Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 192 (2003) 277–311
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Parallel unsteady incompressible viscous flow computations using an unstructured multigrid method

Chin Hoe Tai, Yong Zhao *

School of Mechanical and Production Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Republic of Singapore

Received 15 September 2001; received in revised form 14 July 2003; accepted 16 July 2003

Abstract

The development and validation of a parallel unstructured non-nested multigrid method for simulation of unsteady incompressible viscous flow is presented. The Navier–Stokes solver is based on the artificial compressibility approach and a higher-order characteristics-based finite-volume scheme on an unstructured multigrid. Unsteady flow is calculated with an implicit dual time-stepping scheme. The parallelization of the solver is achieved by multigrid domain decomposition approach (MG-DD), using the single program multiple data (SPMD) programming paradigm and message-passing interface (MPI) for communication of data. The parallel codes using single grid and multigrid are used to simulate steady and unsteady incompressible viscous flows over a circular cylinder for validation and performance evaluation purposes. The speedups and parallel efficiencies obtained by both the parallel single grid and multigrid solvers are reasonably good for both test cases, using up to 32 processors on the SGI Origin 2000. A maximum speedup of 12 could be achieved on 16 processors for high-Reynolds number unsteady viscous flow. The parallel results obtained were compared with those using serial single grid and multigrid codes and it remains the same as those obtained by serial solvers and agrees well with numerical solutions obtained by other researchers as well as experimental measurements.

Keywords: Parallel computing; Multigrid; Unstructured grid; High-order scheme; Unsteady incompressible flow

1. Introduction

Over the last several decades, tremendous progress has been made in both computational fluid dynamics (CFD) algorithms and the computer hardware technologies. The computing speed and memory capacity of computers has increased exponentially during this period. CFD is also one of the most important areas of application for high-performance computing, setting the pace for developments in scientific computing. The demands of CFD to tackle more and more complex problems found in engineering applications at system level, particularly in the aerospace and automotive industries, coupled with the emergence of new, more powerful generations of parallel computers, have led naturally to work on parallel computing. It is widely

* Corresponding author.
E-mail address: myzhao@ntu.edu.sg (Y. Zhao).

0021-9991/$ - see front matter
doi:10.1016/j.jcp.2003.07.005

recognized that parallel computing is the long-term solution for intensive computer simulation in CFD research and its engineering applications. Parallel computers offer the promise to provide the massive computational resource that CFD requires. Algorithms and strategies that successfully map *structured grid* codes onto parallel machines have been developed over the previous decade and have become quite established. Extension of the capabilities of these structured grid codes to include unstructured grid codes requires new algorithms and strategies to be developed.

Multigrid techniques have been demonstrated as an efficient means for obtaining steady-state numerical solutions to both the compressible Euler and Navier–Stokes equations on unstructured meshes in two and three dimensions [1–4], which find their applications mainly in aerospace engineering. Recent years have also seen an increasing interest in applying the above methods to steady incompressible flow simulation [5–8] through the use of the artificial compressibility method proposed by Chorin [9], a field which has been traditionally dominated by the pressure-based methods such the SIMPLE-type methods and the projection method. However, it is still very time consuming to simulate unsteady flows, especially unsteady incompressible viscous flows due to their elliptic nature, which means that global iterations are required to achieve the divergence free condition. To simulate unsteady incompressible flows with the artificial compressibility method, a dual time-stepping scheme is necessary. While physical time integration can usually be obtained with a three-point second-order implicit scheme, time-stepping in pseudo time can be explicit [10–12] or implicit [8,13,14]. Mavriplis [15,16] have developed a parallel unstructured agglomeration multigrid strategy using a graph algorithm to construct the coarse multigrid levels from the given fine grid and partitioned each grid level independently for the computation of steady-state aerodynamic flows. On the other hand, Llorente et al. [17] have presented a highly parallel multigrid method for three-dimensional convection-dominated problems and proposed two different strategies to get the parallel implementation of multigrid method, which are domain decomposition followed by multigrid (DD-MG) generation in each partition and multigrid generation followed by grid partitioning (MG-DD) at all levels.

This work attempts to parallelize the serial unstructured multigrid code by multigrid domain decomposition approach (MG-DD), using the Single Program Multiple Data (SPMD) programming paradigm and Message-Passing Interface (MPI) [18], in order to efficiently simulate unsteady incompressible viscous flows. The ultimate aim of this study is to develop a highly parallel multigrid solver so as to avoid very large single processor memory requirements in sequential processing and to reduce the simulation time for simulating unsteady flows with complex geometries.

