ELSEVIER

Comput. Methods Appl. Mech. Engrg. 184 (2000) 401–425

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

Parallel DNS algorithms on unstructured grids

Constantinos Evangelinos [a], Spencer J. Sherwin [b], George Em Karniadakis [a,*]

[a] *Division of Applied Mathematics, Center for Fluid Mechanics, Brown University, 182 George Street, Box F, Providence, RI 02912, USA*
[b] *Department of Aeronautics, Imperial College of Science, Technology & Medicine, Prince Consort Road, London, SW7 2BY, UK*

Abstract

With the prospect of Petaflop computing to be realized in the near future, we present two different parallel algorithms suitable for simulating turbulent flows in non-separable and multiply-connected computational domains. The algorithms are based on a new class of hierarchical spectral methods appropriate for tensor-product representations in hybrid subdomains, i.e., tetrahedra, hexahedra, prisms and pyramids. We review the numerical implementation of the spectral method and subsequently present two parallel paradigms, the first for a spectral element/Fourier algorithm, and the second for a fully 3D algorithm based on geometric domain decomposition. Emphasis is placed on the communication patterns of these algorithms in conjunction with the features of current or upcoming computer models. Examples of turbulent parallel simulations are included and limitations in currently achieving high parallel efficiencies are discussed. A perspective on the future of DNS on the emerging distributed shared memory (DSM) computer architectures is presented. © 2000 Elsevier Science S.A. All rights reserved.

1. Introduction

The direct numerical simulation (DNS) of turbulent flows was initiated in 1972 by Orszag and Patterson, who obtained accurate simulations of wind-tunnel flows at moderate Reynolds numbers [1]. This simulation was performed on the CDC 7600 with limited memory and only 50 Mflop/s peak speed, however it opened up the possibility of simulating turbulence from first principles without any ad hoc modeling, by directly solving the Navier–Stokes equations of fluid motion. In the last 25 years, the field has developed remarkably due to advances both in algorithms and computer hardware. In the last decade the peak performance of supercomputers sky-rocketed by a factor of 500, compared to only a factor of 10–15 from 1977 to 1987. At the same time, we have moved from the classical Fourier algorithms used in the first simulation of homogeneous turbulence to more sophisticated algorithms involving spectral element methods on unstructured grids handling computational domains of arbitrary geometric complexity.

A review of the state-of-the-art in DNS was presented in 1993 in [2] along with a specific proposal for future developments in DNS: The design of a parallel prototype computer (PPC), a hybrid distributed/shared memory computer of 1000 processors achieving a speed of 1 Teraflop for a load balanced turbulence simulations of 1024^3 resolution. Today, the main concept of that proposal has been realized as part of the developments of the Advanced Strategic Computing Initiative supported by the Department of Energy. Moreover, most of the newer architectural models such as the HP/Convex Exemplar, the SGI Origin 2000 or the new IBM SP have features similar to the PPC, thus providing high efficiency in solving the Navier–Stokes equations. With target simulations of 1 billion grid points on 100 Teraflop computer systems

[*] Corresponding author. Fax: + 1-401-863-3369.
E-mail addresses: gk@cfm.brown.edu (G.E. Karniadakis), s.sherwin@ic.ac.uk (S.J. Sherwin).

achievable in the next couple of years, the current focus has turned on the possibilities of simulations on the next generation of systems corresponding to once unthinkable Petaflop (10^{15} flops) rate systems.

Despite such great developments on the computer hardware side, the progress on the physical modeling side has been limited both in terms of the Reynolds number range as well as the physical or geometric complexity that can be simulated. As regards to the Reynolds number, an increase by a factor of 2 would approximately require one order of magnitude increase in CPU resources [2]. The introduction of non-isothermal processes, chemical reactions or complex geometric domains in the simulation tax the computational resources in a similar manner. The computational complexity is ultimately related to the numerical algorithms used to discretize the continuous equations as well as the parallel paradigm employed. To this end, the majority of the turbulence simulations today has been performed using spectral methods either of Fourier type, as in homogenous turbulence, or of a polynomial type as in shear turbulent flows [3–6]. The parallel paradigm can vary depending on the physical situation modeled but it is typically based on domain decomposition for the most complex cases.

A review of other numerical methods for parallel simulation of fluid flows including turbulent flows has been given in [2,7]. Here, we will concentrate on two prototype cases that are used in simulating turbulence in non-separable and multiply-connected domains: The first one corresponds to physical situations where one of the Cartesian directions is homogeneous and thus Fourier expansions can be employed along that direction while the other two Cartesian directions are inhomogeneous and thus general spectral methods (or finite differences [8,9]) are applied. The second case involves computational domains where all three Cartesian directions are non-homogeneous. Specifically, we will focus on spectral/*hp* type methods which exhibit an intrinsic 'domain-decomposition'; this leads naturally to a geometry-based distribution of work amongst processors which permits a high degree of parallelism. The key computational kernels are scalar products, matrix–vector and matrix–matrix multiplies, while the communication patterns involve pairwise exchanges, global exchanges, global reductions and gathers as well as global synchronizations.

This paper is organized as follows: In the first part we briefly review the spectral method that we will use and the time integration algorithm of the Navier–Stokes equations as they are the basis of the parallel paradigms that we will use. We then proceed with the specific concerns in the spectral/Fourier solver and subsequently we discuss the second paradigm with a full unstructured solver. We analyze some of the generic communication patters used in both these solvers and conclude with a discussion.

2. Spectral/*hp* discretizations on unstructured and hybrid grids

The parallel code $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r$ that we employ in the two aforementioned parallel paradigms is based on a new spectral basis [10–12]. It is appropriate for unstructured meshes based on triangles or tetrahedra in two- and three-dimensions, respectively. In many simulations, however, involving complex-geometry domains or external flows it is more efficient to employ hybrid discretizations, i.e., discretizations using a combination of structured and unstructured subdomains. Such an approach combines the simplicity and convenience of structured domains with the geometric flexibility of an unstructured discretization. In two-dimensions, hybrid discretization simply implies the use of triangular and rectangular subdomains, however in three-dimensions the hybrid strategy is more complex requiring the use of hexahedra, prisms, pyramids and tetrahedra.

Hexahedral domains have been used quite extensively in the *hp* finite element field [13,14]. More recently an unstructured *hp* finite element approach, based upon theoretical work in two-dimensions by Dubiner [15], has been developed for unsteady problems in fluid dynamics [11,16,17]. In the following, we will show how these expansions can be constructed using a unified approach which incorporates all the hybrid subdomains.

This unified approach generates polynomial expansions which can be expressed in terms of a *generalized* product of the form $\phi_{pqr}(x,y,z) = \phi^a_p(x)\phi^b_{pq}(y)\phi^c_{pqr}(z)$. Here we have used the Cartesian coordinates x, y and z but, in general, they can be any set of coordinates defining a specified region. The standard tensor product is simply a degenerate case of this product where the second and third functions are only dependent on one index. The primary motivation in developing an expansion of this form is computational efficiency. Such expansions can be evaluated in three-dimensions in $\mathrm{O}(N^4)$ operations as compared to $\mathrm{O}(N^6)$ operations necessary with non-tensor products based expansions.

2.1. Local coordinate systems

We start by defining a convenient set of local coordinates upon which we can construct the expansions. Moving away from the use of barycentric coordinates, which are typically applied to unstructured domains, we define a set of *collapsed Cartesian* coordinates in non-rectangular domains. These coordinates will form the foundation of the polynomial expansions. The advantage of this system is that every domain can be bounded by constant limits of the new local coordinates; accordingly operations such as integration and differentiation can be performed using standard 1D techniques.

The new coordinate systems are based upon the transformation of a triangular region to a rectangular domain (and vice versa) as shown in Fig. 1. The main effect of the transformation is to map the vertical lines in the rectangular domain (i.e., lines of constant η_1) onto lines radiating out of the point ($\xi_1 = -1,\ \xi_2 = 1$) in the triangular domain. The triangular region can now be described using the 'ray' coordinate (η_1) and the standard horizontal coordinate ($\xi_2 = \eta_2$). The triangular domain is therefore defined by $(-1 \leqslant \eta_1,\ \eta_2 \leqslant 1)$ rather than the Cartesian description $(-1 \leqslant \xi_1, \xi_2;\ \xi_1 + \xi_2 \leqslant 0)$ where the upper bound couples the two coordinates. The ray coordinate (η_1) is multi-valued at ($\xi_1 = -1,\ \xi_2 = 1$). Nevertheless, we note that the use of singular coordinate systems is very common arising in both cylindrical and spherical coordinate systems.

As illustrated in Fig. 2, the same transformation can be repeatedly applied to generate new coordinate systems in three-dimensions. Here, we start from the bi-unit hexahedral domain and apply the triangle to rectangle transformation in the vertical plane to generate a prismatic region. The transformation is then used in the second vertical plane to generate the pyramidal region. Finally, the rectangle to triangle transformation is applied to every square cross section parallel to the base of the pyramidal region to arrive at the tetrahedral domain.

By determining the hexahedral coordinates (η_1, η_2, η_3) in terms of the Cartesian coordinates of the tetrahedral region (ξ_1, ξ_2, ξ_3) we can generate a new coordinate system for the tetrahedron. This new system and the planes described by fixing the local coordinates are shown in Fig. 3. Also shown are the new systems for the intermediate domains which are generated in the same fashion. Here we have assumed that the local Cartesian coordinates for every domain are (ξ_1, ξ_2, ξ_3).

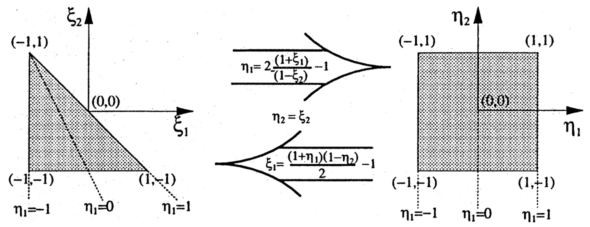


Fig. 1. Triangle to rectangle transformation.

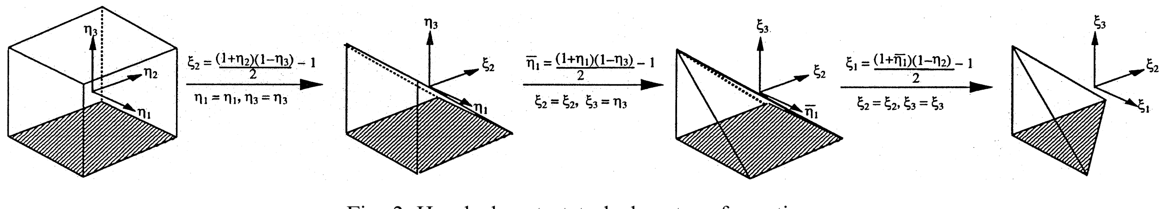
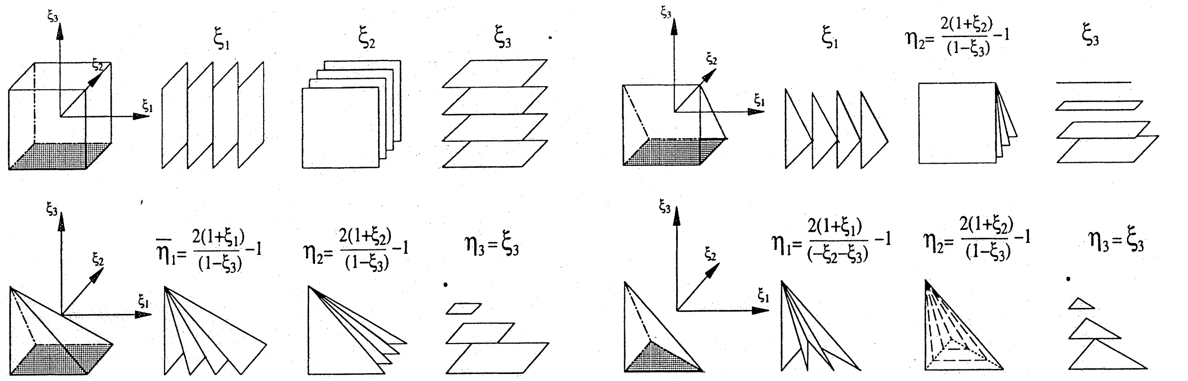

η_3
η_2
η_1
$\xi_2 = \frac{(1+\eta_2)(1-\eta_3)}{2} - 1$
$\eta_1 = \eta_1, \eta_3 = \eta_3$
η_3
ξ_2
η_1
$\bar{\eta}_1 = \frac{(1+\eta_1)(1-\eta_3)}{2} - 1$
$\xi_2 = \xi_2,\ \xi_3 = \eta_3$
ξ_3
ξ_2
$\bar{\eta}_1$
$\xi_1 = \frac{(1+\bar{\eta}_1)(1-\eta_2)}{2} - 1$
$\xi_2 = \xi_2, \xi_3 = \xi_3$
ξ_3
ξ_2
ξ_1

Fig. 2. Hexahedron to tetrahedron transformation.



