demonstrate that the new proposed parallel solver for the 2-D and 3-D Navier-Stokes equations are scalable in the above sense. When the number of processors increases and, therefore, the size of the problem also increases, the parallel computation time stays almost the same, but the serial computation time increases in accordance with eq. (21).

Speedup and efficiency of the 2-D Navier-Stokes parallel solver are presented in Table 13. They are calculated using data from the parallel and serial timing in Tables 2 and 5 and the extrapolation formula 22 for serial timing at larger resolutions, N_x. Denote by N the size of the FFT. We note that for $p > 20$ the efficiency $E > 1$. The reason is the following. The main routine in the present serial algorithm, which is based on the spectral Fourier method, is the DFFT. The computation count (or the performance time) for the FFT is a nonlinear function of the resolution N, $T_{DFFT} \propto N \log N$. In this case the efficiency can be estimated roughly as follows:

$$E = \frac{T_s}{pT_p} \sim \frac{N \log N}{p(\frac{N}{p} \log \frac{N}{p} + \tau_{com})}$$

$$\sim \frac{\log N}{\log N - \log\ p + p\tau_{com}/N},$$

where τ_{com} is the total communication time. Evidently, for large enough N the last expression becomes larger than one. Thus, for a large enough resolution the gain in the computation count due to domain decomposition offsets the communication overheads and the loss due to additional operations inherent in the MDLF method (such as overlapping subdomains and smoothing operations).

It follows that for large resolutions it pays to use the domain decomposition strategy even for the serial computation in a single processor. If the MDLF method is used as a serial

Table 14: Three-dimensional problem: speedup and efficiency (in parentheses) of the 3-D parallel solver for the Navier-Stokes equations.

	$n = 224$			$n = 480$
p	Alpha	MOSIX	SP2	SP2
3	2.01 (0.67)	1.71 (0.57)	1.81 (0.60)	1.87 (0.63)
6	4.02 (0.69)	3.85 (0.64)	3.91 (0.65)	5.21 (0.79)
10	7.04 (0.70)	6.80 (0.68)	7.14 (0.71)	8.31 (0.83)
20	—	14.03 (0.70)	18.09 (0.90)	18.80 (0.94)
30	—	—	26.75 (0.89)	27.63 (0.92)
40	—	—	41.23 (1.03)	43.62 (1.09)
50	—	—	52.51 (1.05)	55.53 (1.11)
60	—	—	65.04 (1.08)	68.39 (1.14)

algorithm, the efficiency is always less than one:

$$E \sim \frac{N \log \frac{N}{p}}{p(\frac{N}{p} \log \frac{N}{p} + \tau_{com})}$$

$$\sim \frac{\log \frac{N}{p}}{\log \frac{N}{p} + \tau_{com}} < 1.$$

From Tables 13 and 14 we can see that for $p > 20$ the efficiency stays almost constant when we increase the size of the problem and the number of processors which indicates that the algorithm is highly scalable.

If we estimate the speedup from eq. (21) we see that the speedup is independent of N_y for both the serial and parallel computation times $\propto N_y \cdot \log N_y$. However, this equation does not take into account the communication time between subdomains, which is an essential part of T_{DCT}. Although the overall computation time depends very heavily on the communication time, when we increase the size of the problem, there is a favorable increase in the computation/communication ratio, which offset this dependence.

5.3.4 Stability and Numerical Accuracy

Figures 11 and 12 exhibit the dependence of the numerical error $\varepsilon = \max_{0<x,y,z<2\pi} |\mathbf{v} - \mathbf{v_{ex}}|$ on time t, where $\mathbf{v_{ex}}$ is the exact solution in eq. (19). These graphs are computed with parameters $p = 3$, $N_x = np = 480 \times 3 = 1240$, $N_y = 32$, $N_z = 32$, and $N_\epsilon = 16$. The time step is $\Delta t = 10^{-2}$ for Figure 11 and $\Delta t = 10^{-3}$ for Figure 12. These figures demonstrate that the implementation is stable. The reduction of the time step by an order of magnitude results in a 3-order decrease in error. This means that the main contribution to the error (for the values Δt under consideration) is due to the temporal discretization and thus further reductions in Δt will yield more accurate results.

