INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2006; **52**:591–615
Published online 16 February 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/fld.1189

Discretization and parallel performance of an unstructured finite volume Navier–Stokes solver

S. A. Mohsen Karimian[‡] and Anthony G. Straatman[*,†]

Department of Mechanical and Materials Engineering, The University of Western Ontario, London, Ont., Canada N6A 5B9

SUMMARY

The discretization, parallelization and performance of an implicit, unstructured, time-dependent Computational Fluid Dynamics code is described. A detailed description is provided of the improvements made on second-order accurate tools for spatial interpolation and gradient calculation to discretize the Navier–Stokes equations in an unstructured framework. The main goal in the development of the discretization tools was to ensure a scalable and accurate parallel code. The performance of the discretization tools has been validated using standard bench-mark problems for non-uniform, non-orthogonal grids. Parallelization of the code is done within the PETSc framework using a single-program-multiple-data (SPMD) parallelization model. The resulting parallel code is shown to scale linearly within the limit of the available number of processors. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: finite-volume method; parallel computing; unstructured grid; incompressible flow

1. INTRODUCTION

With the ever-increasing development of computer hardware and the availability of parallel computing platforms, scientists and engineers are able to investigate more and more complex problems in solid and fluid mechanics. In thermofluids, scientists are being called upon to solve large-scale problems involving single- and multi-phase flows, turbulence, heat transfer and chemical reaction for the purpose of design, development and process control. As such, continued development of physical models, algorithms and parallelization techniques is of immediate relevance and application in science and engineering. To this end, the present paper describes a three-dimensional Computational Fluid Dynamics (CFD) code developed

*Correspondence to: A. G. Straatman, Department of Mechanical and Materials Engineering, The University of Western Ontario, London, Ont., Canada N6A 5B9.
†E-mail: astraatman@eng.uwo.ca
‡E-mail: skarimia@uwo.ca

Contract/grant sponsor: Natural Science and Engineering Research Council
Contract/grant sponsor: SHARCNET

Received 27 May 2005
Revised 3 October 2005
Accepted 26 December 2005

for use in a parallel computing environment. The methodologies used for the discretization and parallelization are chosen/devised on the basis that the code is to be used to investigate engineering flows using a transient Reynolds-averaged-Navier–Stokes (RANS) or VeryLarge-Eddy-Simulation (VLES) approach. The present CFD algorithm is based on an unstructured finite-volume discretization and coded to operate in a distributed-memory environment such as that provided by Shared Hierarchical Academic Research Computing Network (SHARCNET) [1]. The main focus of the work described herein is on basic elements of the discretization that affect the parallel performance and on a procedure that enhances the parallel performance of the resulting code. A brief review is given to describe the origins of the methods proposed for the CFD algorithm.

Three main practices are currently used for solving the governing transport equations for fluid and energy flow in multi-purpose CFD codes: finite-element methods, finite-volume methods, and control-volume based finite element methods (CVFEM), which is a combination of the two former methods. CVFEM [2, 3] use vertex-centred data structures and have the flexibility of the finite-element formulation for discretization on unstructured grids. Therefore, most of the legacy unstructured CFD codes that have been parallelized implement CVFEM. Examples include Tai *et al.* [4], FUN3D [5, 6] (parallelized by Gropp *et al.* [7]) and TAU [8] (parallelized as a part of MEGAFLOW [9]), among many others. Recent efforts also describe the implementation of CVFEM and vertex-centred data structures in parallel unstructured CFD codes, e.g. Reference [10]. In terms of basic structure, two main factors distinguish the CVFEM methods from classical finite-volume methods: (1) In CVFEM, the finite-volume database for which the linear system is constructed and the actual geometric (finite-element mesh) database used to construct the linear system are different and hence need to be managed in domain decomposition [11] and; (2) The computational molecule resulting from CVFEM is generally larger than that resulting from classical FV methods. A computational molecule, for a given control volume (P), is a group of control volumes in the geometric vicinity of the control volume (P), including itself, whose implicit coefficient in the discrete form of the general transport equation for the control volume (P) is non-zero. The size of the computational molecule is an important parameter in CFD in general, since it affects the sparsity of the matrix system that needs to be solved. In parallel coding, the computational molecule is even more important because it affects the amount of overlap required between sub-domains to ensure proper communication between processors. Classical finite-volume (FV) methods have a relatively simple data structure and smaller computational molecule and are a suitable choice for parallel unstructured implicit CFD codes, e.g. Cobalt [12]. However, there are fewer parallel unstructured CFD codes that implement classical finite-volume methods, due partially to difficulties that arise with FV methods in interpolation and implicit gradient field generation for the primitive variables in the absence of line structure. These basic building blocks of a CFD algorithm affect diffusion and convection modelling, source term evaluation, and have an impact on the efficiency of the resulting parallel algorithm, which is of specific interest here. To achieve the best scalability in the parallelization and minimal communication, the interpolation techniques that are used to calculate the value of variables at integration points must be as local as possible. In addition, to preserve memory and to minimize parallel communication, methods are required that do not rely on a large amount of data from the previous iteration or time-step.

Cobalt [12] generated an implicit gradient term using a least squares method to obtain weighting coefficients for all required nodes. This method requires a QR factorization and a

local neighbour search in each control volume and leads to a large computational molecule with the same unfavourable features mentioned above for CVFEM. Moreover, the least squares method is not second-order and is a grid-oriented method and, therefore, tends to be unstable for grids with high aspect ratio. In the method suggested by Lien [13], an artificial line structure is generated by determining the gradient vector at each control surface based on the values of the variable at the centre of straddling control-volumes and the vertices of the control surface. This method also requires a local neighbour search, which is an expensive process, particularly in three dimensions, and generates a large arbitrary computational molecule, leading to the same problems mentioned above. Demirdžić and Muzaferija [14] proposed a second-order polyhedral approach with a compact stencil to overcome the problem with the implicit gradient field faced in the FV method. Their method was followed by Mathur and Murthy [15] and Basara [16], who proposed a simpler version which was not exactly second-order for highly non-uniform grids. Their method is based on a deferred-correction technique with three phases: implicit flux calculation, linear system solution and gradient reconstruction. The implementation of the method in a parallel code is straightforward, with one exception: the second-order implementation of the implicit gradient formula requires a correction term with second derivatives of the primitive variables that in turn requires expensive calculations and, even worse, an increase in the communication overhead of three times in the gradient reconstruction phase. In this paper, an equally accurate and stable approach is proposed that maintains second-order accuracy and does not require the second derivative field.

Parallelization of the present CFD algorithm is carried out using a single-program-multiple-data (SPMD) parallelization model. A message passing method is suitable for a cluster of distributed-memory high performance computers, such as the environment described above. The parallel library PETSc [17] is used to obtain parallel tools for the low-level distributed data structures and message passing and for high-level solvers. The concept of inexact Newton–Krylov method has been adopted in the non-linear iteration of each time-step to speed up the linearization process.

The remainder of the paper describes the unique discretization and parallelization aspects of the proposed CFD code. The acronym chosen for the code is ParTISUN: *Par*allel *T*ime-accurate *I*mplicit *S*olver for *U*nstructured *N*avier–Stokes. The paper is organized such that the mathematical formulation is given first, followed by the discretization details and the parallelization of the algorithm. Emphasis is placed on the novelty developed in the discretization section. A final section provides validation for the code and describes the computational efficiency of the parallel algorithm.

2. MATHEMATICAL FORMULATION

Consider an arbitrary fixed control volume $\Omega \subset R^3$, of volume V and a piecewise smooth boundary $\partial\Omega$, with unit normal surface vector $\hat{n}$ pointing outwards, occupied by continuum (in our case a fluid). The conservation of mass, linear momentum and thermal energy are used in their integral form. To complete the formulation, we assume an incompressible Newtonian fluid and use the constitutive equations

$$\rho = \text{const.}, \quad e = c_p T \tag{1}$$

where ρ is the density, e is the specific energy and T is the absolute temperature of the fluid.

Substitution of the constitutive equations into the integral governing equations gives the set of equations that are solved in the present work

$$\int_{\partial\Omega} \rho \mathbf{V} \cdot \hat{n}\, \mathrm{d}S = 0 \tag{2}$$

$$\int_{\Omega} \rho \partial_t \mathbf{V}\, \mathrm{d}V + \int_{\partial\Omega} \hat{n} \cdot (\rho \mathbf{V}\mathbf{V})\, \mathrm{d}S - \int_{\partial\Omega} \hat{n} \cdot (\mu \nabla \mathbf{V})\, \mathrm{d}S = -\int_{\partial\Omega} P\hat{n}\, \mathrm{d}S \tag{3}$$

$$\int_{\Omega} \rho \partial_t T\, \mathrm{d}V + \int_{\partial\Omega} \hat{n} \cdot (\rho \mathbf{V} T)\, \mathrm{d}S - \int_{\partial\Omega} \hat{n} \cdot \left(\frac{k}{c_p} \nabla T\right) \mathrm{d}S = 0 \tag{4}$$

where $\mathbf{V}$ is the velocity vector, $\mathrm{d}S$ is the surface differential, $\mathrm{d}V$ is the volume differential, μ is the viscosity, k is the heat conductivity coefficient and c_p is the specific heat. It is clear that the equations for conservation of momentum and energy follow the generic transport equation

$$\int_{\Omega} \rho \partial_t \phi\, \mathrm{d}V + \int_{\partial\Omega} \hat{n} \cdot (\rho \mathbf{V} \phi)\, \mathrm{d}S - \int_{\partial\Omega} \hat{n} \cdot (\Gamma^{\phi} \nabla \phi)\, \mathrm{d}S = \int_{\Omega} Q_{\phi}^{V}\, \mathrm{d}V + \int_{\partial\Omega} \hat{n} \cdot \mathbf{Q}_{\phi}^{S}\, \mathrm{d}S \tag{5}$$

where ϕ is the generic transport variable, Γ^{ϕ} is the coefficient for diffusion of the generic transport variable, Q_{ϕ}^{V} is the volumetric source term and $\mathbf{Q}_{\phi}^{S}$ is any additional surface term acting as a source to drive ϕ. Therefore, here and henceforth, rather than specifically discussing momentum and energy equations, we simply refer to the generic transport equation (5). It is good to mention here that the transport equations for turbulence can also be expressed in the form of equation (5). The solution of the governing equation set requires the imposition of appropriate boundary and initial conditions, as will be described for the particular applications considered.