ξ_3
ξ_2
ξ_1
ξ_1
ξ_2
ξ_3
$\eta_2=\frac{2(1+\xi_2)}{(1-\xi_3)}-1$
$\overline{\eta_1}=\frac{2(1+\xi_1)}{(1-\xi_3)}-1$
$\eta_3=\xi_3$
$\eta_1=\frac{2(1+\xi_1)}{(-\xi_2-\xi_3)}-1$

Fig. 3. The local coordinate systems used in each of the hybrid elements and the planes described by fixing each local coordinate.

2.2. Spectral hierarchical expansions

For each of the hybrid domains we can develop a polynomial expansion based upon the local coordinate system derived in Section 2.1. These expansions will be polynomials in terms of the local coordinates as well as the Cartesian coordinates (ξ_1, ξ_2, ξ_3). This is a significant property as primary operations such as integration and differentiation can be performed with respect to the local coordinates but the expansion may still be considered as a polynomial expansion in terms of the Cartesian system.

We shall initially consider expansions which are orthogonal in the Legendre inner product. We define three principle functions $\phi_i^a(z)$, $\phi_{ij}^b(z)$ and $\phi_{ijk}^c(z)$, in terms of the Jacobi polynomial, $P_p^{\alpha,\beta}(z)$, as:

$$\phi_i^a(z) = P_i^{0,0}(z), \quad \phi_{ij}^b(z) = \left(\frac{1-z}{2}\right)^i P_j^{2i+1,0}(z), \quad \phi_{ijk}^c(z) = \left(\frac{1-z}{2}\right)^{i+j} P_k^{2i+2j+2,0}(z).$$

Using these functions we can construct the orthogonal polynomial expansions:

Hexahedral expansion: $\phi_{pqr}(\xi_1, \xi_2, \xi_3) = \phi_p^a(\xi_1)\phi_q^a(\xi_2)\phi_r^a(\xi_3)$,

Prismatic expansion: $\phi_{pqr}(\xi_1, \xi_2, \xi_3) = \phi_p^a(\xi_1)\phi_q^a(\eta_2)\phi_{qr}^b(\xi_3)$,

Pyramidal expansion: $\phi_{pqr}(\xi_1, \xi_2, \xi_3) = \phi_p^a(\overline{\eta_1})\phi_q^a(\eta_2)\phi_{pqr}^c(\eta_3)$,

Tetrahedral expansion: $\phi_{pqr}(\xi_1, \xi_2, \xi_3) = \phi_p^a(\eta_1)\phi_{pq}^b(\eta_2)\phi_{pqr}^c(\eta_3)$,

where

$$\eta_1 = \frac{2(1+\xi_1)}{(-\xi_2-\xi_3)} - 1, \quad \overline{\eta_1} = \frac{2(1+\xi_1)}{(1-\xi_3)} - 1, \quad \eta_2 = \frac{2(1+\xi_2)}{(1-\xi_3)} - 1, \quad \eta_3 = \xi_3$$

are the local coordinates illustrated in Fig. 3.

The hexahedral expansion is simply a standard tensor product of Legendre polynomials (since $P_p^{0,0}(z) = L_p(z)$). In the other expansions the introduction of the degenerate local coordinate systems is linked to the use of the more unusual functions $\phi_{ij}^b(z)$ and $\phi_{ijk}^c(z)$. These functions both contain factors of the form $((1-z)/2)^p$ which is necessary to keep the expansion as a polynomial of the Cartesian coordinates (ξ_1, ξ_2, ξ_3). For example, the coordinate η_2 in the prismatic expansion necessitates the use of the function $\phi_{qr}^b(\xi_3)$ which introduces a factor of $((1-\xi_3)/2)^q$. The product of this factor with $\phi_q^a(\eta_2)$ is a polynomial function in ξ_2 and ξ_3. Since the remaining part of the prismatic expansion, $\phi_p^a(\xi_1)$, is already in terms of a Cartesian coordinate the whole expansion is a polynomial in terms of the Cartesian system.

The polynomial space, in Cartesian coordinates, for each expansion is:

$$\mathscr{P} = \mathrm{Span}\{\xi_1^p\ \xi_2^q\ \xi_3^r\}, \tag{1}$$

where pqr for each domain is

$$\begin{array}{llll} \text{Hexahedron} & 0 \leqslant p \leqslant N_1, & 0 \leqslant q \leqslant N_2, & 0 \leqslant r \leqslant N_3, \\ \text{Prism} & 0 \leqslant p \leqslant N_1, & 0 \leqslant q \leqslant N_2, & 0 \leqslant q + r \leqslant N_3, \\ \text{Pyramidal} & 0 \leqslant p \leqslant N_1, & 0 \leqslant q \leqslant N_2, & 0 \leqslant p + q + r \leqslant N_3, \\ \text{Tetrahedron} & 0 \leqslant p \leqslant N_1, & 0 \leqslant p + q \leqslant N_2, & 0 \leqslant p + q + r \leqslant N_3. \end{array} \tag{2}$$

The range of the p, q and r indices indicate how the expansions should be expanded to generate a complete polynomial space. We note that if $N_1 = N_2 = N_3$ then the tetrahedral and pyramidal expansions span the same space and are in a subspace of the prismatic expansion which is in turn a subspace of the hexahedral expansion.

To enforce C^0 continuity the orthogonal expansion is modified by decomposing the expansion into an interior and boundary contribution [10,12,18]. The interior modes (or bubble functions) are defined to be zero on the boundary of the local domain. The completeness of the expansion is then ensured by adding boundary modes which consist of vertex, edge and face contributions. The vertex modes have unit value at one vertex and decay to zero at all other vertices; edge modes have local support along one edge and are zero on all other edges, and vertices and face modes have local support on one face and are zero on all other faces, edges and vertices. Fig. 4 shows the decomposition of the domain into such elements, with the vertex, edge, face, and interior modes marked for one element. C^0 continuity between elements can then be enforced by matching similar shaped boundary modes. The local coordinate systems do impose some restrictions on the orientation in which triangular faces may connect. However, it has been shown in [10,19] that a C^0 tetrahedral expansion can be constructed for any tetrahedral mesh. A similar strategy could be applied to a hybrid discretization [20].

Finally, we note that the bases are all *hierarchical*, which means that increasing the polynomial order of any expansion simply adds extra modes to the existing basis. Hierarchical expansions naturally lend themselves to p-type adaptivity where the polynomial order of the expansion can differ within each elemental domain. This is a very attractive property as it permits the polynomial order of the expansion to be altered in order to capture the spatial characteristics of the solution.

2.3. Time integration algorithm

A popular time-stepping algorithm for integrating the Navier–Stokes equations is the splitting or fractional scheme. Although many different versions have been developed, here we describe a particular implementation that can give high-order time accuracy [21].

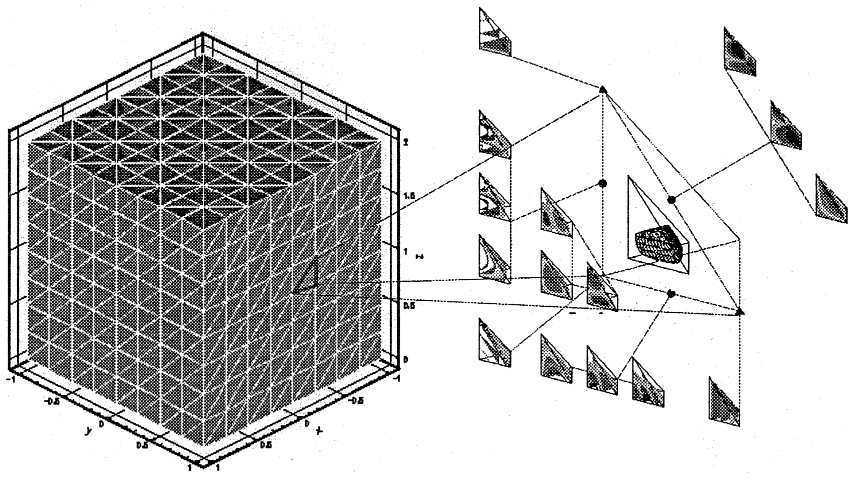


Fig. 4. In the spectral/*hp* method the solution domain is decomposed into elements of characteristic size h and then a polynomial expansion of order N is used within every element. On the left we see a cuboid decomposed into 3072 tetrahedral elements within which we use a polynomial expansion of order 4 as indicated by the mode shapes on the right.

Within a domain Ω, the fluid velocity $\boldsymbol{u}$ and the pressure p can be described by the incompressible Navier–Stokes equations,

$$\begin{aligned}&\frac{\partial \boldsymbol{u}}{\partial t} = -\nabla P + \nu \boldsymbol{L}(\boldsymbol{u}) + \boldsymbol{N}(\boldsymbol{u}) \quad \text{in } \Omega,\\ &\nabla \cdot \boldsymbol{u} = 0,\end{aligned} \tag{3}$$

where

$$\begin{aligned}&\boldsymbol{L}(\boldsymbol{u}) = \nabla^2 \boldsymbol{u}; \quad \boldsymbol{\omega} = \nabla \times \boldsymbol{u},\\ &\boldsymbol{N}(\boldsymbol{u}) = \boldsymbol{u} \times \boldsymbol{\omega}; \quad P = p + \frac{1}{2}\nabla(\boldsymbol{u}\cdot\boldsymbol{u}).\end{aligned} \tag{4}$$

The non-linear operator $\boldsymbol{N}(\boldsymbol{u})$ has been written in rotational form to minimize the number of derivative evaluations (6 vs. 9 for the convective form). A semi-implicit time integrator is used to integrate the system (3) and (4) by using a 3-substep splitting scheme [21]:

$$\frac{\hat{\boldsymbol{u}} - \sum_{q=0}^{J_i-1} \alpha_q \boldsymbol{u}^{n-q}}{\Delta t} = \sum_{q=0}^{J_e-1} \beta_q \boldsymbol{N}(\boldsymbol{u}^{n-q}), \tag{5}$$

$$\frac{\hat{\hat{\boldsymbol{u}}} - \hat{\boldsymbol{u}}}{\Delta t} = -\nabla \bar{P}^{n+1}, \tag{6}$$

$$\frac{\gamma_0 \boldsymbol{u}^{n+1} - \hat{\hat{\boldsymbol{u}}}}{\Delta t} = \nu \nabla^2 \boldsymbol{u}^{n+1}. \tag{7}$$

The time-stepping algorithm can then be summarized in three steps:

1. Calculate the advective terms Eq. (4) and advance the solution in time using a stiffly-stable multi-step integrator.
2. Solve a Poisson equation for the dynamic pressure P to satisfy the divergence-free condition for the solution. Consistent pressure boundary conditions are used to ensure the stability and high-order accuracy [21].
3. Implicitly solve the viscous terms, advancing the solution to the next timestep. This gives rise to a Helmholtz equation for each of the velocity components.

3. The $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ code

The $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ code is appropriate for flows with one homogeneous direction. In this direction a Fourier expansion is used providing a natural parallel paradigm. The spectral/*hp*-Fourier code was a direct evolution of a previous generation code on structured domains *Prism* [22,23] which used the same decomposition in the *x*–*y* plane and *z*-direction. The use of hybrid elements, a hierarchical basis and variable polynomial order across each elemental domain provides greater flexibility and permits a better accuracy for a fixed number of degrees of freedom in the *x*–*y* plane than previously possible. An illustrative example of the *x*–*y* plane mesh for a 3D flow past a cylinder is shown in Fig. 5.