2. Mathematical formulation

The two-dimensional Navier–Stokes equations for incompressible unsteady flows, modified by the artificial compressibility method, can be written in vector form with dimensionless parameters:

$$\mathbf{C}\frac{\partial \mathbf{W}}{\partial \tau}+\mathbf{K}\frac{\partial \mathbf{W}}{\partial t}+\nabla\cdot\vec{\mathbf{F}}_{\mathrm{c}}=\nabla\cdot\vec{\mathbf{F}}_{\mathrm{v}}, \tag{1}$$

where

$$\mathbf{W}=\begin{bmatrix}p\\u\\v\end{bmatrix},\quad \vec{\mathbf{F}}_{\mathrm{c}}=\begin{bmatrix}\vec{\mathbf{U}}\\u\vec{\mathbf{U}}+(p/\rho)\vec{i}\\v\vec{\mathbf{U}}+(p/\rho)\vec{j}\end{bmatrix},\quad \vec{\mathbf{F}}_{\mathrm{v}}=\begin{bmatrix}0\\\frac{1}{Re}\left(\frac{\partial u}{\partial x}\vec{i}+\frac{\partial u}{\partial y}\vec{j}\right)\\\frac{1}{Re}\left(\frac{\partial v}{\partial x}\vec{i}+\frac{\partial v}{\partial y}\vec{j}\right)\end{bmatrix},$$

$$\mathbf{K}=\begin{bmatrix}0&0&0\\0&1&0\\0&0&1\end{bmatrix},\quad \mathbf{C}=\begin{bmatrix}1/\beta&0&0\\0&1&0\\0&0&1\end{bmatrix},$$

where $\mathbf{W}$ is the flow field variable vector, $\mathbf{U} = u \cdot \vec{i} + v \cdot \vec{j}$, and $\vec{\mathbf{F}}_c$ and $\vec{\mathbf{F}}_v$ are the convective and viscous flux vectors, respectively, β being a constant called artificial compressibility, $\mathbf{K}$ is the unit matrix, except that the first element is zero and $\mathbf{C}$ is a preconditioning matrix.

Eq. (1) can be recast in an integral form as follows:

$$\mathbf{C}\frac{\partial}{\partial\tau}\int\int_S \mathbf{W}\,\mathrm{d}S + \mathbf{K}\frac{\partial}{\partial t}\int\int_S \mathbf{W}\,\mathrm{d}S + \int\int_S \nabla\cdot\left(\vec{\mathbf{F}}_c - \vec{\mathbf{F}}_v\right)\mathrm{d}S = 0. \tag{2}$$

The above equation can then be re-written as follows:

$$\mathbf{C}\frac{\partial}{\partial\tau}\int\int_S \mathbf{W}\,\mathrm{d}S + \mathbf{K}\frac{\partial}{\partial t}\int\int_S \mathbf{W}\,\mathrm{d}S + \oint_l \left(\vec{\mathbf{F}}_c - \vec{\mathbf{F}}_v\right)\cdot \mathrm{d}\vec{l} = 0.$$

Once the artificial steady state is reached, those derivative terms with respect to τ become zero and the above equation reduces to the following equation:

$$\mathbf{K}\frac{\partial}{\partial t}\int\int_S \mathbf{W}\,\mathrm{d}S + \oint_l \left(\vec{\mathbf{F}}_c - \vec{\mathbf{F}}_v\right)\cdot \mathrm{d}\vec{l} = 0. \tag{3}$$

Eq. (3) shows that the preconditioning matrix does not affect the solution and the original unsteady incompressible Navier–Stokes equations are fully recovered.

The non-dimensional variables used in above equations are defined as follows:

$$(x,y) = \left(\frac{x^*}{L^*},\frac{y^*}{L^*}\right);\quad (u,v) = \left(\frac{u^*}{U_\infty^*},\frac{v^*}{U_\infty^*}\right);\quad t = \frac{t^*}{L^*/U_\infty^*};\quad p = \frac{p^* - p_o}{\rho(U_\infty^*)^2}.$$

3. Numerical methods

A new unstructured-grid high-order characteristics-based upwind finite-volume algorithm [19] is used to discretize the governing equations. The outline of the method is described in the following sections.

3.1. Finite volume formulation

The 2D equations in Eq. (2) are discretized on an unstructured grid and a cell-vertex scheme is adopted here, i.e., all computed variables in vector $\mathbf{W}$ are stored at vertices of the triangular cells. For every vertex, as shown in Fig. 1, a control volume is constructed by joining the centres of the cells to the centres of the edges using the median dual of the triangular grid.