3. DISCRETIZATION METHOD

The coupled transport equations combined with boundary and initial conditions are solved using finite-volume discretization, whereby the domain of interest is divided into a number of non-overlapping finite-volumes ΔV_i that comprise Ω. The transport equations are integrated discretely over each finite-volume, face-by-face, to form a linear system of equations (implicit flux calculation). The resulting linear system is solved using a preconditioned Krylov subspace method, e.g. BiCGSTAB [18] or GMRER [19]. Since most applications of interest are geometrically complex, an unstructured finite-volume framework is used. Without compromising the generality of the method, we consider the finite-volumes to be arbitrary hexahedra; all interpolation and numerical techniques used for this type of volume can be applied to tetrahedra, wedges or a combination of these patterns. In many of the resulting terms, two important discretization tools, i.e. spatial interpolation and gradient calculation, are required. As suggested in the introduction, these basic elements of discretization are critical to the accuracy and efficiency of the resulting code.

3.1. Spatial interpolation

Figure 1 gives an illustration of two control volumes, p and u. For the sake of simplicity and without losing generality, the control volumes are shown in the form of two-dimensional quadrilaterals. In order to calculate the surface integrals in Equation (5) at each face of the control volume, it is necessary to interpolate variables from nodes, where the solution is obtained, to integration points. Interpolation is required to express the generic variable ϕ and its gradient $\nabla\phi$ at the integration point ip in terms of the values of the variable and its derivatives at the centre of the control volumes p and u. The formulae for interpolation must be implemented implicitly and then improved to the desired accuracy (if necessary) via explicit deferred-correction [20] in the implicit flux calculation phase.

Since symmetric formulae are most stable for interpolation, all calculations and interpolations in this work include a second-order symmetric part that is basically an arithmetic average of the variables between the nodes for the midpoint m

$$\phi_m^{\mathrm{so}} = \tfrac{1}{2}(\phi_p + \phi_u) \tag{6}$$

and a correction part which enhances the accuracy of the calculation to second-order for the integration point ip, in case the midpoint does not coincide with the integration point, as suggested by Demirdžić *et al.* [14]

$$\phi_{\mathrm{ip}}^{\mathrm{so}} = \phi_m^{\mathrm{so}} + \mathbf{R} \cdot (\nabla\phi_m^{\mathrm{so}})$$

or

$$\phi_{\mathrm{ip}}^{\mathrm{so}} = \tfrac{1}{2}(\phi_p + \phi_u) + \tfrac{1}{2}\mathbf{R} \cdot (\nabla\phi_p + \nabla\phi_u) \tag{7}$$

where $\mathbf{R}$ is the displacement vector of ip relative to the midpoint m (Figure 1) and the superscript so indicates *second-order*. The second term in Equation (7) is always treated explicity, whereas the first term is treated either implicitly or explicitly, depending on the implementation.

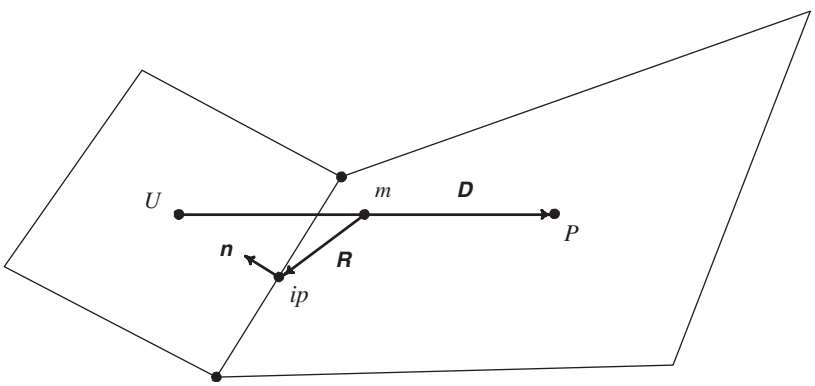

Figure 1. Two control volumes straddling a control surface.

Also, one can directly use the symmetric formula suggested by Demirdžić *et al.* with two one-sided interpolations to determine ϕ_{ip} for a slightly higher computational cost

$$\phi_{\text{ip}} = \tfrac{1}{2}(\phi_p + \phi_u) + \tfrac{1}{2}[\mathbf{R}_{p\to\text{ip}} \cdot \nabla\phi_p + \mathbf{R}_{u\to\text{ip}} \cdot \nabla\phi_u] \tag{8}$$

where $\mathbf{R}_{p\to\text{ip}}$ and $\mathbf{R}_{u\to\text{ip}}$ are displacement vectors of ip relative to p and u, respectively. These types of interpolations give the maximum accuracy with the available information, however Equation (7) is less likely to result in an extremal value at the integration point, compared to Equation (8). This is because the correction term in Equation (7) is based on a symmetric evaluation of the gradient at the midpoint, rather than averaging two one-sided evaluations of the correction as that in Equation (8), and hence suppressing the effect of any extreme change of the gradient field.

3.2. Gradient formulation

In the absence of line structure in an unstructured grid, discretization of diffusion terms and every other term containing a gradient of a generic variable ϕ requires special consideration. An approach appropriate for arbitrary unstructured finite-volume methods was first introduced by Demirdžić and Muzaferija [14] and then implemented in slightly different forms by Mathur and Murthy [15] and Basara [16]. This family of methods contain an implicit part and a deferred-correction explicit part based on the updated gradient field.

In order to discretize the diffusion term in the generic transport equation, Equation (5), a semi-implicit discrete formulation for the gradient operation must be provided. Here, we follow the approach of Demirdžić and Muzafarija. Referring to Figure 1, a second-order formula

$$\nabla\phi_{\text{ip}} = \frac{\phi_p - \phi_u}{\mathbf{D} \cdot \hat{n}}\,\hat{n} + \left(\overline{\nabla\phi}_{\text{ip}} - \frac{\overline{\nabla\phi}_m \cdot \mathbf{D}}{\mathbf{D} \cdot \hat{n}}\hat{n}\right) \tag{9}$$

is used to determine the gradient at the integration point ip, where $\mathbf{D}$ is displacement vector of p relative to u,

$$\overline{\nabla\phi}_m = \tfrac{1}{2}(\nabla\phi_p + \nabla\phi_u) \tag{10}$$

and $\overline{\nabla\phi}_{\text{ip}}$ is a second-order approximation of the gradient at the integration point. Note that in Equation (9) the denominator of the first term, which is treated implicitly, is smaller than its counterpart in Reference [14] for highly non-orthogonal grids. This yields a larger diagonal coefficient for the matrix of the equations and hence, acts as an automatic numerical stabilizer when dealing with skewed grids. Moreover, it is more sensible from a mathematical point of view.

While Basara has proposed a weighted average of $\nabla\phi_p$ and $\nabla\phi_u$ as an approximation for $\overline{\nabla\phi}_{\text{ip}}$, i.e.

$$\overline{\nabla\phi}_{\text{ip}} = f\nabla\phi_p + (1-f)\nabla\phi_u \tag{11}$$

where f is an appropriate weight function, e.g. inverse distance function, and Mathur *et al.* simply assumed that $f = 0.5$, Demirdžić *et al.* have used Equation (8) to determine $\overline{\nabla\phi}_{\text{ip}}$. The last method is second-order even if the grid is highly non-uniform, i.e. when the vector $\mathbf{R}$ in Figure 1 is large. However, to be able to implement this formula one needs to calculate and

save the second derivatives, $\nabla\nabla\phi$, for each generic variable ϕ at each control volume. It is also necessary to update the values of these matrices for the ghost control volumes in every iteration. This means increasing the communication overload from 4 double-precision values to 13 double-precision values per degree of freedom per ghost (or overlapping) control volume. To overcome this problem a new correction term to Equation (11) is proposed. Regarding the integral form of the generic transport equation (and the modified form of the continuity equation which will be discussed later), $\nabla\phi$ always appears in the form of $\hat{n}\cdot\nabla\phi$. Using Equation (7), one could say

$$\hat{n}\cdot\nabla\phi_{\mathrm{ip}} = \hat{n}\cdot[\nabla\phi_m + \mathbf{R}\cdot\nabla\nabla\phi_m] \tag{12}$$

For a smooth function of $\phi(t,x_i)$, one could switch the vectors ∇ and $\hat{n}$ and obtain

$$\hat{n}\cdot\nabla\phi_{\mathrm{ip}} = \hat{n}\cdot\nabla\phi_m + \mathbf{R}\cdot\left.\frac{\partial\nabla\phi}{\partial n}\right|_m \tag{13}$$

In a compressible flow where no shockwave occurs in the flow field, the gradient of all primitive variables are defined and are continuous for every point of the flow field. Therefore, all primitive variables are smooth functions of space.