3.1. Fourier decomposition

If we assume that the problem is periodic in the *z*-direction, we may use a Fourier expansion to describe the velocity and the pressure, i.e., for the velocity,

$$\boldsymbol{u}(x,y,z,t) = \sum_{m=0}^{M-1} \boldsymbol{u}_m(x,y,t)\mathrm{e}^{\mathrm{i}\beta m z}, \tag{8}$$

where β is the *z*-direction wave number defined as $\beta = 2\pi/L_z$, and L_z is the length of the computational domain in the *z*-direction. We now take the Fourier transform of Eq. (3) to get the coefficient equation for each mode *m* of the expansion

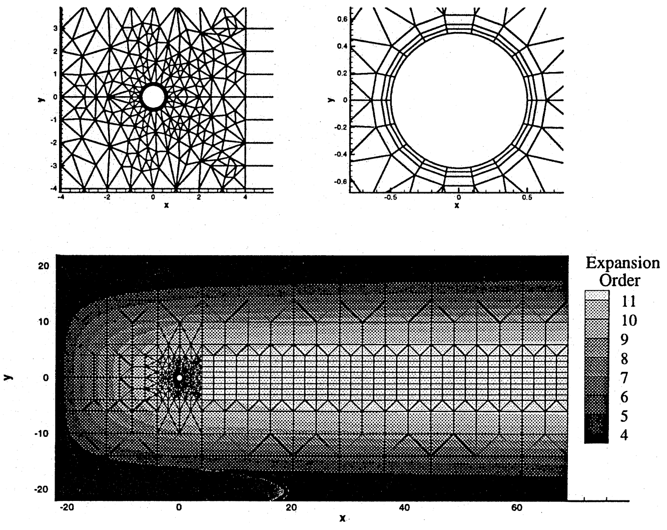


Fig. 5. 2D grid for the x–y plane and polynomial order of approximation for a $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ simulation of flow past a circular cylinder.

$$\frac{\partial \boldsymbol{u}_m}{\partial t} = -\tilde{\nabla} p_m + \nu \boldsymbol{L}_m(\boldsymbol{u}_m) + \mathrm{FFT}_m[\boldsymbol{N}(\boldsymbol{u})] \quad \text{in } \Omega_m, \quad m = 0,\ldots,M-1, \tag{9}$$

where FFT_m is the mth component of the Fourier transform of the non-linear terms and

$$\begin{aligned} \tilde{\nabla} &= \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, im\beta\right), \\ \boldsymbol{L}_m(\boldsymbol{u}_m) &= \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} - \beta^2 m^2\right)\boldsymbol{u}_m. \end{aligned} \tag{10}$$

The computational domain Ω_m is an x–y slice of the domain Ω, implying that all Ω_m are identical copies. From Eq. (9) we see that the only coupling between modes is through the non-linear terms. Therefore the computation of each mode m can be treated independently of one another. The obvious parallelization is to compute the mth Fourier mode on processor m for $m = 0,\ldots,M-1$. Therefore, the 3D computation essentially becomes a set of $N_z = 2M$ 2D problems computed in parallel on P processors where M is a multiple of P. We note that the factor of two comes from the real/imaginary part pairs for the Fourier modes.

To maintain computational efficiency the non-linear product is calculated in physical space while the rest of the algorithm may be calculated in transformed space. The paradigm may therefore be thought of as a two pass process as illustrated in Fig. 6. As mentioned previously the spectral/hp representation in the x–y plane in hierarchical and so we may also consider this representation as a set of elemental modes and corresponding coefficients. In the first pass of the paradigm we need to obtain the physical data values at the quadrature points within each elemental domain. The inverse Fourier transform and differentiation are then performed at these points. For each timestep, Pass I can be summarized by the following substeps:

1. The velocity gets transformed to Quadrature space.
2. Calculation of the vorticity.
3. To form the non-linear terms:
   (a) Global transpose of the velocity and vorticity components.
   (b) N_{xy} 1D inverse FFTs for each velocity and vorticity component, (where N_{xy} is the number of points in one x–y plane divided by the number of processors).
   (c) Computation of $\boldsymbol{N}(\boldsymbol{u})$ using a dealiasing 3/2 rule.
   (d) N_{xy} 1D FFTs for each non-linear term.
   (e) Global transpose of non-linear terms.

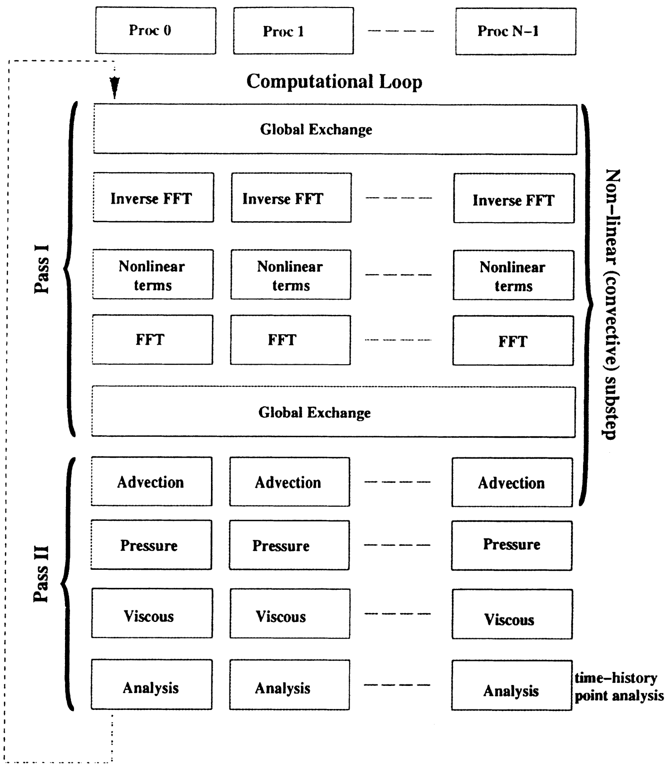


Fig. 6. Solution process in $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$.

In Pass II the explicit time-integration of the non-linear terms and then the Helmholtz solves for pressure and velocity may be performed independently on each processor.

3.2. 2D Helmholtz solves

Having reduced the linear part of the 3D problem to a set of 2D elliptic solves, we now describe Pass II in more detail. The major operations are computation of the gradients and inner products to form the right-hand side of the Helmholtz problem and subsequently find its solution. The operations may be either global or element-wise in nature; solution of the Helmholtz problem is a global operation, while computing gradients in the x- and y-directions and evaluating the inner product are all element-wise operations. In the z-direction, because of the Fourier decomposition, differentiation becomes an exchange of the real and imaginary parts of a mode multiplied by a constant. As both parts of a mode reside on the same processor, this operation is local to the processor and requires no communication overhead.

Furthermore, the x- and y-gradients can be calculated using matrix–matrix multiplies representing differentiation in a single direction, for example: $(\partial/\partial x)\phi_i(k) = D_x(k)\phi_i(k)$, where in this case $\phi_i(k)$ is written out as an $Q^a \times Q^b$ matrix [2] and $D_x(k)$ is the x-direction derivative matrix for element k. The operation count for computing these gradients is $O(N^3)$ per element. These operations involve small matrix–matrix multiplies that unfortunately do not benefit from the usual optimizations employed for large matrices. The inner product evaluation can be handled similarly.

A Helmholtz problem needs to be solved for each velocity component and the pressure at every time step. Each discrete Helmholtz problem, when discretized within a Galerkin formulation, results in a matrix problem which can be solved either directly or iteratively. However, the structure of the spectral/*hp* expansion basis is such that the modes may be classified in terms of boundary modes, which have support on

[2] $Q^a \approx N$ and $Q^b \approx N$ are the numbers of quadrature points in each direction η_1 and η_2. N is the polynomial approximation order.

the boundary of an elemental domains, and interior modes which do not. Therefore, when considering a matrix problems arising from a C^0 continuous expansion we find that the submatrices corresponding to the interior modes of a specific elemental domain are decoupled from the interior modes of another elemental domain. This fact leads to the natural decomposition where the interior modes are decoupled from the boundary system by assembling the Schur complement matrix corresponding to the elemental boundary degrees of freedom, see [16] for more details. Constructing the Schur complement means that we can employ a direct or iterative solve of a much small matrix system. If the matrix is inverted iteratively a preconditioned conjugate gradient (PCG) algorithm is typically used, while the inversion of the matrix directly can be optimized by ordering the boundary degrees of freedom to reduce the matrix bandwidth. A further decomposition, similar to the interior-boundary decomposition mentioned, is possible in the boundary matrix system by identifying groups of elemental boundaries which do not overlap. This technique is known as substructuring [24] and can lead to a greatly reduced interior solve providing a very efficient direct solve technique with reduced memory requirement as compared to using the whole boundary matrix system.

Finally we should make the following points about $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$:

1. A very important advantage of the spectral/*hp*-Fourier decomposition is its speed as we can usually employ direct solvers for the solution of the 2D Helmholtz problems.
2. Increasing the resolution in the z-direction (i.e., increasing the number of Fourier modes) can be accomplished *without* increasing the amount of memory needed per processor if the number of processors is increased accordingly.
3. Increasing the resolution in the z-direction while increasing the number of processors involves a longer execution time per processor but only to the extent of the increase in communication time for global exchanges as well as the time to perform a 1D FFT. While the operation count for an FFT is $O(n \log n)$, the speed a modern processor will perform an FFT will also increase (until a critical value of n) [25–27] thereby partially offsetting the increase in operation count.
4. The x–y plane resolution on distributed memory machines is only limited by the memory available per processor. The real and imaginary parts of a mode share the same matrices which saves memory. With a slightly larger overhead due to extra communication it is also possible to store only the real or the imaginary part of a Fourier mode on each processor, thereby reducing some memory requirements.

3.3. Communications in $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$

The main types of communication patterns used in $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ are:

1. Global Exchanges (All-to-All) in the non-linear substep, specifically in Pass I.
2. Global Reduction operations (addition, min, max) for any runtime flow statistics.
3. Global Gather for any possible outflow boundary conditions in the pressure substep.
4. Gather for any time-history point analysis (tracking of flow variables at some point) in the analysis substep.
5. Global Synchronization (at a few points in the code to ensure that every processor is ready to proceed to the next substep).

In this paper we concern ourselves with the dominant communications in a $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ production run, namely the Global Exchange and to a lesser extent the Global Reduction and Gather operations.

3.3.1. Global exchange

The communication upon which the code is based is the Global or Complete Exchange. This is a communication pattern of great importance to any 2D or 3D FFT-based solver, since it lies behind the transposition of a distributed matrix. For example, it is the dominant communication pattern in any spectral distributed memory homogeneous turbulence in a 'box code'. It is also used in multiplying distributed matrices when one or more of the matrices is specified to be in transposed form. In the case of $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ it is used to move the data between Fourier and Physical space: For most of the calculation, the flow variables are in Fourier space distributed in 'Fourier planes' among the processors. However when the need to form the non-linear products (Pass I) arises, the data are transfered to Physical space (Fig. 7).

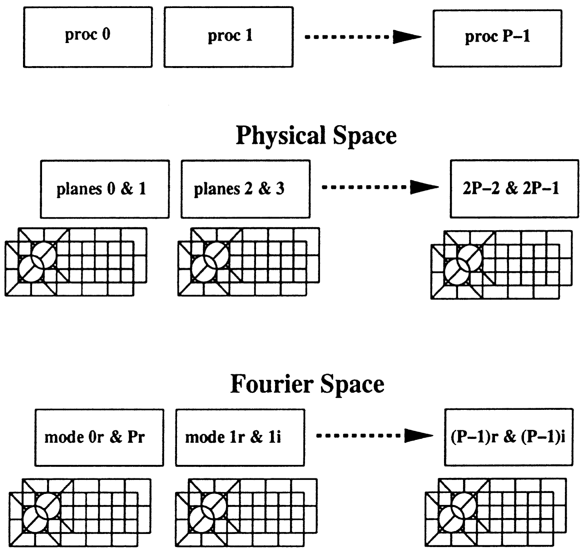


Fig. 7. Data Layout in $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$. Here we have chosen to store a Fourier mode (2 Fourier planes) per processor. This also means that we keep 2 'Physical planes' per processor. Because this is a real-to-complex FFT, $N_z = 2^m = 2P$ Physical planes map to $P+1$ *independent* Fourier modes (0 to $N_z/2 = P$) as the other $P-1$ modes ($N_z/2+1$ to N_z-1) are fixed by symmetry. Of these $P+1$ modes, the first and the last one have vanishing imaginary parts, hence by 'packing' the real part of mode P in place of the imaginary part of mode 0 we are left with $N_z = 2P$ Fourier planes as well.