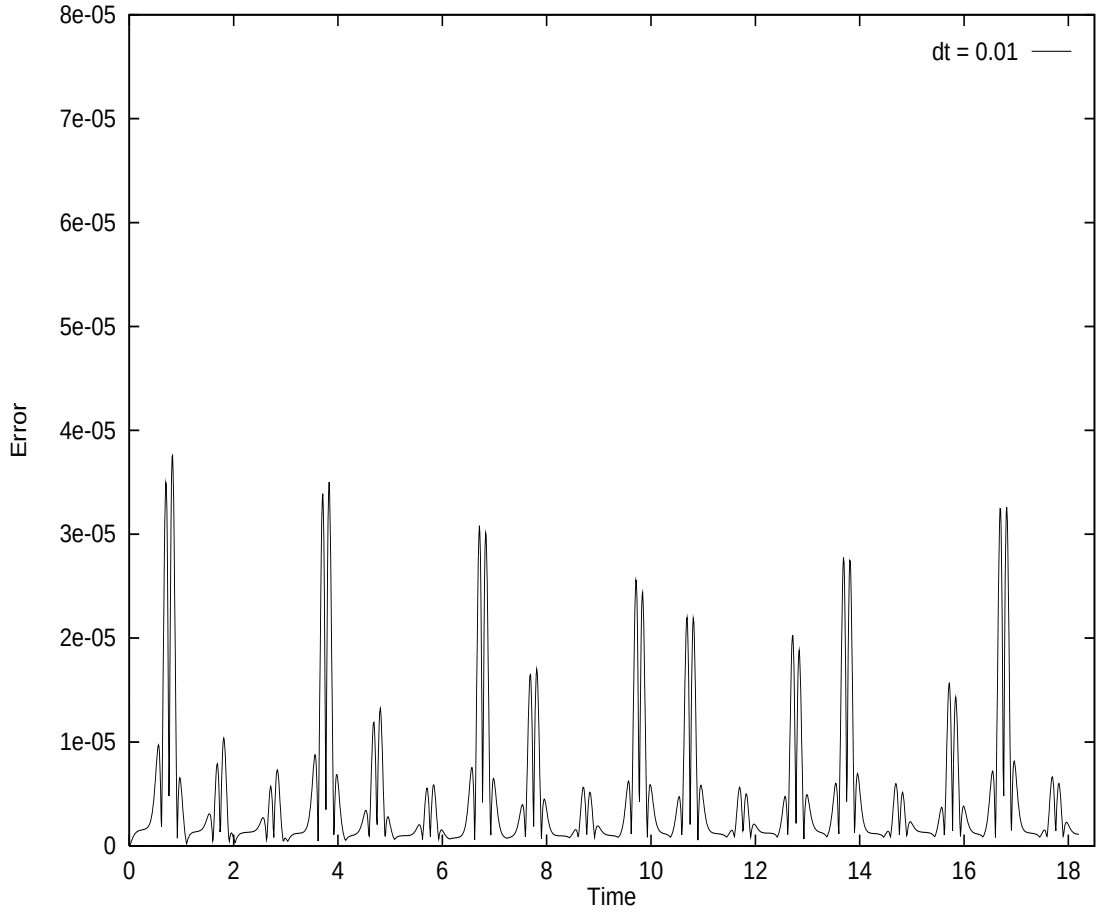

Figure 11: The dependence of the numerical error $\varepsilon = \max_{0<x,y,z<2\pi} |\mathbf{v} - \mathbf{v_{ex}}|$ on time t for 1800 time units.