The orthogonal decomposition of $\frac{\partial\nabla\phi}{\partial n}|_m$ along $\mathbf{D}$ and normal to $\mathbf{D}$ yields

$$\left.\frac{\partial\nabla\phi}{\partial n}\right|_m = \left[\hat{d}\cdot\left.\frac{\partial\nabla\phi}{\partial n}\right|_m\right]\hat{d} + \left[\hat{e}\cdot\left.\frac{\partial\nabla\phi}{\partial n}\right|_m\right]\hat{e} \tag{14}$$

or

$$\left.\frac{\partial\nabla\phi}{\partial n}\right|_m = \frac{(\nabla\phi_p - \nabla\phi_u)}{\mathbf{D}\cdot\hat{n}} + \left[\hat{e}\cdot\left.\frac{\partial\nabla\phi}{\partial n}\right|_m\right]\hat{e} \tag{15}$$

where $\hat{d}$ is the unit vector along $\mathbf{D}$ and $\hat{e}$ is an appropriate unit vector normal to $\mathbf{D}$ in general three-dimensional space. The first term of the right-hand side of Equation (15) is calculated using the available information, i.e. the values of the gradient of the generic variable ϕ at the centre of each control volume. The calculation of the second term still requires $\nabla\nabla\phi$. However, for closely orthogonal grids, in which the relation $\mathbf{D} \parallel \hat{n}$ is approximately valid, the second term is negligible. It is worth mentioning here that even for grids with arbitrary polyhedral control volumes with high grid quality, e.g. Delauney triangles or tetrahedra, it is most likely that the above-mentioned condition applies. Therefore,

$$\left.\frac{\partial\nabla\phi}{\partial n}\right|_m \approx \frac{(\nabla\phi_p - \nabla\phi_u)}{\mathbf{D}\cdot\hat{n}} \tag{16}$$

This suggests that the following approximation is the most accurate formula using the available information:

$$\overline{\nabla\phi}_{\mathrm{ip}} = \frac{1}{2}(\nabla\phi_p + \nabla\phi_u) + \mathbf{R}\cdot\frac{(\nabla\phi_p - \nabla\phi_u)}{\mathbf{D}\cdot\hat{n}}\,\hat{n} \tag{17}$$

Bear in mind that the cross diffusion part of the correction term, i.e. the projection of $\nabla\nabla\phi$ on the control surface, is neglected not because it is negligible, but because its inner product with $\hat{n}$ is always zero and the gradient term always appears in the form of inner product with $\hat{n}$. It is worth mentioning that the size of the correction terms in Equations (7) and (17)

depend on the size of **R**. In other words, in an irregular grid where the mid-point m does not coincide with the integration point ip (the size of **R** is not zero), the correction terms in these two equations are activated to maintain the accuracy.

3.3. Gradient reconstruction

To implement Equations (7), (9) and (17) in the discretization process, we need to reconstruct the gradient field of the generic variable ϕ. There are two methods available for determining the gradient fields: the least squares method or implementation of the divergence theorem.

The least squares method is based on the idea that a correct gradient field at a point p in Figure 2 must satisfy the following equation for every point u in its close vicinity:

$$\nabla\phi_p \cdot \mathbf{D}_i = \phi_{u_i} - \phi_p \tag{18}$$

For an N-edged polyhedral control volume, the above equation forms N constraints on the gradient field $\nabla\phi_p$. The minimum number of constraints is always larger than the number of dimensions, i.e. three constraints for triangular grids in two dimensions and four for tetrahedral grids in three dimensions. This suggests use of the least squares method to determine the gradient field. This method, however, is not appropriate for control volumes of high aspect ratio, which occur in near-wall regions of viscous flows, since the constraints are highly dependent on the shape of the control volumes [21, 22].

Another method is based on the implementation of the divergence theorem and the mean-value theorem

$$\nabla\phi_p = \frac{\sum_{\text{ip}=1}^{n} \phi_{\text{ip}} \hat{n}_{\text{ip}} \, \mathrm{d}S_{\text{ip}}}{\Delta V} \tag{19}$$

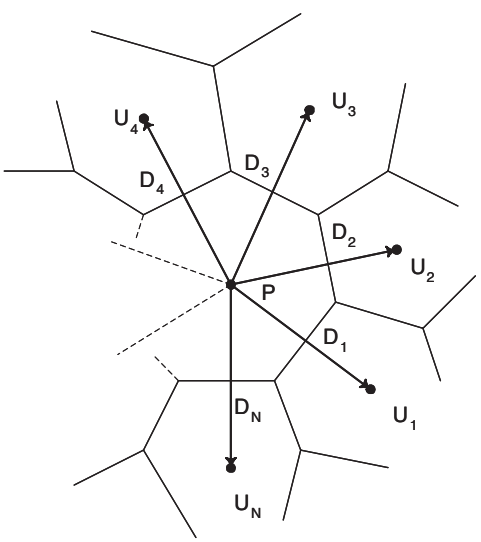

Figure 2. A control volume with the neighbours in its vicinity.

where ϕ_{ip} is calculated using Equation (7) and $\mathrm{d}S_{\text{ip}}$ is the area of the control surface. This method is independent of the shape of the control volume. However, in this equation $\nabla\phi$ is a function of ϕ_{ip}. On the other hand, the calculation of ϕ_{ip} using Equation (7) is based on $\nabla\phi_p$ and $\nabla\phi_u$. Therefore, to ensure good accuracy, $\nabla\phi$ must be calculated in an iterative manner. Experience shows that for a smooth field, two to four iterations give satisfactory accuracy.

3.4. Discretization procedure

As mentioned in the introduction to this section, an unstructured finite-volume procedure was used to discretize the governing equations for the three-dimensional code described herein. Application of this procedure over a general finite-volume results in mass equations of the form

$$\sum_{\text{ip}=1}^{n} \dot{m}_{\text{ip}} = 0 \tag{20}$$

and transport equations of the form:

$$\rho_{\text{cv}} \frac{\partial \phi_{\text{cv}}}{\partial t} \Delta V + \sum_{\text{ip}=1}^{n} \dot{m}_{\text{ip}} \phi_{\text{ip}} - \sum_{\text{ip}=1}^{n} \Gamma_{\text{ip}}^{\phi} \nabla \phi_{\text{ip}} \cdot \hat{n}_{\text{ip}} \, \mathrm{d}S_{\text{ip}} = Q_{\phi}^{\text{cv}} \Delta V + \sum_{\text{ip}=1}^{n} \mathbf{Q}_{\phi}^{\text{ip}} \cdot \hat{n}_{\text{ip}} \, \mathrm{d}S_{\text{ip}} \tag{21}$$

Here, ΔV is the volume of the control volume, a sub or superscript ip indicates the values at the integration points and a sub or superscript cv indicates the values at the centre of the discrete finite volume. As can be seen, the discrete counterpart of Equations (3) and (4) fits into the discrete form of the generic transport Equation (21) by replacing ϕ with $\mathbf{V}$ or T, Γ^{ϕ} with μ or k/c_p, Q_{ϕ}^{cv} with zero and finally $\mathbf{Q}_{\phi}^{\text{ip}}$ with $-P_{\text{ip}}\mathbf{I}$ or zero, respectively, where $\mathbf{I}$ is the Kronecker delta.

The three terms on the left-hand side of Equation (21) are called transient, convection and diffusion terms, respectively, and the right-hand side of this equation consists of volumetric and facial source terms. Transient terms have been discretized using the implicit three time-level approach described by Ferziger and Perić [23]. Using this method, there is no numerical restriction on the time-step size and thus, the time-step size can be set based on the desired resolution of the physical problem being considered.

Gradients for the diffusion terms have been discretized using the method described in Section 3.2. Using Equation (9), each gradient term is cast as the sum of a dominant implicit term and an explicit correction, which ensures second-order accuracy. Convective fluxes are approximated using one of a number of Total Variation Diminishing (TVD) [24] schemes. The implementation of the TVD schemes follows the approach described by Darwish and Moukalled [25], which is based on the same computational molecule considered herein, i.e. only nodes adjacent to the face under consideration are used. The method is second-order and is implemented using the deferred-correction approach [20] whereby a simple upwinding scheme forms the implicit part, which is then corrected using either the well-known central difference scheme (CDS) [23], SUPERBEE [26] or MUSCL [27].

While volumetric source terms are simply integrated over the control volume in question, the pressure source term is treated as a facial force and integrated face-by-face to obtain implicit and explicit fragments to be added into the discrete equations. In the present treatment, the mass and momentum equations are solved as a direct coupled set of equations and thus, coupling between the pressure and velocity fields is implicit in the solution procedure.

The scheme proposed by Rhie and Chow [28] is used to maintain coupling between the pressure and velocity fields during the solution procedure.

The discretization algorithm consists of a main inner loop which cycles through all control volumes, and then within each control volume computes the source terms and fluxes face-by-face to form the complete discrete equation. If a face is adjacent to a boundary, the appropriate condition is imposed by integrating the effect of the boundary condition into discrete equation for the volume. This is done by writing a simplified discrete equation for the variable at the boundary in terms of the integration point and the node at the control volume centre and then absorbing the boundary equation into the discrete equation for the control volume.

4. PARALLEL IMPLEMENTATION

The code is written in Fortran 90 to take advantage of the flexibility of this language in terms of runtime memory allocation, replacement of array-based DO-loops with simpler general array syntaxes and better compile-time optimizations. Moreover, the code is constructed on the PETSc framework using a single-program-multiple-data (SPMD) parallel model to run on distributed memory parallel computer architectures. PETSc, a portable extensible toolkit for scientific computation, is a suite of data structures and routines that provide the building blocks for the implementation of large-scale application codes on parallel (and serial) computers [17]. PETSc has been implemented as a sparse matrix solver as well as an interface library layer for MPI [29], that is to say the message passing is performed by PETSc library subroutines.

In a SPMD parallel model, each process runs the same program to perform computations on its own subset of the global finite-volume database. The problem domain is divided into a set of N subdomains (domain decomposition). Figure 3 shows a schematic of the domain decomposition and details of the control volumes in the vicinity of the border of two sub domains. To solve the governing equations, a global set of linear equations is constructed. The global linear system is divided into N portions of rows. In the implicit flux calculation phase, each process constructs and saves the equations for its own set of local control volumes. As Figure 3 suggests, an equation in one process may have an implicit coefficient for a control volume, as a neighbour, located in another process. The constructed global linear system is solved using a preconditioned Krylov subspace method.

The implicit coefficients in the discrete form of the continuity equations for a control volume are calculated based on the latest values of the diagonal coefficients of the discrete form of the momentum equations for that control volume and its neighbours (see Reference [28]). Moreover, the explicit correction terms in the discrete equations of a control volume are functions of the old values of the primitive variables and their first derivatives of that control volume and its neighbours. Therefore, each subdomain keeps an up-to-date virtual copy, as a ghost control volume, of the control volumes belonging to other subdomains and located at its borders with those subdomains. The main purpose of defining ghost control volumes is to make a communication chain between subdomains so that a flow of information can be maintained between the control volumes in the vicinity of the subdomain boundaries. Here it is obvious that a large computational molecule would lead to a much higher number of ghost control volumes, owing to the larger overlap required between the sub-domains, resulting in much higher communication overhead at each location where information update is required.