The rotational form of the Navier–Stokes equations requires 6 Complete Exchanges to get the 3 velocity and the 3 vorticity fields arranged along 'Fourier pencils'. Then a multiple-point inverse FFT is applied to this collection of 'pencils' per processor to obtain the velocity and vorticity fields along constant (x, y) lines, termed 'Physical pencils', as indicated in Fig. 8). The rotational form of the non-linear (convective) term is then calculated, multiple-point FFTs then transforms the data back into the form of Fourier pencils, and a Complete Exchange per component of the non-linear terms is applied to bring the data back to original form of 'Fourier plane' (see Fig. 6). In total, the rotational form of the Navier–Stokes equations requires 9 Complete Exchanges per timestep.

During a Complete Exchange, each processor communicates with each other processor and so it is one of the most demanding communication patterns on network resources. In the case of $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$, each individual message is of a size

$$\left(\frac{\sum_{k=1}^{n_{\rm el}} {}^aQ_k^bQ_k}{P} \times \frac{N_z}{P}\right)$$

double precision words, where the sum is over all $n_{\rm el}$ elements. Therefore as the number of processors P increases the message sizes decrease, even if N_z/P remains constant, unless the number of elements $n_{\rm el}$ or the spectral order N increase accordingly. This means that as P increases message latency becomes ever more important.

Complete Exchanges are also needed to setup the boundary conditions or in the case of time-dependent boundary conditions.

3.3.2. Global reduction

Global reduction operations (summation, as well as min/max) also appear in the code when runtime statistics are gathered. The values are calculated on each processor and in the end are reduced across all processors.

3.3.3. Global gather

In the rotational form of the Navier–Stokes equations, the dynamic pressure $\Pi = p + (1/2)(u^2 + v^2 + w^2)$ needs to be evaluated at the outflow boundary. This requires the assembly of a

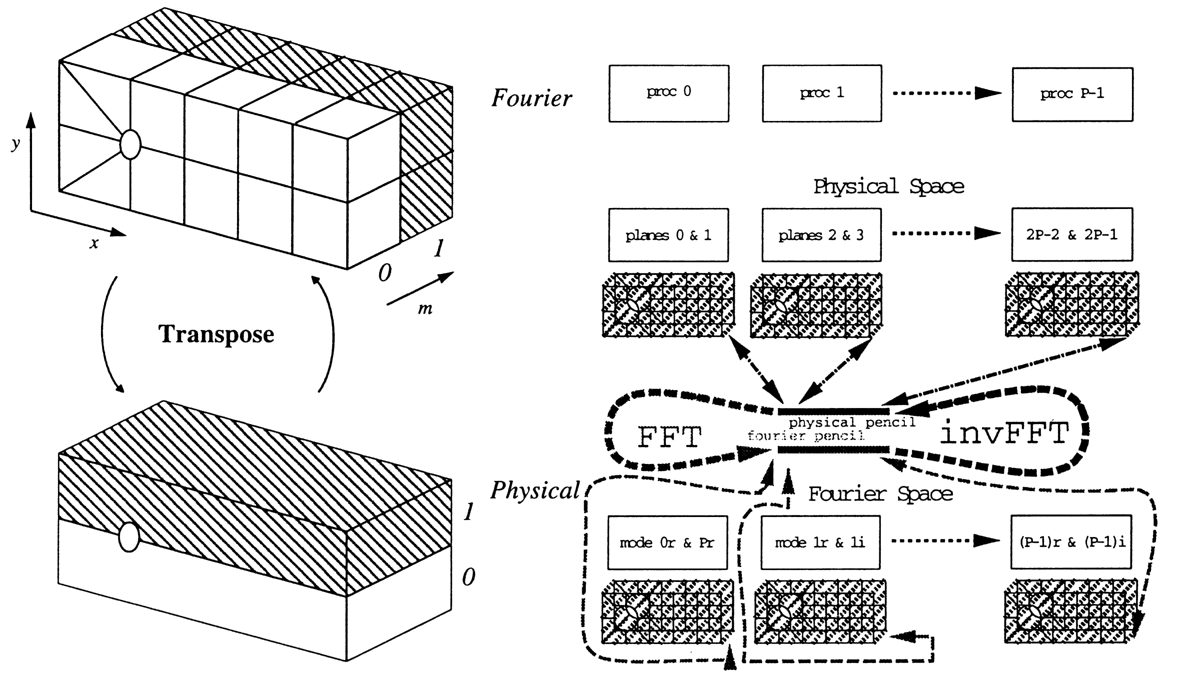


Fig. 8. A Complete Exchange ('Global Transpose') followed by a series of inverse FFTs on the resulting Fourier pencils transforms the data to Physical space arranged along Physical pencils. Pencils' are contiguous vectors containing either all the Fourier modes or all the Physical space data in the z-direction at a given (x, y) point.

vector consisting of the Fourier pencils corresponding to each of the discrete (x, y) points along an outflow boundary. Thus a gather operation on all processors followed by internal rearrangements of the resulting vector of mixed Fourier pencils is necessary.

3.3.4. Gather

For time-history point analysis, we again need the data in Physical space for a specific (x, y, z) triad. This means that a Fourier pencil corresponding to the discrete (x, y) pair needs to be constructed and an FFT applied to obtain the Physical pencil containing the required z point. Therefore, all processors send their part of the Fourier pencil to processor 0, the 'root' processor, where the above procedure is performed.

This gather operation needs to be done for each history point and for each field (typically 4 values – u, v, w and p) whose history is being tracked. The message size in each case is N_z/P double precision words.

3.3.5. Communications summary

In Table 1 we summarize the frequencies and message sizes of the communications in $\mathscr{N}\varepsilon\kappa\mathscr{T}\alpha r\mathscr{F}$.

Table 1
Communication patterns, corresponding message lengths and frequency per timestep in $\mathscr{N}\varepsilon\kappa\mathscr{T}\alpha r\mathscr{F}$

Routine	Times per timestep	Message (bytes) size
All-to-all (rotational)	6 + 3	$8(\sum_{k=0}^{K} {}^aQ_k^b Q_k/P) \times N_z/P$
Allgather (rotational)	Number of outflow edges	$24N_z \sum_{k=0}^{Kb} {}^aQ_k$
Gather	4 × number of history points	$8(N_z/P)$

3.4. Turbulent flow past a cylinder

Using the parallel code $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ we performed systematic simulation of 3D flow past a turbulent cylinder at Reynolds number 500, 1000 and 3900 using a mesh similar to the one shown in Fig. 5.

First, we plot in Fig. 9 the velocity spectra from our DNS at $Re = 500, 1000$ and 3900. The standard non-dimensionalization with the freestream velocity and the cylinder diameter is applied. The history point considered was at $(x = 3D,\ y = D)$ for the $Re = 500, 1000$ cases and at $(x = 2.57D,\ y = 0.52D)$ for the $Re = 3900$ case. The cylinder center was at $(x = 0,\ y = 0)$ and D is the cylinder diameter. We see that a substantial inertial range exists only for the $Re = 3900$ flow.

Next we present comparisons of $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ predictions with data from experiments at $Re = 3900$. In Fig. 10 we compare mean streamwise velocity profiles in the very near-wake with the PIV data of Lourenco and Shih, that were published in [28]. We see that very good agreement is obtained and this has also been verified by extensive resolution tests as shown in [29].

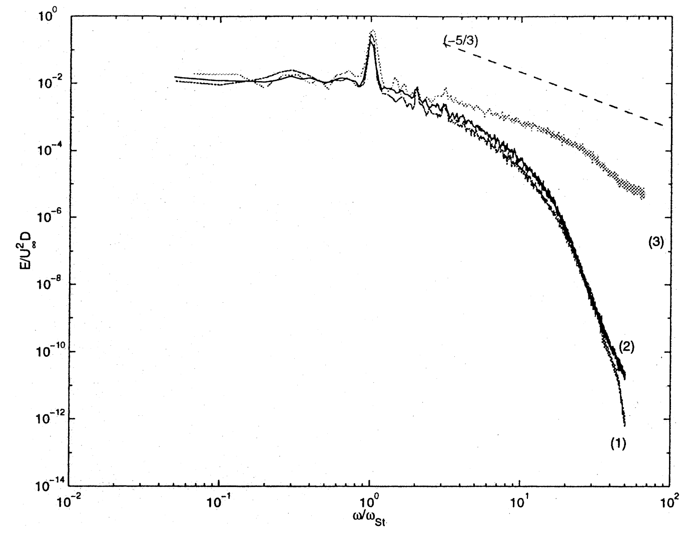


Fig. 9. 1D streamwise velocity spectrum (energy versus normalized frequency) in the near-wake of flow past a cylinder at $Re = 500$ (curve 1), 1000 (curve 2), and 3900 (curve 3).

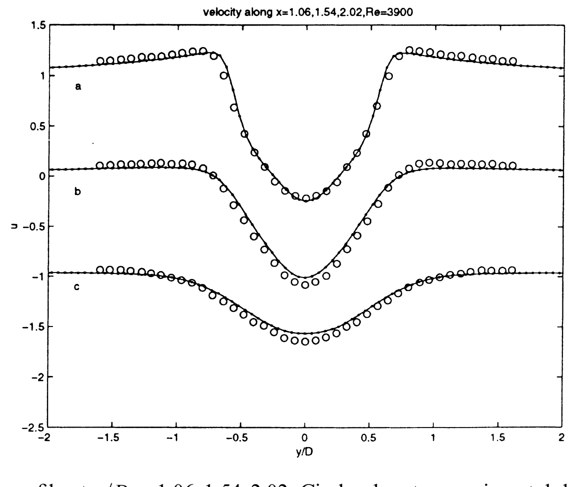


Fig. 10. Streamwise mean velocity profile at $x/D = 1.06, 1.54, 2.02$. Circles denote experimental data of Lourenco and Shih and solid-line $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r$ simulation.

4. The $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r$3D code

The need to directly simulate flows past geometries of even greater complexity, which have no planes of symmetry, requires the development of fully 3D high-order algorithms. Application areas for such algorithm are potentially very broad, however the need for unstructured algorithms becomes even more evident when we simply consider the complexity of meshing the solution domain. The development of 3D spectral/*hp* unstructured and hybrid algorithms [11,10,12] has notably extended the original spectral element algorithms which were traditionally based on structured, hexahedral discretizations. One application area that these technique are currently being applied is the arterial haemodynamics. In Fig. 11 we see a model geometry for internal flow within a bifurcated pipe which is being used to model the distal end of an end-to-sided arterial bypass graft [30,31]. Similar to the $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ code the cost of simulating large 3D computations, with a high degree of accuracy, requires a parallel algorithm. We recall that parallel implementation is particularly suited to the spectral/*hp* type algorithms due to the close coupling of information within an element generated by the high-order polynomial representation. Typically, even in the serial algorithm many operations such as integration and differentiation are treated at a very local elemental level.

The parallelization of the 3D unstructured solver was accomplished by the modification of the serial version of $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r$3D. The key components in this transition from serial to parallel were:

1. Generation of a local to global mapping of the boundary information based on an arbitrary partitioning.
2. Introduction of a communication interface to treat global operations.
3. Modification of the PCG solver to treat interprocessor communications over each iteration.
4. Implementation of a suitable I/O format for the parallel computation.

4.1. Parallelization of $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r$3D

4.1.1. Domain partitioning and numbering

The first modification, and from an implementation standpoint the most costly, was the development of a numbering scheme to relate the global ordering of the solution variables over the whole computational domain to the local partition of the domain. The first step in constructing this numbering scheme is to divide the solution domain across the available processors. Since, the spectral/*hp* method tightly couples information within each element this partitioning is performed on an elemental level. However, to maintain the generality of the implementation it is presumed that any element may be arbitrarily placed on any processor. Although it is desirable in practice to locate as many neighboring elements as possible within a specific processor, as shown in Fig. 12, by assuming a general distribution we are able to use any of the freely available partitioning packages such as 'Metis' [32], 'Chaco' [33], 'Jostle' [34] and 'WGPP' [35].

The spectral/*hp* expansion basis implicitly decouples the interior degrees of freedom of each element from the boundary degrees of freedom. This is important as it reduces the global numbering to the global

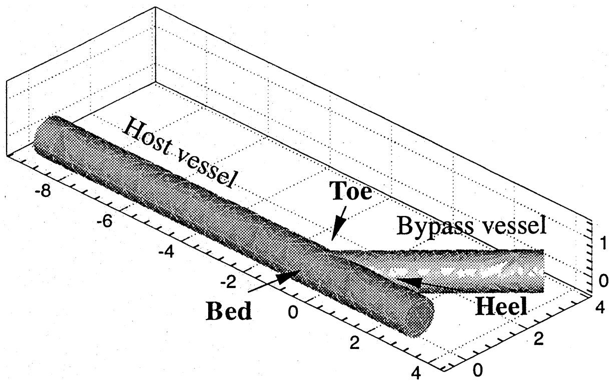


Fig. 11. Model geometry of a distal end-to-side anastomosis using a viscous tetrahedral mesh from the Felisa package.