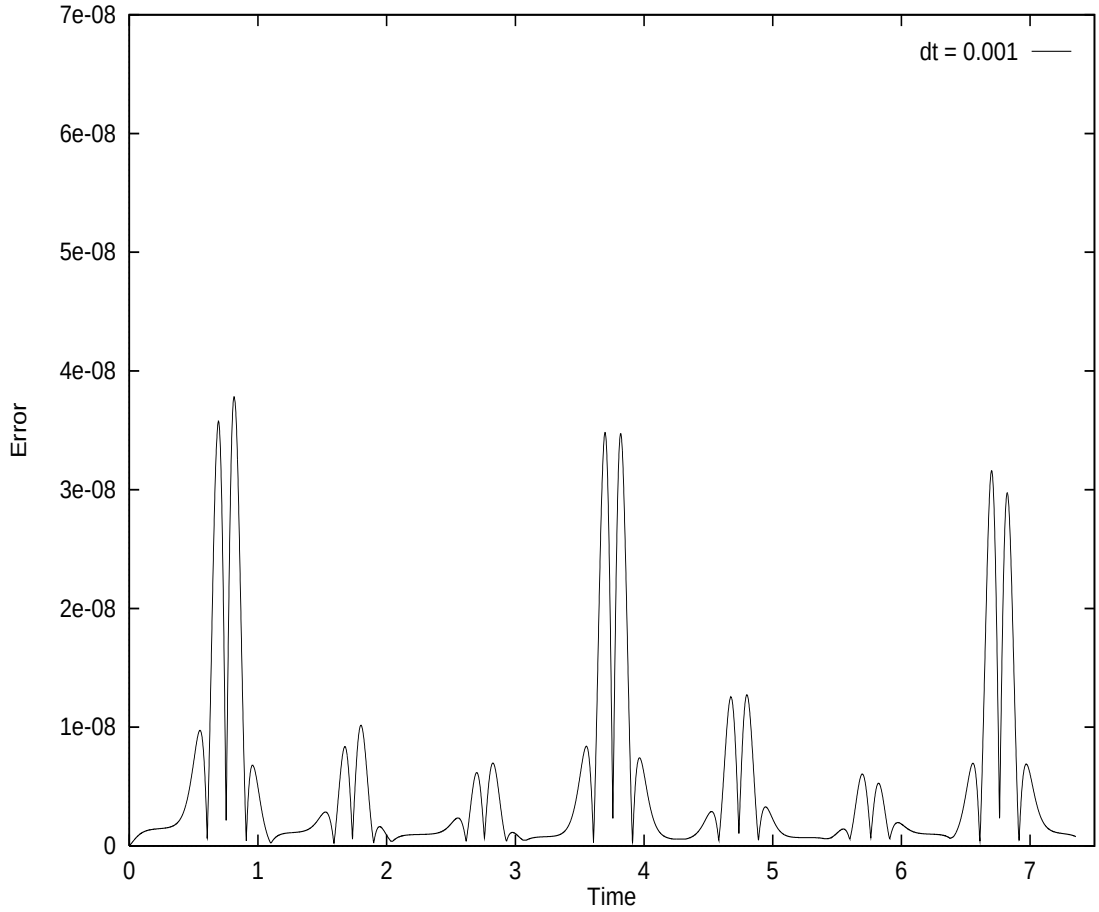

Figure 12: The dependence of the numerical error $\varepsilon = \max_{0<x,y,z<2\pi} |\mathbf{v} - \mathbf{v_{ex}}|$ on time t for 7500 time units.

5.4 Additional Tests on the SP2

The algorithm that was implemented with PVM was transferred to a team of researchers at IBM Science and Technology, Haifa, Israel, who are working in the area of debugging and performance monitoring for distributed parallel systems. The team's goals were to study and evaluate the tools for debugging and performance monitoring on the SP2 and to demonstrate and explore the capabilities and limitations of such tools on the Navier-Stokes 2-D and 3-D parallel solvers. The work reported by D. Zernik et al. [16] focuses on the IBM SP2 platform, but the results are applicable to other distributed parallel systems as well. The algorithm was used as a benchmark to compare MPI, the IBM version of PVM, and PVM and the performance of each communication library.

The application was coverted to MPI and evaluated [16]. The group used our algorithm as a "black box" and applied their monitoring and debugging tools to it and found no bottleneck in the algorithm or in the way it communicated with its different parts during the processing. The team concluded that MPI is more efficient than PVM and they obtained better performance with it.

6 Conclusion

In this paper we presented new parallel algorithms and their implementation on 2-D and 3-D Navier-Stokes equations on three different MIMD message-passing multiprocessors. We showed that our algorithm is highly scalable since most of the computation work requires only local communications between neighboring processors (only a few of the lowest pressure harmonics have to be matched globally). An almost linear speedup with high efficiency was achieved. The parallel software is PVM-based, which is machine independent. We also showed that the algorithm is stable. The implementation covers the case of domain, which is decomposed into 2-D strips and 3-D slabs. The more complicated and general case of domain decomposition into 3-D boxes is currently under development, with encouraging preliminary results.

Acknowledgments
M. Israeli was supported by the fund for the promotion of research at the Technion.
The authors wish to thank Dr L. Shtilman, from the Department of Fluid Mechanics and Heat Transfer, Faculty of Engineering, Tel Aviv University, for continuing support during all stages of this work. Only with his help and guidance with the extensive use of his Alpha farm were we able to successully complete our research, and to Prof. A. Barak, Department of Computer Science, Hebrew University, Jerusalem, who helped us to port the program into his MOSIX system.