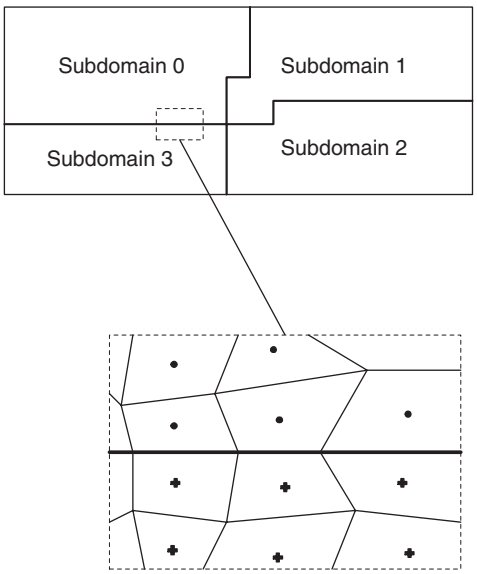

Figure 3. A domain decomposed into four subdomains.

The sequence of computations and communications in each iteration is as follows:

- Gradient reconstruction:
  - — calculate new gradient vectors,
  - — *update* the gradient vectors for the ghost control volumes,
- Implicit flux calculation:
  - — construct the transport equations for u, v, w and T,
  - — *update* the diagonals, a_x, a_y and a_z for the ghost control volumes,
  - — construct the continuity equations to be solved for p,
- Linear system solution:
  - — solve the equations for $\mathbf{F} = \{p, u, v, w\}$ and for T,
  - — *update* the primitive variables for ghost control volumes,
  - — calculate the mass flux at the integration points.

For a fixed size problem, domain decomposition is performed in a preprocessing stage. Load balancing over a homogeneous bank of processors is achieved by making all subdomains approximately the same size. In this work, the unstructured grid is partitioned in equally sized subdomains using MeTiS [30]. Each local control volume is renumbered locally. To enhance the single processor efficiency, the local numbering of control volumes is based on the topological vicinity of the control volumes. The local numbering of vertices follows an ascending pattern based on the numbering of the local control volumes. A separate finite-volume database is generated for each subdomain, to avoid serial I/O and its communication overhead. After performing computations, each process writes its own portion of the results to a separate output file. The serial post-processor reads one output file per process. This significantly reduces the communication overhead of a serial I/O and its master–slave parallel model.

Three sets of vectors are defined in the code: vector of primitive variables, vector of gradients and vector of diagonal coefficients. The importance of storing the data items in the form of these three vectors is to enhance the communication efficiency. Three ghost update levels are recognized in each iteration (see the sequence of the operations above): gradient vector update, matrix diagonals update and primitive variables update. To assist PETSc with generating a communication pattern with large packs of data, all vectors that update at the same ghost-data-update level are interlaced in a single vector. In this way, a larger number of ghost data items belonging to the same process is grouped together. An enhancement of 35% in the performance of the node-to-node communication is gained using this approach to data packing.

5. NUMERICAL PROCESS

Accurate results are required at the end of each time-step. Due to the non-linear nature of the Navier–Stokes equations, discretization techniques used for numerical analysis require methods of linearization. In the approach used in ParTISUN, the implicit convection term in its integral form is (iteratively) converged to a small tolerance by deferring the mass flux and performing non-linear iterations within each time-step. Therefore, each time-step consists of 10–50 non-linear iterations, depending on the size of the problem. This is a conventional approach and the method belongs to a family of methods often called *non-linearly consistent*. The method operates on the basis that the converged solution for the previous time-step serves as the initial guess for the next time-step. Within each time-step, the solver returns a fully converged solution to the linearized problem until the linearized problem approximates the actual non-linear problem to a suitable degree of accuracy. The accuracy of the solution is determined by substituting the solution for the previous iteration in the next linear system and calculating the second norm of the residual vector (non-linear residual). For the discretization methods used in this work, a value of 10^{-6} is deemed satisfactory for the non-linear residual.

In each non-linear iteration, the system of linear equations is solved using a combination of a Krylov subspace method and a preconditioner. The default setting in PETSc for linear system solver is restarted GMRES, preconditioned with block Jacobi with one block per processor, each of which is solved with ILU(0) [31]. There are more than ten other Krylov subspace methods available in PETSc (refer to Reference [17]). However, our experience shows that, for an identical convergence/stop criteria, using BiCGSTAB method with the default preconditioner needs the minimum number of solver iterations and leads to a more stable non-linear convergence.

The convergence in the linear solver is decided by two quantities: the relative decrease of the residual norm, r_{tol} and the absolute size of the residual norm, a_{tol}. Moreover, the linear solver stops after $k_{\max}$ iterations [31]. Obtaining an exact (fully converged) solution for the equations at each non-linear iteration is costly and only necessary near the convergence of the time-step. Therefore, by defining the convergence/stop criteria of the linear system solver as a function of the non-linear residual, the link between the non-linear algorithm and the linear solver is optimized, which in turn enhances the computational performance by saving a significant computational effort at the beginning of each time-step. This approach is a dynamic version of that implemented by Demirdzic *et al.* [14] in a similar non-linear method and that implemented by Gropp *et al.* [7] in the well-known inexact Newton–Krylov method.

In the approach implemented in References [7, 14], a constant inexact convergence criteria is defined for the linear solver at all non-linear iterations, for example $r_{\text{tol}} = 10^{-2}$ or $r_{\text{tol}} = 10^{-3}$ in Reference [14] or $k_{\max} = 10$–50 in Reference [7], whereas in the current approach a more accurate solution is dynamically demanded from the linear solver as the non-linear residual decreases. A speedup of upto four times has been achieved in this work by varying r_{tol} from 5.0×10^{-1} to 10^{-2} as a linear function of the non-linear residual. In general, the values of the upper and lower limits are not global for all problems and may vary based on the complexity of the flow.

6. VALIDATION AND APPLICATIONS

All the tests reported herein have been carried out using the Hammerhead cluster in SHARCNET. Hammerhead is a cluster of 27 computer nodes of Compaq Alpha ES40. Each node contains four 833 MHz Alpha processors with a shared memory of 4 GB. The nodes are connected by a fat-tree with Quadric switches of a sustainable transfer rate of 350 MB/s (this is the net transfer rate after protocol, i.e. the rate of user data movement and overhead for the protocol is already taken into account).

6.1. Discretization accuracy and grid convergence

6.1.1. Lid-driven cavity. To study the accuracy and robustness of the gradient formula, the flow inside a lid-driven cavity has been studied using four grids with similar grid size and different grid quality. Figure 4(a) shows an outline for the geometry. Two factors decrease the grid quality, the size of **R** and non-orthogonality. As Figure 5 shows, Grid B, Grid C and Grid D have low grid qualities with variying grid density and angle that lead to non-orthogonal lines with considerable displacement of the midpoints with respect to the integration points. The two-dimensional flow was simulated by applying symmetry boundary conditions on the z edges of the domain. Except for the symmetry walls, no-slip boundary conditions were applied on the remaining walls. The steady state solution was obtained for $Re = 1000$ and $Re = 3200$ with a maximum absolute difference of $\xi_{\text{st}} = 10^{-4}$ for the steady state convergence and a maximum non-linear residual of $\xi = 10^{-6}$. The velocity profiles along the horizontal

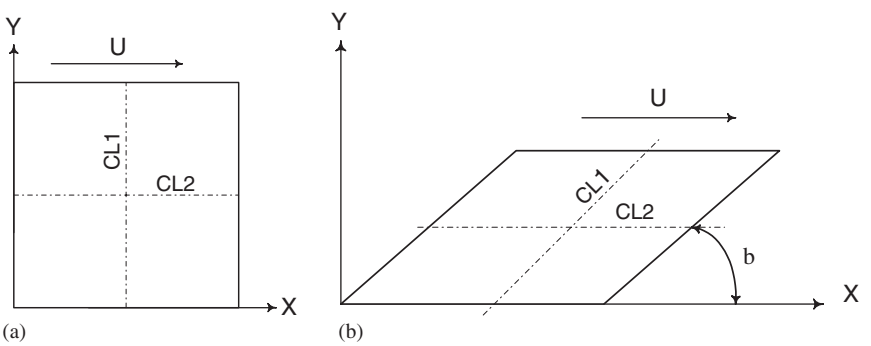

Figure 4. A schematic of the geometry for: (a) the lid-driven cavity; and (b) the inclined lid-driven cavity.