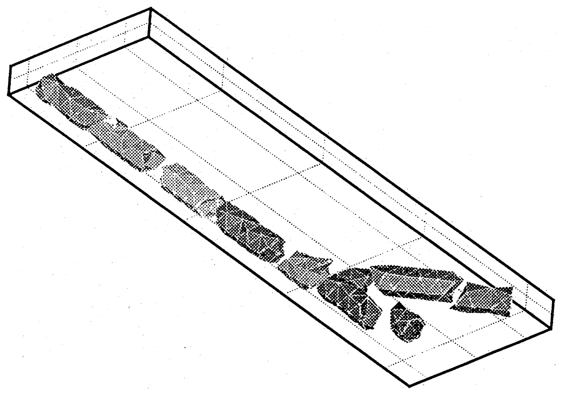


Fig. 12. Partitioning of the computational mesh shown in Fig. 11 into $P = 8$ regions using the Metis package.

degrees of freedom on the boundaries of elements. The mapping of the global boundary numbering system was then generated by constructing a global 'skeleton' mesh on each processor with a unique global numbering. The scheme can be summarized in three-steps as:

1. Determine the global numbering of the boundary system on the complete mesh. (This is assumed to be unique so each processor has an identical numbering scheme.)
2. Construct a compact local boundary numbering using the global numbering to identify any multiply defined elemental information (e.g., due to vertex multiplicity).
3. Determine the mapping relating the local system to the global system and then remove the global system.

Although such a scheme is conceptually easy to interpret it is rather more complex to implement. Nevertheless, determining this mapping is the most significant aspect of the work necessary to parallelize our algorithm.

At present equal weighting is attributed to every element when using the domain partitioner. However when using a variable polynomial expansion or if there is a very high multiplicity of elements at a vertex or edge a weighting system should be applied. Node and edge weighting is typically available in most graph partitioners however the relation of such a weighting system to the spectral/*hp* element mesh requires further investigation.

4.1.2. Communication interface

Typically, the introduction of a communication interface would represent a major part of any parallel implementation. However, this part of the implementation was dealt with by a 'Gather–Scatter' (GS) library developed by Tufo and Fischer which is currently freely available software [36]. Having obtained a global to local processor mapping, as discussed in the Section 4.1.1, this interface performs all the necessary operations in a finite element type calculation such as direct stiffness assembly and therefore greatly reduced the overhead of the implementation. We note that the package is not restricted to the spectral/*hp* element algorithm but could be applied to a range of potential implementations including the standard finite element method. The interface is a very versatile package, allowing the treatment of all the communications using a 'binary-tree' algorithm, 'pairwise' exchanges or a mix of these two approaches where pairwise exchange are used for communicating values that are shared by only a few processors and a tree-like approach is used for values shared by many processors. This latter approach is basically a global reduction operation on a subset of the total number of processors.

4.1.3. PCG solver modification

The standard iteration of the PCG routine to solve $\boldsymbol{Ax} = \boldsymbol{b}$ is shown in Fig. 13(a) where α_p and β_p are constants and $\boldsymbol{M}$ is the preconditioning matrix. It can be appreciated that in a parallel algorithm each processor may keep its local contribution to the vectors $\boldsymbol{r}_i, \boldsymbol{p}_i$ and z_i, however, to update these vectors we need to perform a global operation to determine α_i and β_i which requires us to evaluate the matrix–vector product $\boldsymbol{w}_i = \boldsymbol{Ap}_i$ and the inner products $\boldsymbol{r}_{i-1}^T z_{i-1}$ and $\boldsymbol{p}_i^T \boldsymbol{w}_i$

(a)
```
Solve $\boldsymbol{M}\boldsymbol{z}_i = \boldsymbol{r}_i$.
$i \quad = i + 1$
$\beta_i \quad = \boldsymbol{r}_{i-1}^T\boldsymbol{z}_{i-1}/\boldsymbol{r}_{i-2}^T\boldsymbol{z}_{i-2}$
$\boldsymbol{p}_i \quad = \boldsymbol{z}_{i-1} + \beta_i\boldsymbol{p}_{i-1}$
$\boldsymbol{w}_i = \boldsymbol{A}\boldsymbol{p}_i$
$\alpha_i \quad = \boldsymbol{r}_{i-1}^T\boldsymbol{z}_{i-1}/\boldsymbol{p}_i^T\boldsymbol{w}_i$
$\boldsymbol{x}_i \quad = \boldsymbol{x}_{i-1} + \alpha_i\boldsymbol{p}_i$
$\boldsymbol{r}_i \quad = \boldsymbol{r}_{i-1} - \alpha_i\boldsymbol{w}_i$
```

(b)
```
Solve $\boldsymbol{M}\boldsymbol{z}_i = \boldsymbol{r}_i$.
$i \quad = i + 1$
$\beta_i^p \quad = \boldsymbol{r}_{i-1}^T\boldsymbol{m}_{ult}\,\boldsymbol{z}_{i-1}/\boldsymbol{r}_{i-2}^T\boldsymbol{M}_{ult}\,\boldsymbol{z}_{i-2}$
Globally sum $\beta_i^p$
$\boldsymbol{p}_i \quad = \boldsymbol{z}_{i-1} + \beta_i\boldsymbol{p}_{i-1}$
$\boldsymbol{w}_i^p = \boldsymbol{A}^p\boldsymbol{p}_i$
$\alpha_i^p \quad = \boldsymbol{r}_{i-1}^T\boldsymbol{m}_{ult}\boldsymbol{z}_{i-1}/\boldsymbol{p}_i^T\boldsymbol{w}_i^p$
Globally sum $\alpha_i^p$ and assemble $\boldsymbol{w}_i^p$
$\boldsymbol{x}_i \quad = \boldsymbol{x}_{i-1} + \alpha_i\boldsymbol{p}_i$
$\boldsymbol{r}_i \quad = \boldsymbol{r}_{i-1} - \alpha_i\boldsymbol{w}_i$
```

Fig. 13. Operations in the ith iteration of a preconditioned conjugate gradient loop. (a) Serial version and (b) parallel version on pth processor.

As shown in Fig. 13(b) the PCG routine can be parallelized by introducing two global communication steps. Each processor is assumed to have duplicate copies of the relevant entries of the global vectors. However, the evaluation of β_i involves interprocessor communication. Initially, the inner product is evaluated on the pth processor and then globally summed. However, since parts of the global vector are multiply stored on different processors, the local evaluation of the inner products involves the multiplicity matrix, $\boldsymbol{m}_{\text{ult}}$, of the entries of the vector over all processors.

Typically the matrix $\boldsymbol{A}$ can be expressed in terms of its elemental contributions and so the product $\boldsymbol{Ap}$ can be performed over each element independently, which makes it trivial to distribute this operation over the processors. However, the elemental contributions need to be globally assembled. Therefore, we first determine the local product $\boldsymbol{w}^p = \boldsymbol{A}^p\boldsymbol{p}$ from which we can also evaluate the contribution of the pth partition to α_i^p. The remaining communication to globally assemble $\boldsymbol{w}_i^p$ and sum α_i^p may then be combined, thereby reducing the overhead latency.

A further communication is generally necessary to evaluate a measure of the residual to use as a stopping criterion. This communication can be amalgamated with one of the other two existing communications if the residual calculation is lagged at the possible expense of an extra iteration.

4.1.4. I/O modifications

The last modification for the parallel algorithm was the introduction of a parallel output format. At present we assume that all processors may access a single input file, however, due to the memory restriction on any given processor it is not feasible to assemble output data via a single processor. Therefore, it was necessary to replace the existing output format which produced a local 'header' and 'data' output file on each processors. The header files contains information about the local distribution of data on the specific processor while the data files contains all the solution data. The advantage of this format is that a single output file may then be generate by reading the relatively small header file and concatenating the data file with a simple 'cat' system call.

4.2. Parallel validation

In order to evaluate the performance of the solver we conducted tests on a scalar Helmholtz problem which forms the backbone of a splitting scheme [21] used to solve the Navier–Stokes equations. The timings were performed up to 192 (thin) nodes of the IBM SP2 at Cornell Theory Center. The SP2 communication network has a flat logical topology, and so communications between two processors do not depend on their proximity or whether neighboring processors are concurrently communicating. Therefore, the mapping of the solution domain partitions to processors need not preserve the relative positions of the subdomains to each other.

To keep our prototype problem as simple as possible we have chosen a bi-unit box subdivided into tetrahedral elements of equal volumes as illustrated in Fig. 4 where we see the box geometry divided up into 3072 elements. A fixed polynomial expansion order ($N = 6$) was used for all elements so as not to create load imbalance. The solution converged in all cases, giving us the same value for the L^2 and H^1 errors. The following results represent a lower limit to the performance of the main solver.

Our initial tests were performed on a 3072 element mesh using a purely pairwise communication strategy. However as can be seen in Fig. 14 the scaling of this test case was very poor. There were two possible reasons for this bad performance. Firstly, we believed that the solver may be saturating due to a low loading on each processor and secondly we also believed that the purely pairwise communication strategy was inappropriate since it is most suited to information which is only shared between a small number of processors. However since the inner product evaluation of α_i^k is lumped with the direct stiffness assembly in our conjugate gradient algorithm there is always at least one piece of information which is communicated over all processors.

This second fact motivated the introduction of a binary-tree type communication strategy into the 'GS' communication package as well as combination of tree and pairwise communication depending on how many processors the information is distributed over. These three approaches are labeled as 'all tree', pairwise and 'partial tree' in Fig. 15. From this plot we see that the mixed partial tree approach is

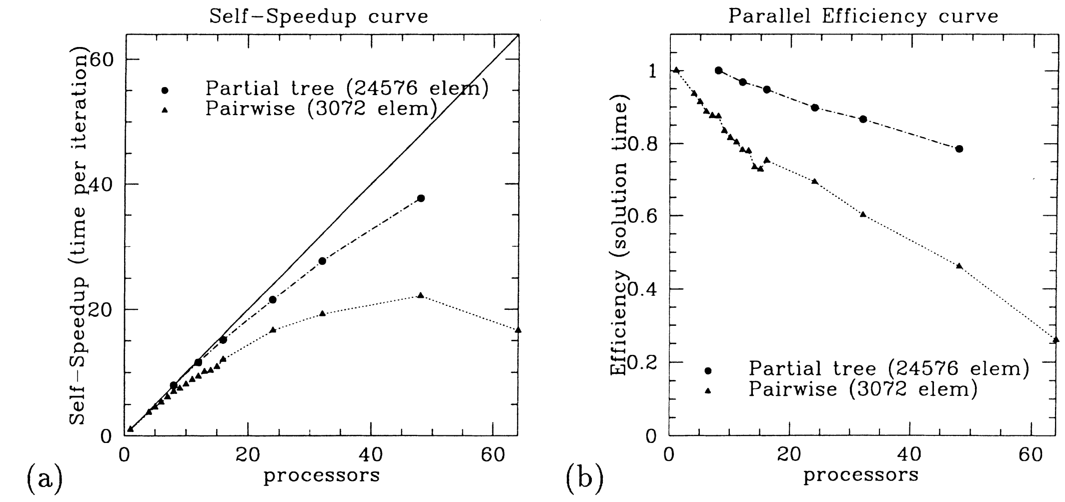


Fig. 14. Self speed up (a) and parallel efficiency (b) for a Helmholtz problem in a cubic domain using $n_{\rm el} = 3072$ and $n_{\rm el} = 24,576$ elements with ($N = 6$).