Spatial discretization is performed by using the integral form of the conservation equations over the control volume surrounding node or vertex P, as shown in the following equation:

$$\mathbf{C}\frac{\partial}{\partial\tau}\int\int_{S_{cv}} \mathbf{W}_p\,\mathrm{d}S + \mathbf{K}\frac{\partial}{\partial t}\int\int_{S_{cv}} \mathbf{W}_p\,\mathrm{d}S + \int\int_{S_{cv}} \nabla\cdot\vec{\mathbf{F}}_c\,\mathrm{d}S = \int\int_{S_{cv}} \nabla\cdot\vec{\mathbf{F}}_v\,\mathrm{d}S. \tag{4}$$

In order to introduce the upwind scheme using an edge-based procedure, the convective term is transformed into a summation as in the following equation:

$$\int\int_{S_{cv}} \nabla\cdot\vec{\mathbf{F}}_c\,\mathrm{d}S = \oint_{L_{cv}} \vec{\mathbf{F}}_c\cdot\vec{\mathbf{n}}\,\mathrm{d}l = \sum_{k=1}^{\mathrm{nbseg}}\left[\left(\vec{\mathbf{F}}_c\right)_{ij}^{k}\cdot\vec{\mathbf{n}}\Delta l_k\right], \tag{5}$$

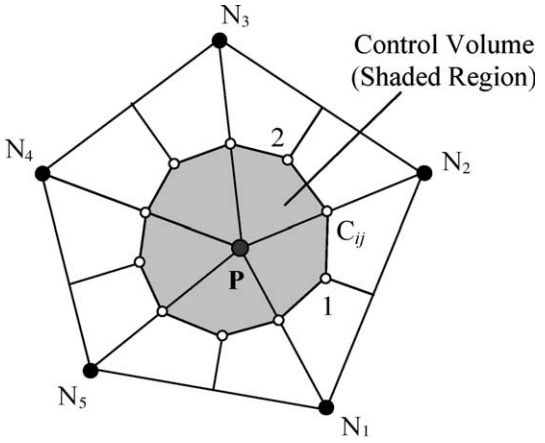


Fig. 1. Construction of control volume for node P.

where nbseg is the number of the edges connected to node P, $(\vec{\mathbf{F}}_c)_{ij}^k$ is the convection flux through the part of control volume boundary (similar to 1-C_{ij}-2 in Fig. 1). The length of the boundary is Δl_k and its effective outward normal unit vector is $\vec{n}$. Therefore, all the fluxes are calculated for the edges and then collected at the two ends of each edge for updating of flow variables using the time-marching scheme.

The viscous term is calculated using a cell-based method, which is shown in the following equation:

$$\int\int_{S_{cv}} \nabla \cdot \vec{\mathbf{F}}_v \, dS = \oint_{L_{cv}} \vec{\mathbf{F}}_v \cdot d\vec{l} = \sum_{i=1}^{ncell} (\vec{\mathbf{F}}_v \cdot \Delta\vec{l}_c)_i = \frac{1}{2} \sum_{i=1}^{ncell} (\vec{\mathbf{F}}_v \cdot \Delta\vec{l}_p)_i, \tag{6}$$

where $\Delta\vec{l}_{ci}$ is the part of control volume boundary in cell C_i and $\Delta\vec{l}_{pi}$ is the edge vector of the cell edge opposite to node P of the triangle under consideration. Here $(\vec{\mathbf{F}}_v)_i$ is calculated at the centre of the triangular cell, and can be obtained by using the Green's Theorem and the variables at the three vertices of the triangle. Here ncell is the number of cells surrounding node P. The viscous flux in Eq. (6) is actually calculated in a cell-by-cell manner and then collected at the nodes of the cells for the calculation of the residuals at all the nodes.

3.2. The upwind characteristics-based method

Similar to the approach in [20] for compressible flow and that in [19] for incompressible flow on structured grids, a high-order characteristics-based scheme for incompressible flow and heat transfer on arbitrary unstructured grids has been developed in [21,33]. This method is used for the discretization of the convective part of the Navier–Stokes equations. A 2D version of this is briefly shown below. The Euler equation modified by the Chorin's method [9] are rewritten in partial differential form in Cartesian coordinate system for the derivation of the method of characteristics:

$$\frac{\partial p}{\partial \tau} + \beta \frac{\partial u_i}{\partial x_i} = 0, \tag{7}$$