References

1. A. Averbuch, M. Israeli, and L. Vozovoi. Parallel implementation of non-linear evolution problems using parabolic domain decomposition. Parallel Computing, 21,7: 1151-1183, 1995.

2. A. Averbuch, L. Ioffe, M. Israeli, L. Vozovoi. A parallel spectral Fourier - Nonlinear Galerkin algorithm for simulation of turbulence. Submitted to Numerical Methods in PDEs.

3. A. Barak, S. Guday, and R. Wheeler. The MOSIX Distributed Operating System, Load Balancing for UNIX. Lecture Notes in Computer Science, Vol. 672, Springer-Verlag, 1993.

4. A. Barak, O.Laden, and Z. Yarom. The NOW MOSIX and its Preemptive Process Migration Scheme. IEEE TCOS, 7,2: 5-11, 1995.

5. M. Brachet, D.I. Meiron, and S.A. Orzag et al. Small-scale structure of the Taylor-Green vortex. Journal of Fluid Mechanics, 130: 411-452, 1983.

6. G.L. Bronwing and H.O. Kreiss. Comparison of numerical methods for the calculation of two-dimensional turbulence. Mathematic of Computation, 52,186: 369-388, 1989.

7. A. Geist and A. Beguelin et al. PVM: parallel virtual machine. The MIT Press, Cambridge, Mass., 1994.

8. D. Gottlieb and R.S. Hirsh. Parallel pseudo-spectral domain decomposition techniques, Journal of Scientific Computing, 4: 309-325, 1989.

9. M. Israeli, L. Vozovoi, and A. Averbuch. Spectral multi-domain technique with Local Fourier Basis. Journal of Scientific Computing, 8,2: 181-195, 1993.

10. M. Israeli, L. Vozovoi, and A. Averbuch. Parallelizing Implicit algorithms for time - dependent problems by parabolic domain decomposition. Journal of Scientific Computing. 8,2: 197-212, 1993.

11. G.E. Karniadakis, M. Israeli, and S.A. Orszag. High order splitting methods for the incompressible Navier-Stokes Equations. Journal Computational Physics, 97,2: 414-443,1991.

12. R.Temam. Stability analysis of the nonlinear galerkin method. Mathematics of Computation, 57:196, 477-505, 1991.

13. L. Vozovoi, M. Israeli, and A. Averbuch. Spectral multidomain technique with Local Fourier Basis II: decomposition into cells. Journal of Scientific Computing, 9,3: 311-326, 1994.

14. L. Vozovoi, M. Israeli, and A. Averbuch. Multidomain local Fourier method for PDEs in complex geometries. Journal Computational and Applied Mathematics, 66: 543-555, 1996 .

15. L. Vozovoi, M. Israeli, and A. Averbuch. Multidomain Fourier algorithms for parallel solution of the Navier-Stokes equation. Contemporary Mathematics, 180: 539-546, 1994.

16. D. Zernik, A. Agbaria, and Z. Yehudai. Parallel programming tools, IBM, Haifa Research Lab, Technical Report 88.372, October 1996.

Appendix: Spectral Multidomain Technique with the LFB: Discretization in Space

Space discretization (splitting) is done by using the pseudospectral method that is based on the local Fourier basis (LFB). We illustrate the basic technique on the one-dimensional Helmholtz equation. We compute eq. (8) where f is a known function. The interval L is divided into S equal subdomains of length $l = L/S$. The algorithm consists of two main steps. First, we construct the particular solutions in each subdomain independently by applying arbitrary boundary conditions at the interfaces. For a space discretization we use the LFB. The LFB is particularly suitable for spatial discretization in DD methods because nonperiodic functions are represented in this basis by rapidly converging series. Matrices of differential operators in the transformed space are diagonal (in contrast with the full matrices for the Chebyshev or Legendre bases). Thus, in contrast to the ordinary Fourier basis, the LFB preserves the spectral accuracy when applied to (nonperiodic) local functions in subdomains. At the same time this basis possesses all the advantages of the usual trigonometric bases, such as a uniform mesh of collocation points, diagonal matrices for the differential operators in the transform space, and a fast transform algorithm. After this step the overall solution appears as a piecewise function with jumps on the interfaces. In the second step, we match all the local solutions by using the boundary Green's functions (the solutions of the corresponding homogeneous problem).