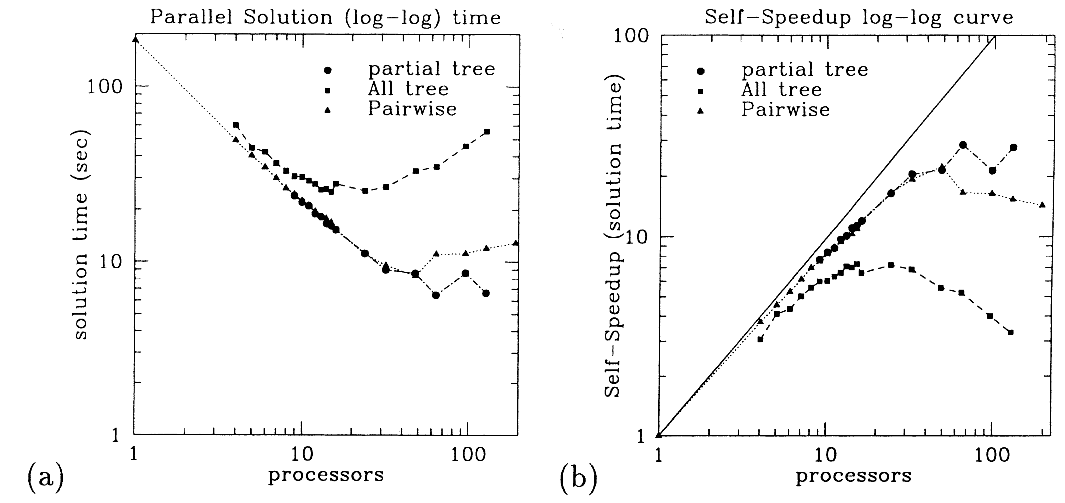


Fig. 15. Solver time (a) and parallel self-speedup (b) for Helmholtz problem in a cubic domain decomposed in 3072 elements of polynomial order ($N = 6$).

advantageous compared to purely pairwise or a pure binary-tree. The reason for this is because the majority of the communicated data is only distributed over a limited number of processors. However we always have some data due to the inner-product evaluation which is distributed over all processors. The pairwise approach suffers greatly when communicating over all processors since each processor needs to send a small message to every other processor. On the other hand, using the binary tree algorithm for all, every communicated piece of data is very wasteful as in most cases values are shared by only two processors.

As can be seen in Fig. 15(a) the solver time for the partial tree runs decreased with increasing the number of processors until 64 processors where it reaches a minimum and then flattens out (allowing for the larger timing irregularities at that number of processors). This plateau is seen at a later stage for higher polynomial order calculations as Amdahl's law would suggest. In the case of the pairwise approach, as the number of processors increased beyond 48 the solution time actually starts increasing as previously explained. The "all tree" method is actually the worst in terms of performance, again as expected, showing that a binary tree algorithm is a very poor choice for dealing with this global reduction operation.

As a final test case we considered a $n_{\mathrm{el}} = 24,576$ element mesh with $N = 6$ and a partial tree communication strategy. As shown in Fig. 14 this case demonstrates greatly improved performance. Although we were only able to scale the problem to $P = 48$ processors, a parallel efficiency of 80% was achieved as compared with the $P = 8$ processor case.

4.3. Results

To demonstrate the parallel $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r$3D we consider the steady flow within the model bypass geometry shown in Fig. 11. This model was used in [16] to study how 3D deformation of the inflow bypass vessel affects the internal flow pattern and wall shear stress. In Fig. 11 we see a comparison of the numerical simulation with magnetic resonance images (MRI) taken from a similar model both at a Reynolds number of $Re = 250$. As can be appreciated, there is a good agreement between the numerical and the experimental data. A flow quantity which is not so easy to extract experimentally is the wall shear stress. This quantity is of interest since it has potentially important implications for the onset of disease which may lead to the failure of bypass grafts. While experimentally determining the wall shear stress is difficult, it is relatively straight forward task to extract it computationally. Therefore, the comparison in Fig. 16 is particularly useful in building confidence in the numerical wall shear stress. The computations were performed on $P = 8$ processors of a Fujitsu AP3000 and although we have considered only the steady-state solution the computation was performed using the time dependent algorithm discussed in Section 2.3. The computational mesh contained $n_{\mathrm{el}} = 1687$ elemental domains and was generated using a viscous tetrahedral mesh from the Felisa package [37,38]. The simulations were performed at polynomial orders of $N = 2, 4$ and $N = 6$ which is equivalent to hierarchically refining the mesh.

5. Communication timings

It is instructive to try and measure some of aforementioned communication primitives on their own, to see how they scale with the number of processors P and the individual message sizes M_{sz} involved. In this manner, any limitations of algorithms or of available hardware/software can be explored in a more efficient manner.

We varied the P from 1 to 256 in powers of 2, and M_{sz} from 8 to 800,000 bytes in multiples of 8, 80, 800, etc. We timed the MPI implementations of three collective communication primitives, namely MPI_Alltoall( ) (Global Exchange), MPI_Allreduce( ) (Global Reduction) and MPI_Allgather( ) (Global Gather). While timing collective communications is not very common and usually gets done for either a fixed message size or a fixed processor count, the send–receive pairs that are needed for the pairwise exchanges in the GS package are among the standard performance tests widely available so we will not deal with them here [39–43]. The timings were performed on all processors and repeated 100 times. Each time we recorded the maximum time it a processor took to return from the MPI call. As all of these calls are blocking, the

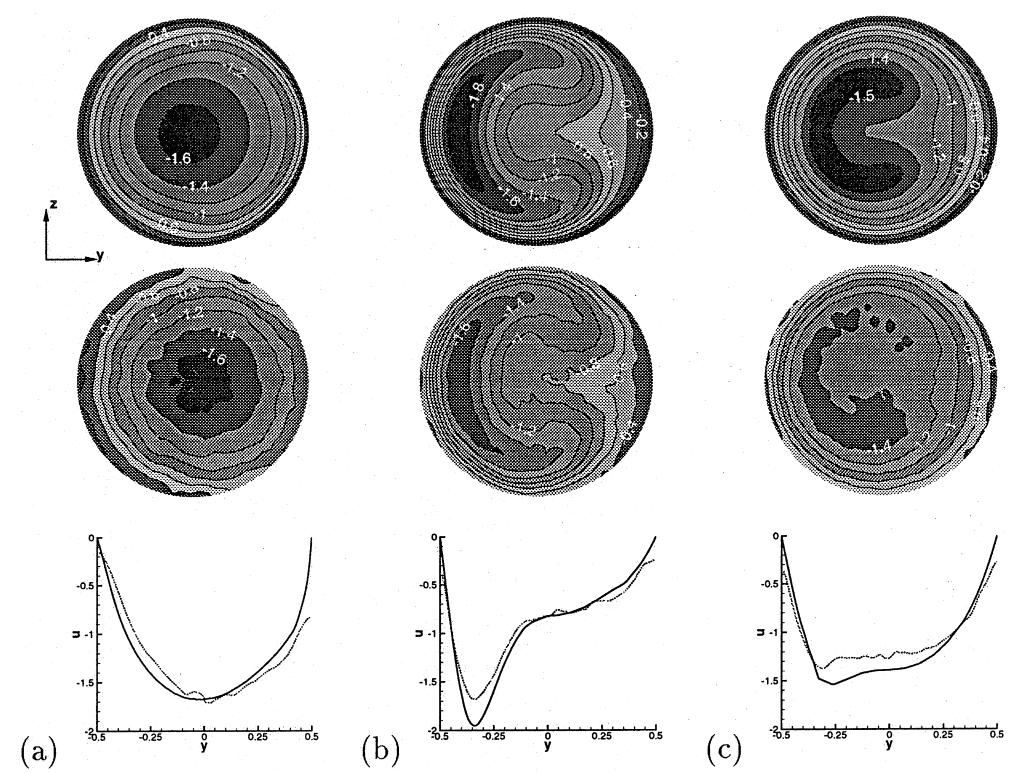


Fig. 16. Comparison CFD (top) and MRI (middle) axial velocity at (a) toe, (b) 2D distal to the toe and (c) 5D distal to the toe. Also shown are a comparison of the CFD (dark line) and MRI (light line) profile of velocity extracted along the constant z centerline.

processor returns only when it has finished communicating. In Figs. 17–19 we plot the minimum values observed during these 100 experiments.

As can be seen in Fig. 17, there exists a latency dominated region for small M_{sz} where time is virtually independent of M_{sz}. Above 64 B messages (corresponding to 8 double precision real numbers of 4 double precision complex ones) the effect of M_{sz} becomes appreciable. Towards the upper right-hand corner of the plot (the limit of large messages and many processors) the time for the communication operation increases rapidly.

In the case of $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ we usually increase both P and resolution in the (homogeneous) z-direction so that their ratio remains constant. This means that our message size is inversely proportional to the number of processors ($M_{sz} \propto 1/P$). Then for a given x–y resolution we vary along one of the solid lines of slope -1 drawn in the graph. As can be readily seen, for small P we cross the contour lines at a sharp angle and time increases slowly. As P increases though, this angle becomes larger (we approach the latency dominated limit) and time increases faster.

Provided we start with more than one Fourier mode per processor we can increase P (until that limit is reached) keeping resolution in z constant. Then $M_{sz} \propto 1/P^2$ and this time we vary along the dashed line of slope -2. For small P this line stays parallel with the contour lines but very quickly it runs into the latency dominated area and parallel efficiency declines. The same scenario holds for the case of homogeneous turbulence, where the resolution is kept constant and the number of processors is increased.

Still worse is the case of spectral homogeneous turbulence codes where we increase the resolution (in all directions) along with P trying to attack bigger and bigger problems. Then we are moving along the dotted line of slope 1 as our message sizes are proportional to the number of processors ($M_{sz} \propto P$). Time increases very rapidly making communication costs challenge computation costs that scale as $P\log P$.

Looking at Fig. 18 we see both a latency dominated region for small messages and a bandwidth dominated one for large ones. As M_{sz} gets larger the effect of the number of P on communication time becomes less pronounced. So, as long as the message size remains small (as would be in the case of a dot product during a conjucate gradient iteration for example) the communication costs remain under control.

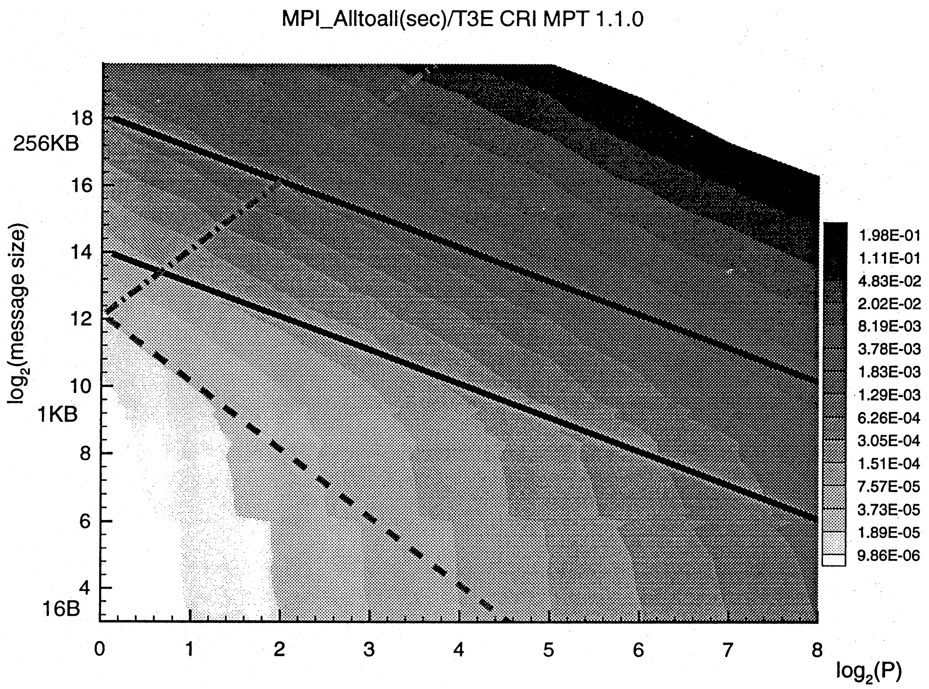


Fig. 17. Contour plot of MPI_Alltoall( ) performance on the Cray T3E. The black parallel lines with slope −1 show how $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ usually scales as the problem size scales with P. The grey dashed line with slope −2 corresponds to $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ or a 3D homogeneous turbulence code with fixed problem size and finally the dotted–dashed line with slope 1 corresponds to a 3D homogeneous turbulence code that scales the problem size with P.