$$\frac{\partial u_i}{\partial \tau} + u_j \frac{\partial u_i}{\partial x_j} + u_i \frac{\partial u_j}{\partial x_j} + \frac{\partial p}{\partial x_i} = 0, \tag{8}$$

where subscripts i and j equal 1 or 2, representing two spatial co-ordinates. Suppose that ξ is a new co-ordinate outward normal to the boundary of a control volume that surrounds a particular vertex. In order to extend the method of characteristics to the unstructured grid solver, it is assumed that flow in the ξ direction is approximately one-dimensional and the above equations can then be transformed into the following equations:

$$\frac{\partial p}{\partial \tau} + \beta \frac{\partial u_j}{\partial \xi} \xi_{x_j} = 0, \tag{9}$$

$$\frac{\partial u_i}{\partial \tau} + u_j \frac{\partial u_i}{\partial \xi} \xi_{x_j} + u_i \frac{\partial u_j}{\partial \xi} \xi_{x_j} + \frac{\partial p}{\partial \xi} \xi_{x_i} = 0, \tag{10}$$

where $\xi_{x_i} = \partial \xi / \partial x_i$ and $\xi_{x_j} = \partial \xi / \partial x_j$.

In the $\tau - \xi$ space as shown in Fig. 2, flow variable $\mathbf{W}$ at pseudo time level $m+1$ can be calculated along a characteristics, k, using a Taylor series expansion and the initial value at pseudo time level $m(\mathbf{W}^k)$

$$\mathbf{W} = \mathbf{W}^k + \mathbf{W}_\xi \xi_\tau \Delta\tau + \mathbf{W}_\tau \Delta\tau \tag{11}$$

and

$$\mathbf{W}_\tau = \frac{\mathbf{W} - \mathbf{W}^k}{\Delta\tau} - \mathbf{W}_\xi \xi_\tau. \tag{12}$$

A wave speed, λ^k, is introduced

$$\xi_\tau = \lambda^k \sqrt{\xi_{x_i} \xi_{x_i}},$$

and the unit normal vector components are

$$n_{x_j} = \frac{\xi_{x_j}}{\sqrt{\xi_{x_i} \xi_{x_i}}}.$$

Substituting components of W_τ into Eqs. (11) and (12), we have

$$\frac{1}{\sqrt{\xi_{x_i} \xi_{x_i}}} \frac{(p - p^k)}{\Delta\tau} - p_\xi \lambda^k + \beta(u_\xi n_x + v_\xi n_y) = 0, \tag{13}$$

$$\frac{1}{\sqrt{\xi_{x_i} \xi_{x_i}}} \frac{(u - u^k)}{\Delta\tau} - u_\xi(\lambda^0 - \lambda^k) + u(u_\xi n_x + v_\xi n_y) + p_\xi n_x = 0, \tag{14}$$

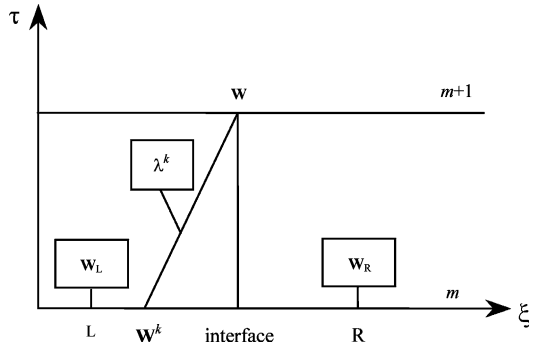


Fig. 2. τ–ξ Co-ordinate.

$$\frac{1}{\sqrt{\xi_{x_i}\xi_{x_i}}}\frac{(v-v^k)}{\Delta\tau} - v_\xi(\lambda^0-\lambda^k) + v(u_\xi n_x + v_\xi n_y) + p_\xi n_y = 0, \tag{15}$$

where λ^0 is the contravariant velocity

$$\lambda^0 = un_x + vn_y.$$

In order to derive the compatibility equations, the spatial derivatives, such as u_ξ, v_ξ, p_ξ have to be eliminated from the above equations. Following the approach of Eberle [20] for compressible flow equations, each of the above four equations is multiplied by an arbitrary variable and all the resulting equations are summed to form a new equation as follows:

$$\frac{1}{\Delta\tau\sqrt{\xi_{x_i}\xi_{x_i}}}A - p_\xi B + u_\xi C + v_\xi D = 0, \tag{16}$$