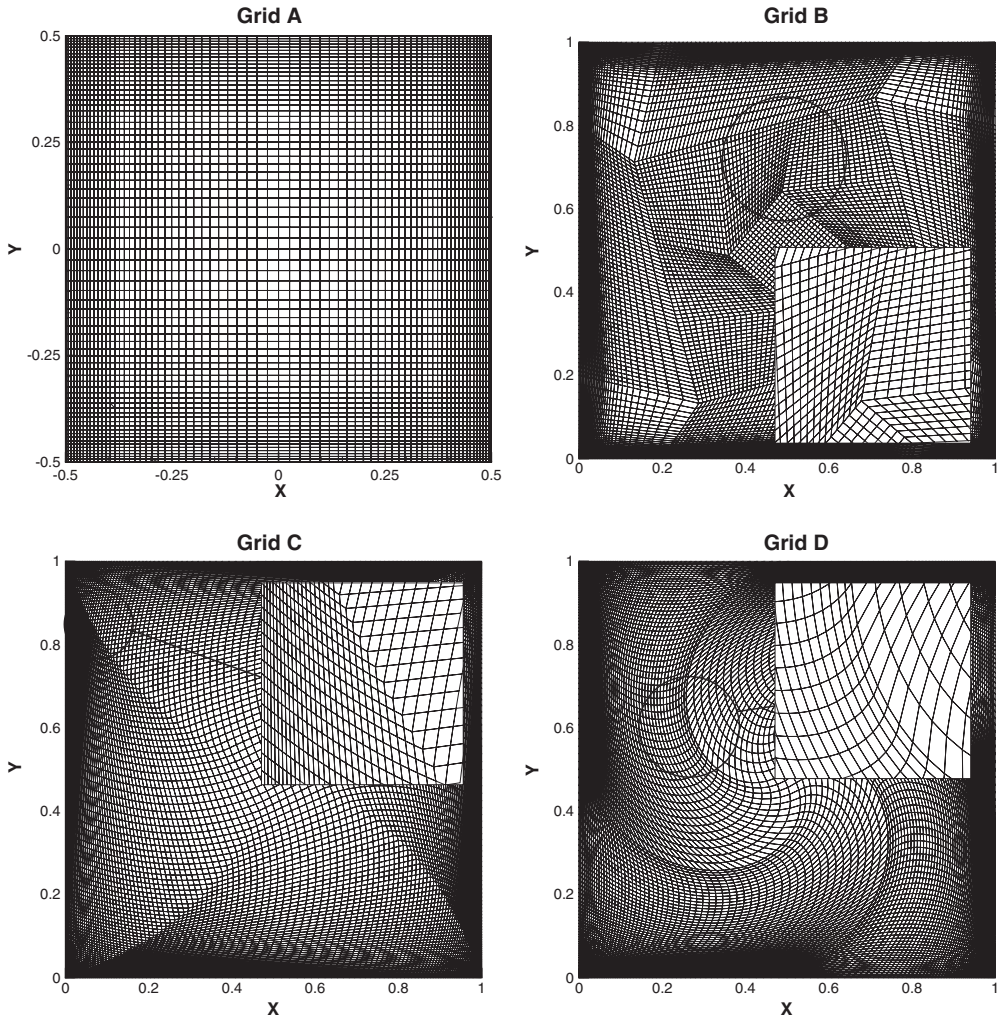

Figure 5. Lid-driven cavity: orthogonal and non-orthogonal grids.

and vertical centrelines are shown for two flows of $Re = 1000$ and $Re = 3200$ in Figure 6. Comparison of the results with that of Ghia *et al.* [32] shows that the new proposed gradient formula preserves the accuracy even in the case of a low quality grid. It is evident from Figure 7 that the effect of the grid quality on the convergence history of the code is not significant. This indicates the robustness of the method, even when the quality of the grid is low.

6.1.2. Inclined lid-driven cavity. To furthur validate the accuracy of the interpolation and gradient formulae, the flow inside an inclined lid-driven cavity has been studied. An outline of the geometry is shown in Figure 4(b). An inclination angle of $b = 30^\circ$ is chosen for this

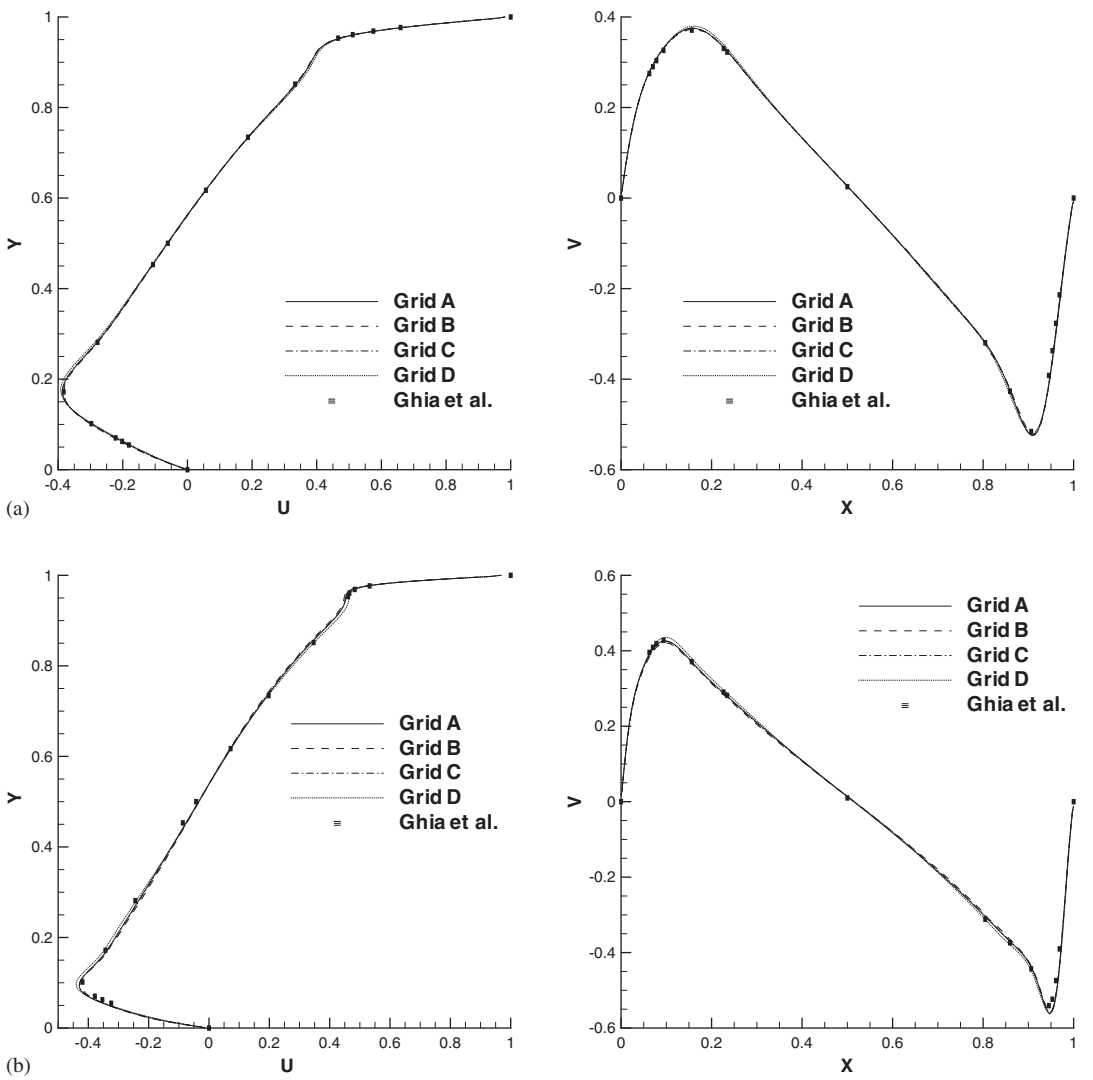

Figure 6. Lid-driven cavity: (a) velocity profiles along the centrelines for $Re = 1000$; and (b) velocity profiles along the centrelines for $Re = 3200$.

test case. Non-uniform grids of $40 \times 40 \times 20$, $80 \times 80 \times 40$ and $120 \times 120 \times 60$ with higher grid density near the walls were employed. The two-dimensional flow was simulated by applying the same boundary conditions as the above test case. The flow was computed for Reynold's numbers of $Re = 100$ and $Re = 1000$. In both cases, 32 processors were used for the two coarser grids and 64 processors for the finest grid. The steady state solution was obtained with a maximum absolute difference of $\xi_{st} = 10^{-4}$ for the steady state convergence and a maximum non-linear residual of $\xi = 10^{-6}$. The total number of iterations to obtain a steady state solution on the finest grid was 30 for $Re = 100$ and 67 for $Re = 1000$. To validate the

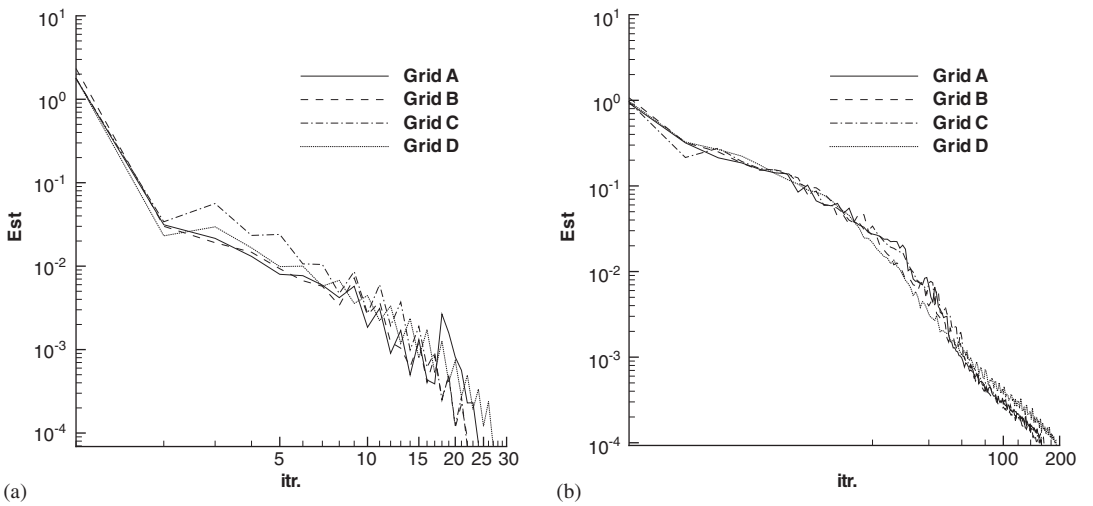

Figure 7. Convergence history: (a) $Re = 1000$; and (b) $Re = 3200$.

accuracy of the solution and the grid convergence, the velocity profiles along the horizontal and inclined centrelines are compared to those of Reference [33]. Figure 8 shows the velocity profiles along the centrelines CL1 and CL2 of Figure 4(b). For the case of $Re = 100$ a slight grid dependency is observed for grid $40 \times 40 \times 20$, whereas the grid dependency on the same grid for the case of $Re = 1000$ is not negligible. The profiles obtained from a grid of $120 \times 120 \times 60$ coincide almost exactly with the profiles obtained by Demirdžić *et al.* [33] on a grid of 320×320, thereby confirming the accuracy of the interpolation and grid calculation formulae implemented in the discretization of the equations in this work.