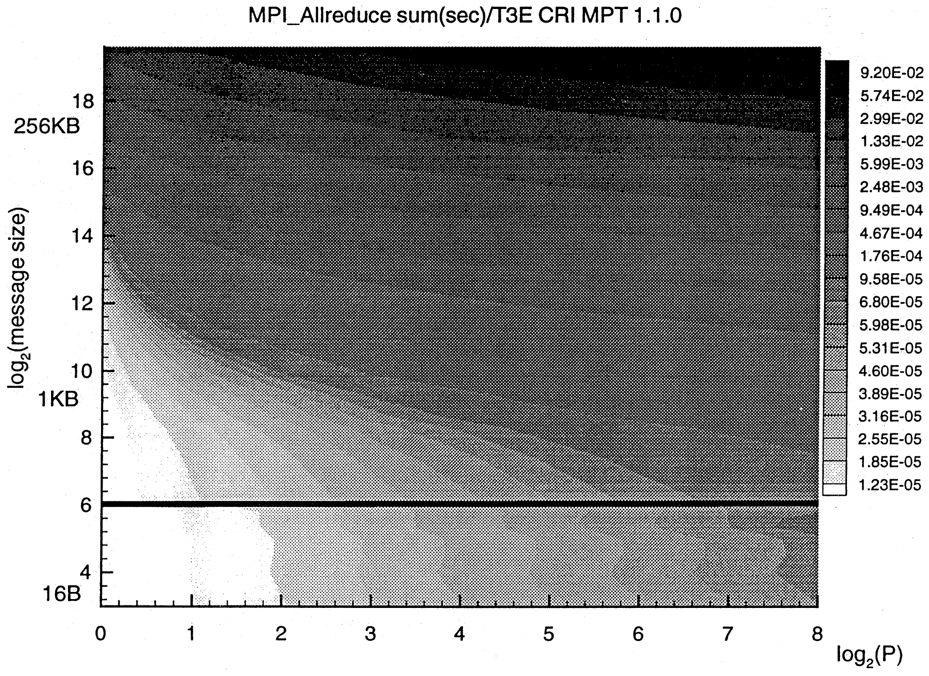


Fig. 18. Contour plot of MPI_Allreduce( ) performance on the Cray T3E. The black horizontal line corresponds to constant message size.

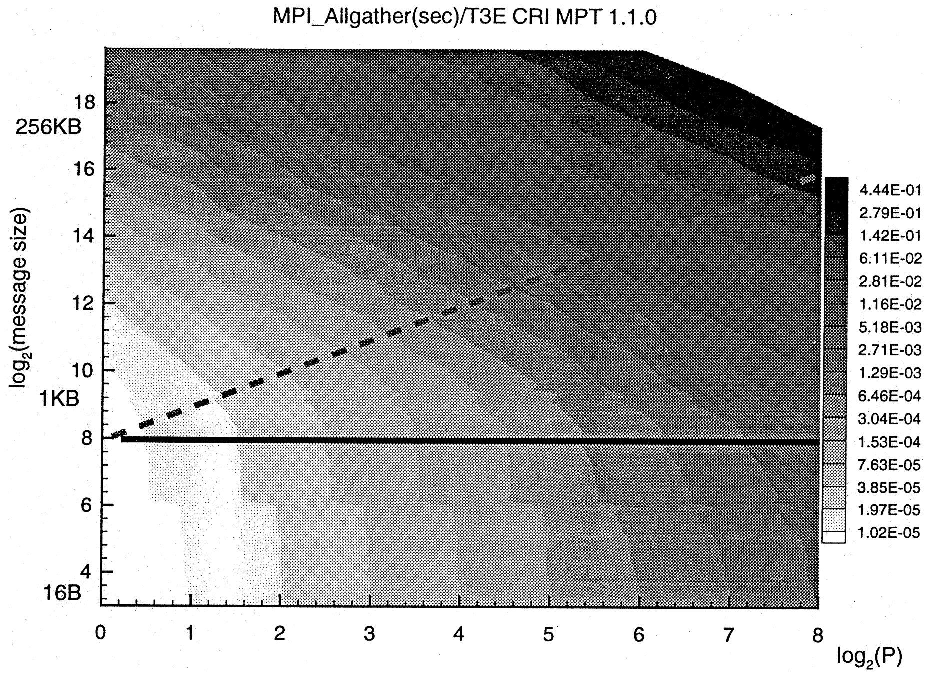


Fig. 19. Contour plot of MPI_Allgather( ) performance on the Cray T3E. The black horizontal line corresponds to constant message size; the grey dashed one to $M_{sz} \propto P$ as N_z increases with P.

Finally Fig. 19 looks similar to Fig. 17. This is not surprising as a Global Exchange can be interpreted as a sequence of Global Gathers. When considering $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$, however, increasing the z-resolution with P, this operation has $M_{sz} \propto P$ (the dotted line) and time increases rapidly with P. This is a disadvantage that comes with using the rotational form of the advection operator. If, instead, only P is increased keeping the same z-resolution, message size remains constant and time increases far less rapidly.

6. Perspective

In this paper we have presented two parallel algorithms for the solution of the incompressible Navier–Stokes equations based on the MPI programming model and have timed and analyzed some collective MPI operations on the Cray T3E. This platform (and its predecessor, the T3D) was one of the first computers to incorporate ‘shared memory’ capabilities [41] albeit in a form requiring explicit management by the user (through use of the ‘SMA’ library). Such an architecture marks the significant transition that is currently taking place, from physically and logically distributed memory in the last decade to distributed shared memory (DSM) in the current and future designs. In the newer architectures (e.g., Convex Exemplar, SGI O2000, upcoming IBM designs) all (local and remote) memory management is handled by hardware, with no extra effort needed from the programmer to access data *physically* stored in a remote node.

In order to understand this trend towards DSM, the programming model it promotes, related hardware and software issues and how they can beneficial in simulating turbulence in a Petaflop computing environment, we preview the basics for upcoming computers and computational tools in the following.

6.1. Hardware

On the hardware side, the hope is that a supercomputer based on a mass produced microprocessor (and, better still, other mass produced sub-components) will get the volume benefits with respect to price and maintenance costs. Moreover, in order to support a shared memory programming model, a DSM architecture (groups of SMP multiprocessor ‘hypernodes’ connected via fast interconnects with all the memory being directly addressable from any processor in any hypernode) in some way or other appears in the roadmaps of all vendors.

At the compute engine level CMOS microprocessor technology is already prevailing. More specifically, for cost reasons even multi-chip vector supercomputer (NEC SX-4 series, Fujitsu VPP-300,700 series, CRI J90 series and future generation of CRI vector machines) as well as mainframes (IBM, Hitachi) have moved to CMOS from expensive ECL technology without taking a significant hit in performance.

Furthermore, DRAM (and derivatives, SDRAM, SLDRAM etc.) memory is getting more inexpensive, relegating the faster but more expensive SRAM type to the role of cache, even on vector supercomputers. 64-bit processors (already most RISC microprocessors) and operating systems enable the addressing of what will be affordable Terabytes of memory within a DSM architecture. More memory, of course, suggests that bigger DNS problems can be attacked using the typically more reliable direct solvers.

Similarly, new disk technology with higher densities along with RAID configurations with striping (for performance) and mirroring (for data integrity) provide a parallel approach to getting very large files and filesystems, both important for large simulations involving hundreds of Gigabytes to Terabytes of data. In addition, I/O bandwidth is increasing due to both faster interfaces (e.g., Fibre Channel) and parallel I/O software support.

Finally, the drive towards DSM architectures has increased the relative importance of low latencies to that of high interconnect bandwidth (of the order of 1 GB/s). While bandwidth is increasing, latencies are not proportionately decreasing. In particular, high bandwidth is still required between hypernodes (the SMP building blocks) within a system (not visible to the user who will only see a single system image). In a message passing context the target of achieving less than 1 μs latency has yet to be reached, whereas in a shared memory context, average remote memory data access even in the best system to day still reaches the 1 μs mark in a system configuration of hundreds of processors [44]. Therefore data locality remains paramount and domain decomposition techniques still bear the greatest promise for DNS.

6.2. Software

The shared memory programming paradigm is seen by most as the easiest way to code in parallel and there are good reasons for it as:

- It appears to be more natural and certainly easier than the message-passing way of partitioning work and manually coordinating it by exchanging messages.
- Automatically parallelizing compilers are getting better every day, although they are still a very long way off being a truly usable tool for supercomputing.
- Especially with the advent of OpenMP [45] *portable* directives based shared memory, parallelism is a reality.
- Explicit threads based shared memory parallelism is also a possibility, and it is currently the best way to get performance. With POSIX threads it is also portable.

Some of the most popular mathematical libraries (e.g., BLAS and LAPACK) already offer *vendor optimized* parallelism for shared memory systems. As object oriented languages find their way in the High Performance Computing world, class libraries that are internally parallel will become more common, providing an easy migration to parallelism. Finally, performance tools will need to become part of everyday use for most users (as opposed to only a few today) as it is unlikely that at least initially one will be able to get decent performance otherwise.

6.3. Migration problems

While the developments on both hardware and software fronts are very exciting and of direct benefit to the DNS of turbulence, there are a number of problems that even a sophisticated simulation scientist will experience during this migration to a new technology.

While peak computer speeds increase with Moore's law most applications see only a small fraction of that peak. Peak performance is achieved through high clock rates and multiple pipelined execution units but these need to be continuously fed with data. The two main problems here are bandwidth and latency of memory accesses.

Data bandwidth is not enough to feed these highly pipelined units when data are not streaming from cache. Once the processor needs to go to main memory for data a 'memory wall' is reached and performance drops appreciably [46,47]. Fixes such as wide memory interfaces (256 or 512 bits) or massively interleaved memory subsystems are not common due to cost reasons. Moreover, memory interface frequency cannot be anywhere near that of the processor – a divisor of that latter frequency is used and with time this disparity increases. Specifically, SMP platforms with more than a couple of processors are forced to go to lower main memory interface frequencies.

Furthermore, memory latency [48] can play a crucial role in some cases (especially for irregular memory accesses) and its value is in the high decades if not hundreds of processor cycles. Processors now have latency tolerant and latency hiding characteristics such as multiple outstanding cache misses and prefetching, and along with compiler optimizations such as speculative code motion the pipelines can be kept working for a little longer. Such techniques do not cure the problem though, they only postpone it.

In contrast, emerging techniques such as Multi-Threaded Processors (Tera) [49] or simultaneous multithreading (SMT) [50] attempt to attack the latency as well as the bandwidth problem using the notions of parallelism at the processor instruction level. However, programming efficiently such different architectures could prove to be very different [51].

While programming in object oriented languages is becoming more widespread, the corresponding current compiler technology is still unable to offer good performance. Shared memory parallel programming may be easier for most users but achieving *high performance* usually needs much more effort in ensuring good data spatial and temporal locality that the code looks very much like a message passing code! Otherwise, the message passing version of the code will be faster despite the fact that it requires a lot of redundant memory copies [52]. Finally, directives, while simple to use, do not currently offer the performance of explicit thread management. Moreover, careful management of thread creation and destruction is needed to avoid excessive overheads, especially for heavyweight threads.

6.4. Applications to dynamic DNS of turbulence

With the new architectural and software features realized in the near future, and particularly with the DSM architecture and shared memory programming model prevailing, it is clear that algorithms such as the spectral/*hp* element method presented here bear great promise for simulating accurately complex geometry and complex physics turbulent flows while advancing DNS as a design tool. The use of multiple threads within shared memory enables the effective implementation of *dynamic* DNS or dDNS: The unstructured grid remains basically the same during the computation but selective *p*-refinement is performed.

General domain decomposition methods, and multi-domain high-order algorithms in particular, can exploit the hybrid nature of the DSM architecture by assigning macro-elements or subdomains to each hypernode. This assignment would follow the same logic that we have been following so far for the distributed memory model (e.g., for $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ a Fourier plane per hypernode, for $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r$3D a compact group of elements per hypernode) to preserve good data locality. The collective communications like MPI_all_to_all( ) could either be eliminated altogether (as they would be redundant) or kept *for performance reasons* but replaced by matrix transpose operations for example – these, if properly implemented, should offer equivalent or even better performance due to lower overheads. Pairwise exchanges (as needed in the GS library employed by $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r$3D) would be replaced by simple (local or remote) memory loads and stores. Moreover, extra parallelism can be used, for example the 2D solution process for each Fourier mode in $\mathcal{N}\varepsilon\kappa\mathcal{T}\alpha r\mathcal{F}$ can now be parallelized within each hypernode.