We consider the solution of eq. (8) in one of the subdomains $(a, b) \subset (0, L)$. The algorithm consists of two steps: (1) a smoothing procedure across the boundaries (interfaces) to localize the functions in a smooth manner by transforming the function to a periodic form and (2) the application of the spectral Fourier method with the locally supported trigonometric basis.

To implement smoothing, we introduce the bell function $B(x)$, supported on the extended interval $a_1 < a < b < b_1$:

$$\begin{array}{ll} B^2(x) + B^2(2\bar{a} - x) = 1 & x \in [a_1, a], \\ B(x) = 1 & x \in [a, b], \\ B^2(x) + B^2(2\bar{b} - x) = 1 & x \in [b, b_1], \\ B(x) = 0 & x < a_1,\ x > b_1, \end{array} \tag{23}$$

where $\bar{a} = (a + a_1)/2$, $\bar{b} = (b + b_1)/2$. This function is equal to $B = 1$ inside the subdomain and smoothly decays out words over the distance $2\epsilon = b_1 - b = a - a_1$. Some particular forms of $B(x)$ were tested by M. Israeli et al. [9].

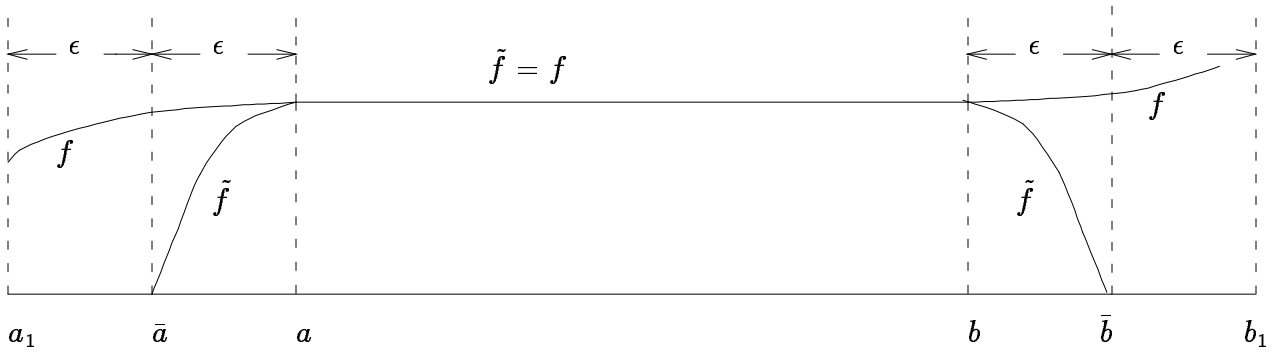

Figure 13: The folding operation.

The smoothing of the function f, denoted by $\tilde{f}$, appears as a "folding" across the lines $\bar{a}$ and $\bar{b}$ (see Figure 13):

$$\tilde{f} \equiv B \cdot f = \mathcal{F}_{\bar{a}}\mathcal{F}_{\bar{b}} f(x) = B(x)f(x) - B(2\bar{a} - x)f(2\bar{a} - x) - B(2\bar{b} - x)f(2\bar{b} - x) \qquad (24)$$

(the "folded" function $\tilde{f}$ is defined in $[\bar{a}, \bar{b}]$; the second term is "switched on" only in the interval $x \in [a_1, a]$ and the third term in the interval $x \in [b, b_1]$, respectively). The extra pieces of the function f, required for the smoothing operation, are provided by overlapping the neighboring subdomains over 4ϵ range. On the interval $x \in [a, b]$ we have $\tilde{f} = f$.

The smoothing procedure keeps the function $\tilde{f}$ highly continuous at $x = a, b$. In addition, eq. (24) yields that in the vicinity of the points $x = \bar{a}$, $x = \bar{b}$ the function $\tilde{f}(x)$ is odd and thus all even derivatives $\tilde{f}^{(2r)}(\bar{a}) = \tilde{f}^{(2r)}(\bar{b}) = 0$ for $r = 0, 1, \ldots$. After an antisymmetric reflection across the point $x = \bar{b}$ (or $x = \bar{a}$) is performed, we obtain a smooth periodic function, which can be represented by a rapidly converging Fourier series.