6.2. *Parallel performance*

Three-dimensional, time-dependent flow around a circular cylinder mounted in a channel serves as the test case to describe and assess parallel performance of the code. Figure 9 gives a schematic of the computational domain and shows the positioning of the cylinder in the domain. No-slip/zero-penetration boundary conditions were applied to the body of the circular cylinder and on all walls of the channel. At the domain outlet, a zero-normal gradient condition was applied to all variables, with the exception of pressure, which was set to an average value of zero. At the inlet, a uniform axial velocity was specified (with zero transverse components) and pressure was extrapolated. A Reynolds number of $Re = \rho UD/\mu = 200$ was simulated for all of the cases described below, where U is the inlet velocity and D is the diameter of the cylinder. The domain was discretized using unstructured hexahedral grids with 711 160 (Grid A) or 173 460 (Grid B) control-volumes. This translates to systems of equations with 2 844 640 or 693 840 unknowns, respectively.

Before describing the code performance, plots are shown to illustrate the results of the transient flow field for the problem under consideration. Figure 10 shows the computed streamlines for the flow around the circular cylinder. The flow is seen to form the familiar von Karman street behind the cylinder, although the vortex street is slightly supressed due to the proximity

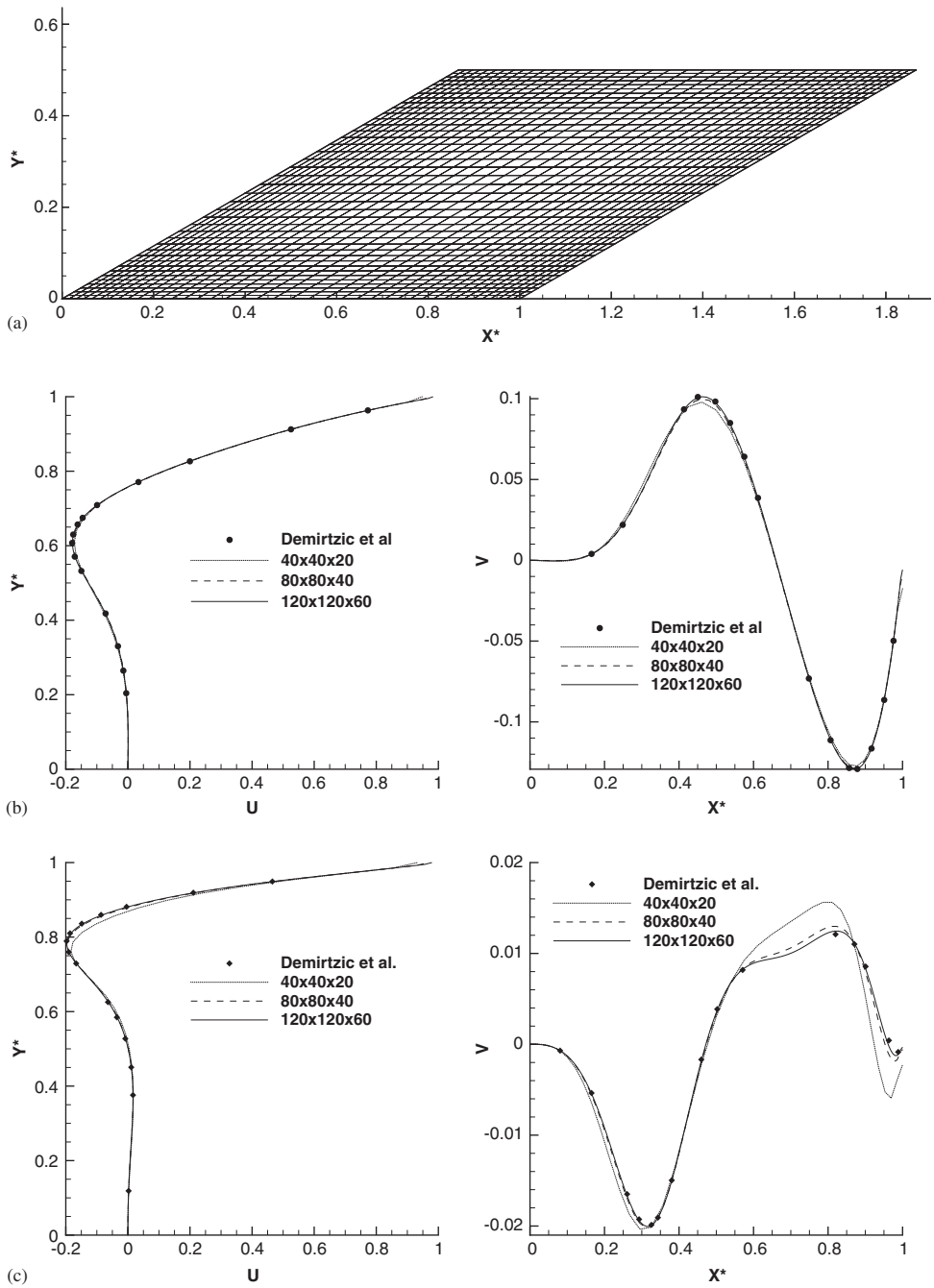

Figure 8. Inclined lid-driven cavity: (a) Grid for $40 \times 40 \times 10$; (b) velocity profiles along the centrelines for $Re = 1000$; and (c) velocity profiles along the centrelines for $Re = 100$.

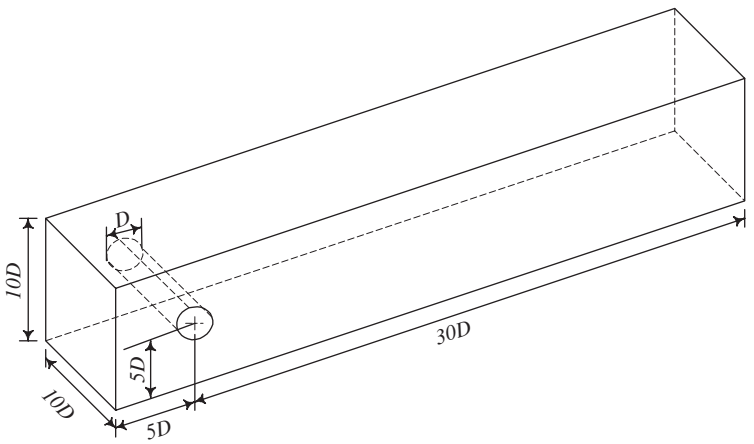

Figure 9. A geometry outline of the flow around a circular cylinder.

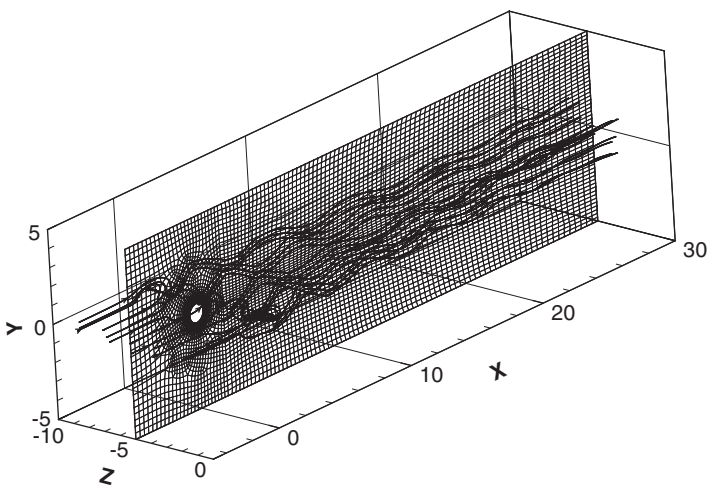

Figure 10. Computed streamlines for the flow around the circular cylinder.

of the cylinder walls. Figure 11 compares the computed lift and drag force versus time for solutions computed on three different domain decompositions. The results are seen to be virtually identical validating the numerical consistency of the code. Since the focus of the present paper is on the performance of the CFD code, a more detailed description of the physics of this test case is omitted.

Two factors are used [7] to describe the performance of the parallel code: (i) parallel scalablity, i.e. time per iteration in inverse proportion to the number of processors, and (ii) algorithm scalability, i.e. the required number of non-linear iterations with increased number of processors.

To study the performance of the parallel code in obtaining the solution, the domain was divided into N subdomains, where $1 \leqslant N \leqslant 96$ was considered. As an example, Figure 12

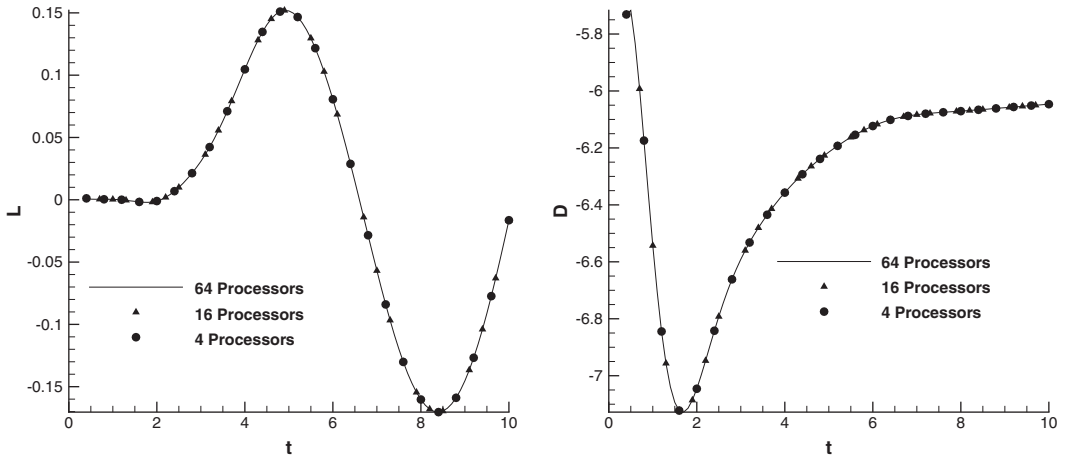

Figure 11. Evolution of lift and drag forces versus time for the first 100 time-steps, $Re = 200$.