where

$$\begin{aligned} A &= a(p-p^k) + b(u-u^k) + c(v-v^k),\\ B &= -a\lambda^k + bn_x + cn_y,\\ C &= an_x\beta + b(\lambda^0 - \lambda^k + un_x) + cvn_x,\\ D &= an_y\beta + bun_y + c(\lambda^0 - \lambda^k + vn_y), \end{aligned} \tag{17}$$

and a, b, c and d are the arbitrary variables used to multiply the equations. We define the coefficients of the partial space derivatives to be zero, i.e., A, B, C and D are zero,

$$A = 0, \tag{18}$$

$$B = 0, \tag{19}$$

$$C = 0, \tag{20}$$

$$D = 0. \tag{21}$$

Eqs. (18)–(21) constitute a linear system $\Phi X = 0$ with $X = \{a, b, c, d\}$. Variables a, b, c and d are generally non-zero, thus the system of equations has non-trivial solution. This means that $\det(\Phi) = 0$, and the following eigenvalues can be derived as:

$$\lambda^0 = un_x + vn_y,$$

$$\lambda^1 = \lambda^0 + \sqrt{(\lambda^0)^2 + \beta} = \lambda^0 + C,$$

$$\lambda^2 = \lambda^0 - \sqrt{(\lambda^0)^2 + \beta} = \lambda^0 - C.$$

For each eigenvalue or characteristics speed, characteristic equations can be derived from Eqs. (18)–(21). For example, for $\lambda^k = \lambda^0$, we have

$$a = \frac{bn_x + cn_y}{\lambda^0}.$$

Substituting this into Eq. (17), we obtain,

$$\frac{bn_x + cn_y}{\lambda^0}(p - p^0) + b(u - u^0) + c(v - v^0) = 0,$$

i.e.,

$$b[n_x(p - p^0) + \lambda^0(u - u^0)] + c[n_y(p - p^0) + \lambda^0(v - v^0)] = 0.$$

For any b, c and d, the above equation is always satisfied. Therefore, all the terms in square brackets are zero. As a result, we have

$$(u - u^0)n_y - (v - v^0)n_x = 0. \tag{22}$$

For $\lambda = \lambda^1$,

$$p - p^1 = -\lambda^1[(u - u^1)n_x + (v - v^1)n_y]. \tag{23}$$

For $\lambda = \lambda^2$,

$$p - p^2 = -\lambda^2[(u - u^2)n_x + (v - v^2)n_y]. \tag{24}$$

The flow parameters at $(m + 1)$ pseudo time level are then calculated using the characteristics-based method along the three characteristics (see, e.g. [21,33,34]):

$$u = fn_x + u^0 n_y^2 - v^0 n_x n_y, \tag{25}$$

$$v = fn_y + v^0 n_x^2 - u^0 n_y n_x, \tag{26}$$

$$p = p^1 - \lambda^1[(u - u^1)n_x + (v - v^1)n_y], \tag{27}$$

where

$$C = \sqrt{(\lambda^0)^2 + \beta};$$

$$f = \frac{1}{2C}[(p^1 - p^2) + n_x(\lambda^1 u^1 - \lambda^2 u^2) + n_y(\lambda^1 v^1 - \lambda^2 v^2)].$$

Flow quantities at $m + 1$ pseudo time level obtained from the above equations on the characteristics are then used to calculate convection fluxes at the control volume interface. Those on different characteristics at m time level are approximately evaluated by an upwind scheme using the signs of the characteristics as suggested in [19]:

$$\mathbf{W}^j = \tfrac{1}{2}\left[\left(1 + \operatorname{sign}\left(\lambda^j\right)\right)\mathbf{W}_\mathrm{L} + \left(1 - \operatorname{sign}\left(\lambda^j\right)\right)\mathbf{W}_\mathrm{R}\right], \tag{28}$$

where $\mathbf{W}_\mathrm{L}$ and $\mathbf{W}_\mathrm{R}$ are obtained by the high-order upwind-biased interpolation as introduced in the following section.

3.3. Upwind-biased interpolation

Here, an edge-based method for calculating the total inviscid flux is adopted by calculating and storing the flux integrals based on the associated edges. The left and right state vectors $\mathbf{W}_\mathrm{L}$ and $\mathbf{W}_\mathrm{R}$ on both sides of

a control volume surface associated with an edge, ij, can be evaluated using an upwind-biased interpolation scheme along the edge which is similar to the approach presented in [35] for the finite element method:

$$\mathbf{W}_{\mathrm{L}} = \mathbf{W}_i + \tfrac{1}{4}[(1-\kappa)\Delta_i^- + (1+\kappa)\Delta_i^+], \tag{29a}$$