With the prospect of Petaflop computing not too distant in the future, we can extrapolate the range in Reynolds number that we can achieve. In [2] we argued that a 1024^3 DNS of homogeneous turbulence can be achieved with reasonable turnaround on 1 Teraflop DSM type system. Following similar arguments, we can foresee that DNS of turbulence in the Reynolds number range of $Re = 10^6$ will be achievable for simple geometries, and of the order of $Re = 10^5$ in complex geometries.

dDNS can be used to enhance such anticipated capabilities. Performing dDNS requires primarily on-the-fly *p*-refinement and occasional *h*-refinement, which in turn requires good load balancing. This is difficult to achieve for message passing programs although several efforts to introduce parallel frameworks that handle data migration transparently have progressed significantly [53–55]. In a shared memory environment, such dynamic refinement will be easier with threads picking up more work as the need arises. This will allow for the easier development of more complicated solvers that can change their *h*- or *p*-resolution as the simulation progresses.

However, until the shared memory programming model can provide the performance, message passing (using MPI) should continue being the preferred model for High Performance Computing, even on shared memory machines. A mixed model making use of thread on a local hypernode level and message passing between the multithreaded processes running on each hypernode may be able to offer the best of both worlds for a while at an additional programming complexity cost. As communication bandwidth will be growing faster than latency will be decreasing, algorithms that try to minimize on the number of messages (or remote memory accesses for the shared memory model) at the expense of their size (or even extra memory copies) will be in most cases preferable. This means that even in a shared memory environment it might still be better to do a global transpose for a 3D FFT operation and tests such as those presented in Section 5 will still be relevant.

Acknowledgements

This work was supported partially by DOE, AFOSR, ONR and NSF. Computations were performed at the following computer centers: MHPCC, SDSC, NCSA, NAVO, the AP3000 at Imperial College and at the IBM SP at the Center for Fluid Mechanics at Brown University. We would like to thank Tim Warburton and Ma Xia of our research group at Brown for providing software and figures used in this paper.

References

[1] S.A. Orszag, G.S. Patterson, Numerical simulations of turbulence, in: M. Rosenblatt, C. Van Atta (Eds.), Statistcal Models and Turbulence, Springer, Berlin, 1972, pp. 127–147.
[2] G.E. Karniadakis, S.A. Orszag, Nodes modes and flow codes, Phys. Today 46 (1993) 34.
[3] J. Kim, P. Moin, R. Moser, Turbulence statistics in fully developed channel flow at low Reynolds number, J. Fluid Mech. 177 (1987) 133.
[4] S.L. Lyons, T.J. Hanratty, J.B. McLaughlin, Large-scale computer simulation of fully developed turbulent channel flow with heat transfer, Internat. J. Numer. Methods Fluids 13 (1991) 999.
[5] D.C. Chu, G.E. Karniadakis, A direct numerical simulation of laminar and turbulent flow over riblet-mounted surfaces, J. Fluid Mech. 250 (1993) 1–42.
[6] S. Biringen, R. Reichert, Application of DNS to complex turbulent flows, Turbulence Modeling and Vortex Dynamics, Lecture Notes in Physics, no. 491, Springer, Berlin, 1997.
[7] P.F. Fischer, A.T. Patera, Parallel simulation of viscous incompressible flows, Ann. Rev. Fluid Mech. 26 (1994) 483–528.
[8] M. Rai, P. Moin, Direct numerical simulation of transition and turbulence in a spatially evolving boundary layer, J. Comput. Phys. 109 (2) (1993) 169–192.
[9] C.J. Montgomery, G. Kosály, J.J. Riley, Direct numerical simulation of turbulent reacting flow using a reduced hydrogen–oxygen mechanism, Comb. Flame 95 (1993) 247–260.
[10] S.J. Sherwin, G.E. Karniadakis, A new triangular and tetrahedral basis for high-order finite element methods, Internat. J. Num. Meth. Eng. 38 (1995) 3775.
[11] S.J. Sherwin, G.E. Karniadakis, Tetrahedral *hp* finite elements: Algorithms and flow simulations, J. Comput. Phys. 122 (1995) 191.
[12] S.J. Sherwin, Hierarchical *hp* finite elements in hybrid domains, Finite Elements in Analysis and Design 27 (1997) 109–119.
[13] B. Szabo, I. Babuska, Finite Element Analysis, Wiley, New York, 1991.
[14] L. Demkowicz, J.T. Oden, W. Rachowicz, O. Hardy, Toward a universal *h–p* adaptive finite element strategy, Part 1. Constrained approximation and data structure, Comp. Meth. Appl. Mech. Eng. 77 (1989) 79.
[15] M. Dubiner, Spectral methods on triangles and other domains, J. Sci. Comput. 6 (1991) 345.
[16] S.J. Sherwin, G.E. Karniadakis, A triangular spectral element method; applications to the incompressible Navier–Stokes equations, Comput. Meth. Appl. Mech. Eng. 23 (1995) 83.
[17] B.A. Wingate, J.P. Boyd, Triangular Spectral Element Methods for Geophysical Fluid Dynamics Applications, ICOSAHOM '95, 1995.
[18] T.C.E. Warburton, S.J. Sherwin, G.E. Karniadakis, Spectral basis functions for 2D hybrid *hp* elements, SIAM J. Scientific Comput. 20 (5) (1999) 1671–1695.
[19] T.C.E. Warburton, S.J. Sherwin, G.E. Karniadakis, Unstructured *hp*/spectral elements: Connectivity and optimal ordering, ICES '95, 1995.

[20] T.C.E. Warburton, Spectral/hp methods for polymorphic elements, PhD thesis, Division of Applied Mathematics, Brown University, May 1999.
[21] G.E. Karniadakis, M. Israeli, S.A. Orszag, High-order splitting methods for the incompressible Navier–Stokes equations, J. Comput. Phys. 97 (1991) 414.
[22] R.D. Henderson, G.E. Karniadakis, Unstructured spectral element methods for simulation of turbulent flows, J. Comput. Phys. 122 (1995) 191–217.
[23] C.H. Crawford, C. Evangelinos, D.J. Newman, G.E. Karniadakis, Parallel benchmarks of turbulence in complex geometries, Comput. Fluids 25 (7) (1996) 677–698.
[24] B. Smith, P. Bjorstad, W. Gropp, Domain Decomposition. Parallel Multilevel Methods for Elliptic Differential Equations, Cambridge University Press, Cambridge, 1996.
[25] C. Evangelinos, G.E. Karniadakis, Parallel CFD benchmarks on Cray computers, Parallel Algorithms Appl. 9 (1996) 273–298.
[26] M. Frigo, S.G. Johnson, The Fastest Fourier Transform in the West, Technical Report MIT-LCS-TR-728, MIT, MIT Laboratory for Computer Science, 545 Technology Square NE43-203, Cambridge MA 02139, September 1997.
[27] W.P. Petersen, An FFT Benchmark on various Workstations, Technical Report, Interdisziplinäres Projektzentrum für Supercomputing, ETH, Zürich, wpp@ips.id.ethz.ch.
[28] P. Beaudan, P. Moin, Numerical experiments on the flow past a circular cylinder at subcritical Reynolds number, Technical Report, Report No. TF-62, Stanford University, Stanford, CA 94305, 1994.
[29] M. Xia, G.E. Karniadakis, The spectrum of the turbulent near-wake: A comparison of DNS and LES, in: Advances in DNS/LES, Proceedings of the first AFOSR International Conference on DNS/LES, Greyden Press, 1997, pp. 129–136.
[30] D.J. Doorly, J. Peiró, S.J. Sherwin, O. Shah, C.G. Caro, M. Tarnawski, M. MacLean, C. Dumoulin, L. Axel, Helix and model graft flows: MRI measurements and CFD simulations, ASME, 1997, Biomedical Symposium, Colorado.
[31] S.J. Sherwin, O. Shah, D.J. Doorly, J. Peiró, Y. Papaharilaou, N. Watkins, C.G. Caro, C.L. Dumoulin, The influence of out-of-plane geometry on the flow within a distal end-to-side anastomosis, ASME J. Biomechanical Engrg. 122 (2000) 1–10.
[32] G. Karypis, V. Kumar, Multilevel *k*-way Partitioning Scheme for Irregular Graphs, Technical Report, Department of Computer Science, University of Minnesota, 1995.
[33] B. Hendrickson, R. Leland, An improved spectral graph partitioning algorithm for mapping parallel computations, SIAM J. Sci. Stat. Comput. 16 (2) (1995) 452–469.
[34] C. Walshaw, M. Cross, M. Everett, Parallel dynamic graph partitioning for adaptive unstructured meshes, J. Par. Dist. Comput. 47 (2) (1997) 102–108.
[35] A. Gupta, Fast and effective algorithms for graph partitioning and sparce-matrix ordering, IBM J. Res. Develop. 41 (1/2) (1997) 171–183.
[36] H.M. Tufo, Algorithms For Large Scale Parallel Simulation of Unsteady Incompressible Flows in 3D Complex Geometries, PhD thesis, Brown University, 1998.
[37] J. Peiro, A.I. Sayma, A 3-D unstructured multigrid Navier–Stokes solver, in: K.W. Morton, M.J. Baines (Eds.), Numer. Meth. Fluid Dynamics V., Oxford University Press, Oxford, 1995.
[38] J. Peraire, J. Peiro, K. Morgan, Multigrid solution of the *D* compressible Euler equations on unstructured tetrahedral grids, Internat. J. Numer. Meth. Engrg. 36 (1993) 1029–1044.
[39] J. Dongarra, T. Dunigan, Message-Passing Performance of Various Computers, Technical Report CS-95-299, University of Tennessee, Computer Science Department, July 1995.
[40] G.A. Abandah, Modelling the Communication and Computation Performance of the IBM SP2, Technical Report, Advanced Computer Architecture Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109-2122, May 1995.
[41] S.L. Scott, Synchronization and communication in the T3E multiprocessor, in: Proceedings of ASPLOS-VII, Cambridge, MA, October 1996.
[42] R. Reussner, Measurements of SKaMPI, Version 1.13, Special Karlsruhe MPI-Benchmark of a IBM RS 6000 SP, Technical Report, University of Karlsruhe, Department of Informatics, Germany, March 1998.
[43] M. Mueller, Measurements of SKaMPI, Version 1.15, Special Karlsruher MPI-Benchmark of a T3E-900, Technical Report, University of Karlsruhe, Department of Informatics, Germany, March 1998.
[44] C. Hristea, D. Lenoski, Measuring Memory Hierarchy Performance of Cache-Coherent Multiprocessors using Micro Benchmarks, in: Proceedings of the Sc'97 IEEE and ACM, November 1997.
[45] OpenMP: A Proposed Industry Standard API for Shared Memory Programming, October 1997.
[46] J.D. McCalpin, Memory Bandwidth and Machine Balance in Current High Performance Computers, IEEE Technical Committee on Computer Architecture newsletter, December (to appear).
[47] J.D. McCalpin, Sustainable memory bandwidth in current high performance computers, submitted to SIGMETRICS'96, October 1995.
[48] A. Saulsbury, F. Pong, A. Nowartzyk, Missing the Memory Wall: The Case for Processor/Memory Integration, in: Proceedings of the 23rd Annual International Symposium on Computer Architecture, ACM, June 1996.
[49] G. Alverson, P. Briggs, S. Coatney, S. Kahan, R. Korry, Tera Hardware–Software Cooperation, Technical Report, Tera Computer Company, 1997.
[50] S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo, R.L. Stamm, D.M. Tullsen, Simultaneous multithreading: a platform for next-generation processors, IEEE Micro (1997) 12–18.
[51] J.L. Lo, S.J. Eggers, H.M. Levy, S.S. Parekh, D.M. Tulsen, Turning Compiler Optimizations for Simultaneous Multithreading, in: Proceeding of the Micro-30, IEEE, December 1997.

[52] R.A. Fiedler, Optimization and scaling of shared-memory and message-passing implementations of the ZEUS hydrodynamics algorithm, in: Proceedings of the SC'97, IEEE and ACM, November 1997.
[53] C. Walshaw, M. Cross, Load-balancing for parallel adaptive unstructured meshes, in: Proceedings of the International Conference on Numerical Grid Generation in Computational Field Simulation, Greenwich, UK, 1998.
[54] L. Oliker, R. Biswas, Efficient Load Balancing and Data Remapping for Adaptive Grid Calculations, in: Proceedings of the Ninth ACM Symposium on Parallel Algorithms and Architectures, ACM, June 1997.
[55] R. Biswas, R.C. Strawn, Tetrahedral and hexahedral mesh adaptation for CFD problems, Appl. Numer. Math. (to appear).