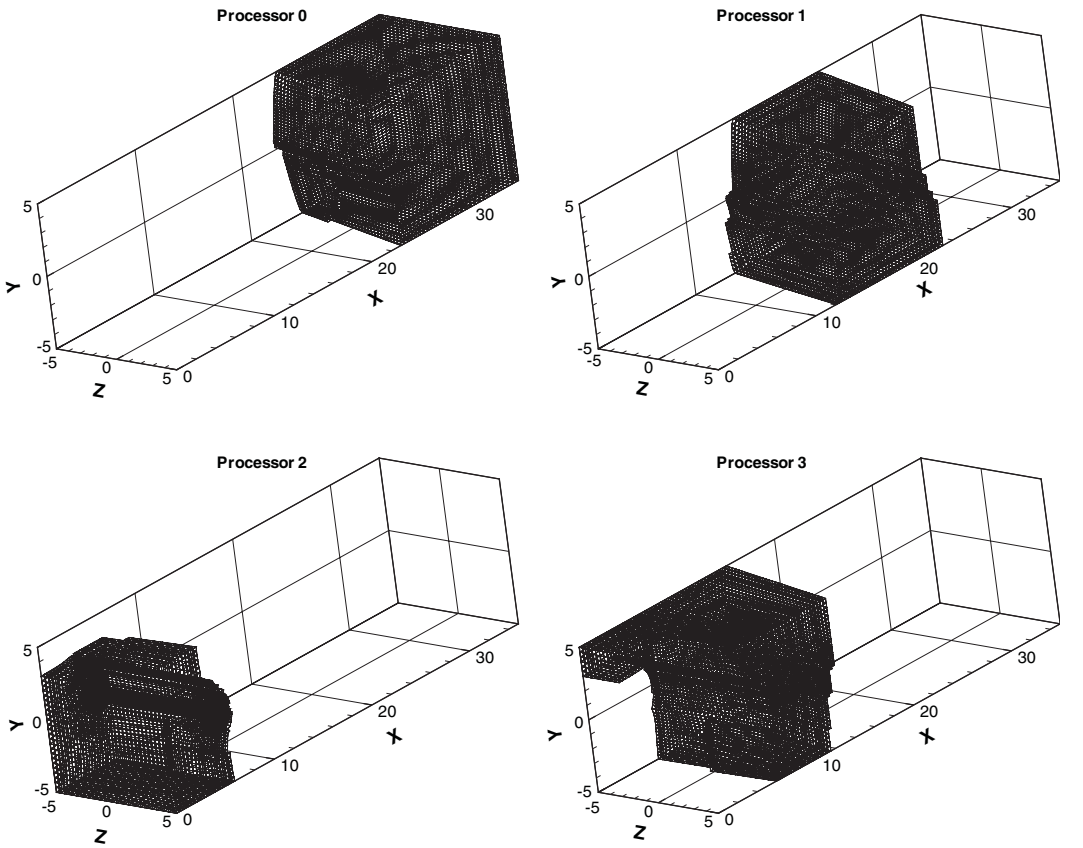

Figure 12. Grid partitioning into four subdomains.

Table I. Domain decomposition for Grid A.

No. of proc.	No. of local CV	Average no. of ghost CV	Comm. imbalance (%)	Load imbalance (%)
2	355 580	6320	0.00	0.00
4	177 790	5912	11.79	0.00
8	88 895	5272	18.33	0.00
16	44 447	4125	22.45	0.00
32	22 223	3178	16.23	0.00
64	11 112	2203	15.65	0.00
96	7408	1780	17.49	0.00

Table II. Domain decomposition for Grid B.

No. of proc.	No. of local CV	Average no. of ghost CV	Comm. imbalance (%)	Load imbalance (%)
2	86 730	1500	0.00	0.00
4	43 365	2296	38.47	0.00
8	21 683	2134	16.79	0.00
16	10 841	1763	11.86	0.00
32	5421	1251	12.24	0.00
64	2710	856	15.79	0.00
96	1355	701	16.33	0.00

shows the domain decomposition for $N=4$. Tables I and II show the statistics of all the domain decompositions considered for the parallel performance assessment of Grid A and Grid B, respectively. The results of the transient calculations were written to files after each time-step and thus the file I/O is included in all measurements of time. The efficiency and parallel speedup, which are indicators of the parallel scalability, are determined using parallel wall-clock time, total CPU time and the time per non-linear iteration. The parallel wall-clock time is defined as the maximum wall-clock time among all processors. The total CPU time is the aggregate CPU time of all processors spent on computations, communications and file I/O. The time per non-linear iteration is defined as the wall-clock time divided by the total number of non-linear iterations. The time-per-iteration is used to study the parallel scalibility without the effect of algorithm scalability. The difference between per-processor CPU time and the wall-clock time for each process is due to the idling time, the time each process is waiting for a communication taking place, the time a process is waiting for other processes at other types of synchronization points or the time a process waits for the file server to respond. The parallel speedup for n processor is defined as

$$S_p(n)=\frac{T_1}{T_n} \tag{22}$$

where T_1 is the time for a single processor run and T_n is the time for a run with n processors. The efficiency is determined by the following relation:

$$\varepsilon(n)=\frac{T_1}{nT_n} \tag{23}$$

Table III. Execution times (Grid A).

	CPU		WC			Iteration		
No. of proc.	Time (s)	ε_{cpu} (%)	Time (s)	ε_{wc} (%)	Idle (%)	No.	t/itr (s)	ε_{itr} (%)
1	75 652	100	75 675	100	0	171	442.54	100
2	118 469	64	68 398	55	13	253	270.35	82
4	122 839	62	30 751	62	0	220	139.78	79
8	134 808	56	17 046	55	1	239	71.32	78
16	115 205	66	7259	65	1	214	33.92	82
32	118 361	64	3729	63	1	232	16.07	86
64	118 984	64	1876	63	1	245	7.66	90
96	114 053	66	1200	66	1	242	4.96	93

Table IV. Execution times (Grid B).

	CPU		WC			Iteration		
No. of proc.	Time (s)	ε_{cpu} (%)	Time (s)	ε_{wc} (%)	Idle (%)	No.	t/itr (s)	ε_{itr} (%)
1	12 807	100	12 815	100	0	127	100.91	100
2	13 361	96	6955	92	4	126	55.20	91
4	12 903	99	3254	98	1	126	25.83	98
8	13 424	95	1686	95	0	127	13.28	95
16	13 840	93	870	92	1	133	6.54	96
32	13 945	92	443	90	2	137	3.24	97
64	14 063	91	220	91	0	136	1.62	97
96	13 188	97	143	93	4	136	1.05	100

Three factors adversely affect the efficiency based on wall-clock time ε_{wc}: the idling times, the communication pattern and the algorithm scalability. The percentage of the idling time with respect to the wall-clock time in Tables III and IV suggests that, except for the case of 2 processors, the idling time is negligible. One should expect such a conclusion on the basis of the zero load imbalance indicated in Tables I and II. The communication imbalance is defined as the standard deviation of the number of ghost control volumes of the subdomains divided by the number of processors. A high quality communication pattern can compensate such an imbalance and result in a minimum waiting time at the ghost-data-update points. From Tables III and IV it is also evident that the communication pattern generated by PETSc has compensated the communication imbalance. However, this is not the case for the two-processor execution, since in this case there is just one node-to-node communication connection and one process has to wait for the other to complete the communication. As can be observed from Figure 13, the parallel efficiency of ParTISUN remains almost constant over a wide range of N, regardless of the size of problem. The trend of increasing efficiency with the number of processors ε_{itr} on Grid A can be iterpreted in two ways:

- By increasing the number of processors, the average number of ghost control volumes decreases for a fixed size problem. Also, because the clusters used operate on a fat-tree network, there is almost no limitation in communication bandwidth as the number of processors increases. This suggests a significant decrease in the communication time, specifically at the communication bottlenecks, i.e. ghost-data-update points.

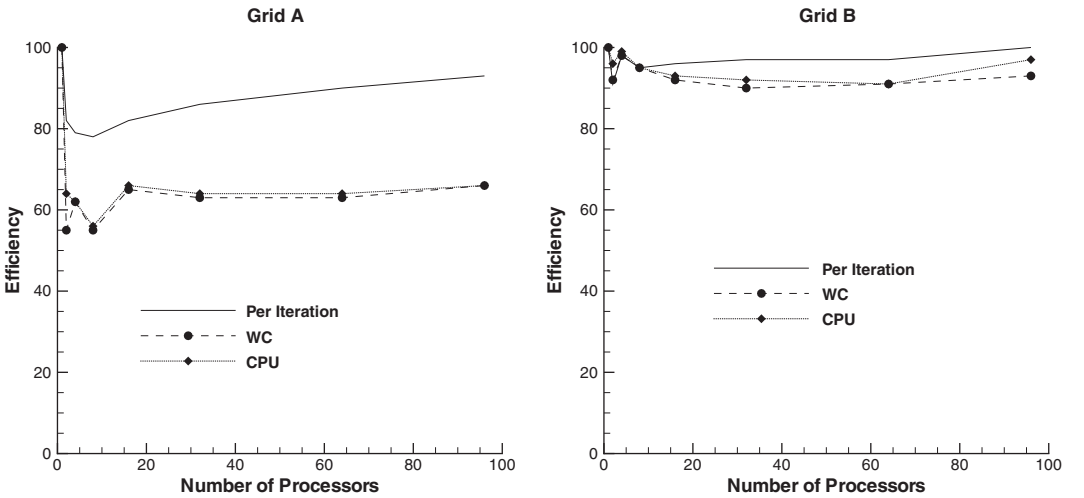

Figure 13. Parallel efficiency for Grid A and Grid B.