$$\mathbf{W}_{\mathrm{R}} = \mathbf{W}_j - \tfrac{1}{4}[(1-\kappa)\Delta_j^+ + (1+\kappa)\Delta_j^-], \tag{29b}$$

where

$$\Delta_i^+ = \Delta_j^- = \mathbf{W}_j - \mathbf{W}_i,$$

$$\Delta_i^- = \mathbf{W}_i - \mathbf{W}_{i-1} = 2\overrightarrow{ij}\cdot\nabla\mathbf{W}_i - (\mathbf{W}_j - \mathbf{W}_i) = 2\overrightarrow{ij}\cdot\nabla\mathbf{W}_i - \Delta_i^+,$$

$$\Delta_j^+ = \mathbf{W}_{j+1} - \mathbf{W}_j = 2\overrightarrow{ij}\cdot\nabla\mathbf{W}_j - (\mathbf{W}_j - \mathbf{W}_i) = 2\overrightarrow{ij}\cdot\nabla\mathbf{W}_j - \Delta_j^-.$$

Therefore,

$$\mathbf{W}_{\mathrm{L}} = \mathbf{W}_i + \tfrac{1}{2}[(1-\kappa)\overrightarrow{ij}\cdot\nabla\mathbf{W}_i + \kappa\Delta_i^+], \tag{30a}$$

$$\mathbf{W}_{\mathrm{R}} = \mathbf{W}_j - \tfrac{1}{2}[(1-\kappa)\overrightarrow{ij}\cdot\nabla\mathbf{W}_j + \kappa\Delta_j^-], \tag{30b}$$

where κ is set to 1/3, which corresponds to a nominally third-order accuracy, subscripts i and j represent indices for the two end nodes defining the edge. The two values will then be used in the characteristics-based method, which has already been introduced in the previous section. The gradients of $\mathbf{W}$ at i and j are calculated by volume-averaging the gradients of the cells that surround i and j.

3.4. Dual time-stepping algorithm

Finally, for a given node P, the spatially discretized equations form a system of coupled ordinary differential equations, which can be reformulated as

$$\mathbf{C}\frac{\partial}{\partial\tau}(\Delta S_{\mathrm{cv}}\mathbf{W}_p) + K\frac{\partial}{\partial t}(\Delta S_{\mathrm{cv}}\mathbf{W}_p) = -\left\{\sum_{k=1}^{\mathrm{nbseg}}[(\vec{\mathbf{F}}_{\mathrm{c}})_{ij}^k\cdot\vec{\boldsymbol{n}}\Delta l_k)] - \frac{1}{2}\sum_{i=1}^{\mathrm{ncell}}(\vec{\mathbf{F}}_{\mathrm{v}}\cdot\vec{\boldsymbol{n}}\Delta l_P)_i\right\} = -R(\mathbf{W}_P), \tag{31}$$

where $R(\mathbf{W}_P)$ represents the residual which includes the contributions of the convective and diffusive fluxes and ΔS_{cv} is the control volume of node P.

An implicit scheme is adopted to approximate Eq. (31) in the physical time, and the semi-discrete equations are

$$\mathbf{C}\frac{\partial}{\partial\tau}(\Delta S_{\mathrm{cv}}\mathbf{W}_p) + \mathbf{K}\frac{\partial}{\partial t}(\Delta S_{\mathrm{cv}}^{n+1}\mathbf{W}_p^{n+1}) = -R(\mathbf{W}_P^{n+1}). \tag{32}$$

The superscript $(n+1)$ denotes the physical time level $(n+1)\Delta t$ and all the variables are evaluated at this time level. The time-dependent term in Eq. (32) can be discretized by a three-point second-order difference scheme, thus it becomes

$$\mathbf{C}\frac{\partial}{\partial\tau}(\Delta S_{\mathrm{cv}}\mathbf{W}_p) = -R(\mathbf{W}_P^{n+1}) - \mathbf{K}\left(\frac{1.5\Delta S_{\mathrm{cv}}^{n+1}\mathbf{W}_p^{n+1} - 2.0\Delta S_{\mathrm{cv}}^{n}\mathbf{W}_p^{n} + 0.5\Delta S_{\mathrm{cv}}^{n-1}\mathbf{W}_p^{n-1}}{\Delta t}\right) = \tilde{R}(\mathbf{W}_p^{n+1}), \tag{33}$$

where $\tilde{R}(\mathbf{W}_p^{n+1})$ is the new modified residual which contains both the time derivative and flux vectors. The advantage of the above implicit scheme is that the physical time-step size is not restricted by numerical stability, but only by numerical accuracy. This is especially useful in unsteady flow simulation. The derivative with respect to a fictitious pseudo time τ is discretized as