- The decreasing size of the subdomains with the number of processes results in a better fitting of data in the higher memory levels. This can be clearly concluded by observing the trend of increasing ε_{itr} in Figure 13. This fact has been considered as a drawback of a performance study in some other literature [34]. However, in the authors' opinion, if one's goal in single processor performance is to use the higher memory levels more often and to avoid transferring data from lower levels as much as possible, this fact must be considered an advantage of multiprocessing, i.e. if wall-clock time, which is directly related to the computation cost, is the key factor of comparison, everything that helps to reduce it must be taken into account.

The condition for algorithm scalability is that the convergence rate must be essentially independent of the number of processors. Therefore, the gap between ε_{wc} and ε_{itr} is the result of algorithm scalability. The number of non-linear iterations changes from about 170 for the serial run to about 220 for the parallel run on Grid A and from 127 to 137 on Grid B. However, it remains almost independent of the number of processors once it is running in parallel. The main reason for this jump in the number of iterations lies in the fact that the solver in PETSc implements different preconditioning methods for serial and parallel computations. However, as expected, Figure 14 shows that this gap does not affect the parallel scalability significantly; it just changes the slope of the speedup diagram. As shown in this figure, ParTISUN is scalable to within the limit of available computing resources for small size or large size problems, despite the fact that I/O is included in all time measurements.

Note that in a scalable program, the computation time per iteration is constant when the size of the problem and the number of processors are scaled proportionally. To show the scalability from this point of view, the computation time per iteration of Grid A is compared with its proportional counterpart on Grid B in Table V. Note that Grid A is almost four times larger than Grid B. From Table V, it is evident that the scalability increases with the number

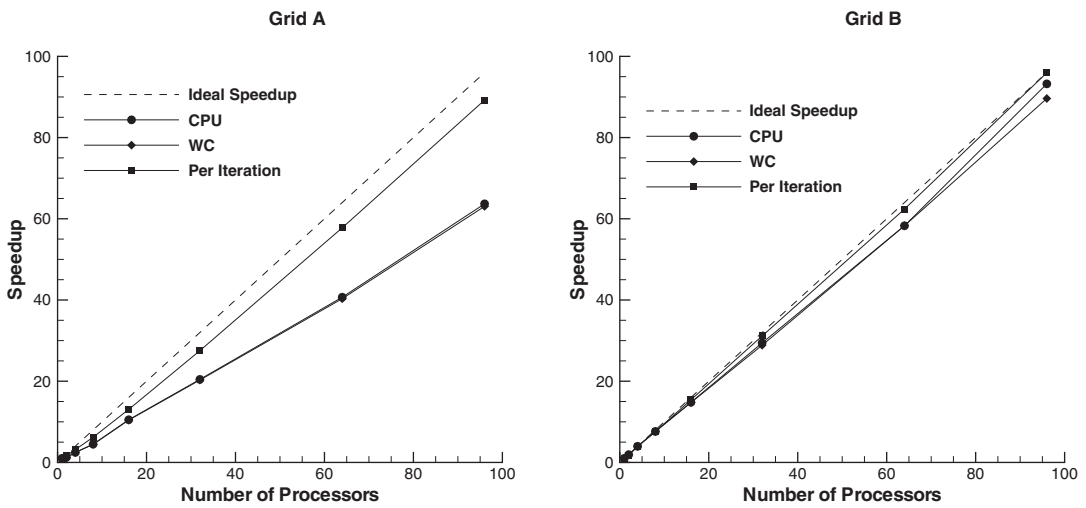

Figure 14. Parallel speedup using different timing schemes.

Table V. Scalability based on problem size.

Grid A		Grid B		
No. of proc.	Time per iteration	No. of proc.	Time per iteration	Scalability (%)
4	139.78	1	100.91	72
8	71.32	2	55.20	77
16	33.92	4	25.83	76
32	16.07	8	13.28	83
64	7.66	16	6.54	85

of processors. This indicates that ParTISUN functions more efficiently for larger problems with a larger number of processors.

7. CONCLUSION

The discretization details and performance of a three-dimensional, unstructured finite-volume code (ParTISUN) is presented in this work. The accuracy of the proposed discretization method was verified by running standard bench-mark test cases for non-orthogonal, non-uniform grids. The parallel performance of the code was described by considering a complex, three-dimensional, transient problem. The code was shown to scale linearly in both per-iteration performance and overall performance on two different density grids. The code was shown to be efficient for large size problems and large numbers of processors.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the financial support from the Natural Science and Engineering Research Council (NSERC) and from SHARCNET. The authors would also like to thank Baolai Ge and John Morton from SHARCNET for their valuable assistance.

REFERENCES

1. SHARCNET Homepage. http://www.sharcnet.ca, 2005.
2. Baliga BR, Patankar SV. A control-volume finite element method for two-dimensional fluid flow and heat transfer. *Numerical Heat Transfer* 1983; **6**:245–261.
3. Schneider GE, Raw MJ. Control-volume finite-element method for heat transfer and fluid flow using co-located variables-1. Computational procedures. *Numerical Heat Transfer* 1987; **11**:363–390.
4. Tai CH, Zhao Y. Parallel unsteady incompressible viscous flow computations using an unstructured multigrid method. *Journal of Computational Physics* 2003; **192**:277–311.
5. Anderson WK, Bonhaus DL. An implicit upwind algorithm for computing turbulent flows on unstructured grids. *Computers and Fluids* 1994; **23**:1–21.
6. Anderson WK, Rausch RD, Bonhaus DL. Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids. *Journal of Computational Physics* 1996; **128**:391–408.
7. Gropp WD, Kaushik DK, Keyes DE, Smith BF. High-performance parallel implicit CFD. *Parallel Computing* 2001; **27**:337–362.
8. Gerhold T, Friedrich O, Evans J, Galle M. Calculation of complex three-dimensional configuration employing the DLR-TAU-Code. *35th Aerospace Sciences Meeting and Exhibit*, 1997. AIAA-97-0167.
9. Aumann P, Barnewitz H, Schwarten H, Becker K, Heinrich R, Roll B, Galle M, Kroll N, Gerhold Th, Schwamborn D, Franke M. MEGAFLOW: parallel complete aircraft CFD. *Parallel Computing* 2001; **27**: 415–440.
10. Dolean V, Lanteri S. Parallel multigrid methods for the calculation of unsteady flows on unstructured grids: algorithmic aspect and parallel performances on clusters of PCs. *Parallel Computing* 2004; **30**:503–525.
11. Koubogiannis DG, Poussoulidis LC, Rovas DV, Giannakoglou KC. Solution of flow problems using unstructured grids on distributed memory platforms. *Computer Methods in Applied Mechanics and Engineering* 1998; **160**:89–100.
12. Grismer MJ, Strang WZ, Tomaro RF, Witzeman FC. Cobalt: a parallel, implicit, unstructured Euler/Navier–Stokes solver. *Advances in Engineering Software* 1998; **29**(3–6):365–373.
13. Lien F-S. A pressure-based unstructured grid method for all-speed flows. *International Journal for Numerical Methods in Fluids* 2000; **33**:355–374.
14. Demirdžić I, Muzaferija S. Numerical method for coupled fluid flow, heat transfer and stress analysis using unstructured moving meshes with cells of arbitrary topology. *Computer Methods in Applied Mechanics and Engineering* 1995; **125**:235–255.
15. Mathur SR, Murthy JY. A pressure-based method for unstructured meshes. *Numerical Heat Transfer*, *Part B* 1997; **31**:195–215.
16. Basara B. Employment of the second-moment turbulence closure on arbitrary unstructured grids. *International Journal for Numerical Methods in Fluids* 2004; **44**:377–407.
17. Balay S, Buschelman K, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Smith BF, Zhang H. PETSc Homepage, 2001.
18. van der Vorst HA. BiCGSTAB: a fast and smoothly converging variant of bicg for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1992; **13**:631–644.
19. Saad Y, Schultz MH. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1986; **10**:36–52.
20. Kholsa PK, Rubin SG. A diagonally dominant second-order accurate implicit scheme. *Computers and Fluids* 1974; **2**:207–209.
21. Blazek J. *Computational Fluid Dynamics*: *Principles and Applications*. Elsevier: Amsterdam, 2001.
22. Barth TJ. Aspects of unstructured grids and finite-volume solvers for the Euler and Navier–Stokes equations, special course on unstructured grid for advection dominated flows. *Technical Report 787*, AGARD, 1992.
23. Ferziger JH, Perić M. *Computational Methods for Fluid Dynamics* (2nd edn). Springer: Berlin, 1997.
24. Harten A. High resolution schemes for hyperbolic conservation laws. *Journal of Computational Physics* 1983; **49**:357–393.
25. Darwish MS, Moukalled F. TVD schemes for unstructured grids. *International Journal of Heat and Mass Transfer* 2003; **46**:599–611.
26. Roe PL. Some contributions to the modeling of discontinuous flows. *Proceedings of the AMS/SIAM Seminar*, San Diego, 1983.

27. Van Leer B. Towards the ultimate conservative difference scheme V. A second-order sequel to Godunov's method. *Journal of Computational Physics* 1979; **32**:101–136.
28. Rhie CM, Chow WL. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal* 1983; **21**(11):1525–1532.
29. Gropp W, Lusk E, Skjellum A. *Using MPI*: *Portable Parallel Programming with the Message Passing Interface* (2nd edn). MIT Press: Cambridge, MA, 1999.
30. Karypis G, Kumar V. A fast and high quality scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 1999; **20**:259–392.
31. Balay S, Buschelman K, Eijkhout V, Gropp WD, Kaushik D, Knepley MG, McInnes LC, Smith BF, Zhang H. PETSc users manual. *Technical Report ANL-95/11—Revision 2.1.5*, Argonne National Laboratory, 2004.
32. Ghia U, Ghia KN, Shin T. High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method. *Journal of Computational Physics* 1982; **48**:387–411.
33. Demirdžić I, Lilek Ž, Perić M. Fluid flow and heat transfer test problems for non-orthogonal grids: bench-mark solutions. *International Journal for Numerical Methods in Fluids* 1992; **15**:329–354.
34. Baker L, Smith BJ. *Parallel Programming*. McGraw-Hill: New York, 1996.