$$\mathbf{C}\Delta S_{\mathrm{cv}}^{n+1}\frac{\mathbf{W}_p^{n+1,m+1}-\mathbf{W}_p^{n+1,m}}{\mathrm{d}\tau}=\tilde{R}(\mathbf{W}_p^{n+1,m}), \tag{34}$$

whose solution is sought by marching to a pseudo steady state in τ. Here m and $(m+1)$ denote the initial and final pseudo time levels. Once the artificial steady state is reached, the derivative of $\mathbf{W}_p$ with respect to τ becomes zero, and the solution will satisfy $\tilde{R}(\mathbf{W}_p^{n+1})=0$. Hence, the original unsteady Navier–Stokes equations are fully recovered. Therefore, instead of solving each time-step in the physical time domain (t), the problem is transformed into a sequence of steady-state computations in the artificial time domain (τ), and Eq. (34) is integrated in pseudo time by the five-stage Runge–Kutta time stepping scheme as follows:

$$\begin{aligned}
\mathbf{W}_p^{(0)}&=\mathbf{W}_p^{m},\\
\mathbf{W}_p^{(1)}&=\mathbf{W}_p^{(0)}-\alpha_1\frac{\Delta\tau}{\Delta S_{\mathrm{cv}}}\tilde{R}(\mathbf{W}_p^{(0)}),\\
\mathbf{W}_p^{(2)}&=\mathbf{W}_p^{(0)}-\alpha_2\frac{\Delta\tau}{\Delta S_{\mathrm{cv}}}\tilde{R}(\mathbf{W}_p^{(1)}),\\
\mathbf{W}_p^{(3)}&=\mathbf{W}_p^{(0)}-\alpha_3\frac{\Delta\tau}{\Delta S_{\mathrm{cv}}}\tilde{R}(\mathbf{W}_p^{(2)}),\\
\mathbf{W}_p^{(4)}&=\mathbf{W}_p^{(0)}-\alpha_4\frac{\Delta\tau}{\Delta S_{\mathrm{cv}}}\tilde{R}(\mathbf{W}_p^{(3)}),\\
\mathbf{W}_p^{(5)}&=\mathbf{W}_p^{(0)}-\alpha_5\frac{\Delta\tau}{\Delta S_{\mathrm{cv}}}\tilde{R}(\mathbf{W}_p^{(4)}),\\
\mathbf{W}_p^{(m+1)}&=\mathbf{W}_p^{(5)},
\end{aligned} \tag{35}$$

where the stage coefficients are as follows:

$$\alpha_1=1/4,\quad \alpha_2=1/6,\quad \alpha_3=3/8,\quad \alpha_4=1/2,\quad \alpha_5=1.$$

4. Convergence acceleration techniques

In this work, time-dependent calculations require the convergence of the Navier–Stokes equations to the steady state in pseudo-time for each real time-step. The methods used to accelerate convergence to steady state in pseudo time are local time-stepping, implicit residual smoothing and multigrid. Since these convergence acceleration techniques are only used to accelerate convergence in pseudo-time, they will not directly affect the accuracy in real time. In the dual time-stepping algorithm, global time-stepping is used to advance the solution in real or physical time whereas local time-stepping is used to advance the solution in pseudo time.

4.1. Local time-stepping in pseudo time

In this work, the large variation in grid size in the unstructured mesh will restrict the time-step used and the smallest control volume dictates the maximum time-step. In order to overcome the above problems,

each control volume can be advanced in pseudo time by its own maximum local time-step, which greatly enhances the convergence rate. The local time-step size can be estimated via the Courant–Friedrichs–Lewy (CFL) stability condition:

$$\Delta\tau = \mathrm{CFL} \cdot \frac{\Delta l}{\left|\vec{U}\right| + c}, \tag{36}$$

where Δl is the length scale associated with a node under consideration. Normally, it is taken as the smallest height of all the cells sharing the node. For global time-stepping, Δl is taken as the smallest value of all the nodes in the domain. And c is the speed of sound. Another condition that limits the time-step in τ is $\Delta\tau \leqslant \frac{2}{3}\Delta t$.

4.2. Implicit residual smoothing

In order to speed up the convergence rate, an implicit residual smoothing scheme developed for unstructured grids [21] is employed. The idea behind this is to replace the residual at one point of the flow field with a smoothed or weighted average of the residuals at the neighboring points. The averaged residuals are calculated implicitly in order to increase the maximum CFL number, thus increasing the convergence rate. Normally this procedure allows the CFL number to be increased by a factor of 2 or 3. The smoothing equation for a vertex k can be expressed as follows:

$$\bar{R}_k = R_k + \varepsilon\nabla^2\bar{R}_k, \tag{37}$$

where R is the original residual, $\bar{R}$ is smoothed residual and ε is the smoothing coefficient, which can be defined as,

$$\varepsilon = \max\left\{\frac{1}{4}\left[\left(\frac{\mathrm{CFL}}{\mathrm{CFL}^*}\right)^2 - 1\right], 0\right\}, \tag{38}$$

where CFL* is the maximum CFL number of the basic scheme. The solution to the above equations can be obtained on an unstructured grid by using the Jacobi iterative method as follows:

$$\bar{R}_k^{(m)} = R_k^{(0)} + \varepsilon \sum_{i=1}^{\mathrm{numnod}(k)} [\bar{R}_i^{(m)} - \bar{R}_k^{(m)}],$$

i.e.,

$$\bar{R}_k^{(m)} = \frac{R_k^{(0)} + \varepsilon \sum_{i=1}^{\mathrm{numnod}(k)} \bar{R}_i^{(m-1,m)}}{1 + \varepsilon \cdot \mathrm{numnod}(k)}, \tag{39}$$

where $\mathrm{numnod}(k)$ is the number of neighboring nodes of vertex k.

4.3. Multigrid method

In this work, the idea of the multigrid method is to carry out early iterations on a fine grid and then progressively transfer these flow field variables and residuals to a series of coarser grids. On the coarser grids, the low frequency errors become high frequency ones and they can be easily eliminated by a

time-stepping scheme. The flow is then solved on the coarser grids and the corrections are then interpolated back to the fine grid. The process is repeated over a sufficient number of times until satisfactory convergence on the fine grid is achieved. For the ease of implementation, the non-nested mesh method using independently generated non-nested (or overset) coarse meshes is adopted [4,7]. Two different cycle strategies [2,3] have been investigated in the present work, which are V-cycle and W-cycle as depicted in Fig. 3. For a three-grid system, a multigrid cycle begins by solving the flow on a fine grid, then the flow field variables and residuals are transferred to a coarser grid. After the flow is solved on this coarser grid, it is then transferred to the coarsest grid where the flow is solved again. The correction is then interpolated up to the fine grid with an additional iteration being performed on the fine grid. For one complete V-cycle, the flow is interpolated back up to the two finer grids once the coarsest grid is reached, with one new evaluation of the flow field variables on the fine grid. For one complete W-cycle, the flow is interpolated up one grid after performing the iteration on the coarsest grid. Then the flow field variables are evaluated on this grid and the variables are transferred down to the coarsest grid again. The flow is solved on the coarsest grid and the correction is then interpolated up to the fine grid.

The initial solution and residuals on the coarse grid $(h+1)$ is transferred from the fine grid (h) as shown below

$$\mathbf{W}_{h+1}^{(0)} = T_h^{h+1}\mathbf{W}_h, \tag{40}$$

$$\tilde{R}_{h+1}^{(0)} = Q_h^{h+1}\tilde{R}(\mathbf{W}_h), \tag{41}$$

where $\mathbf{W}_{h+1}^{(0)}$ and $\tilde{R}_{h+1}^{(0)}$ are the initial flow field values and residuals, respectively, which are transferred from the fine grid. $\mathbf{W}_h$ and $\tilde{R}(\mathbf{W}_h)$ are the fine grid flow field variables and residual. T_h^{h+1} and Q_h^{h+1} are the solution and residual transfer operator, based on weighted averaging. These transfer operators are *restriction* operators that may not necessarily be the same.

In order to drive the coarser grid solution using the fine grid residual, a forcing function is calculated at the first stage of the Runge–Kutta scheme and subsequently added to the residual on the coarse grid. The forcing function on the coarse grid is

$$P_{h+1} = \tilde{R}_{h+1}^{(0)} - \tilde{R}(\mathbf{W}_{h+1}^{(0)}). \tag{42}$$

It should be noted that on coarse grids, time-dependent terms in the residual containing $\mathbf{W}$ at n and $n-1$ time levels are not included for ease of calculation. Instead, they are only included in the fine-grid residual and directly transferred to the coarse grids.

After calculating the variables on the coarsest grid, the corrections are evaluated and interpolated back to the fine grid. The correction is the difference between the newly computed value on the coarse grid, $\mathbf{W}_{h+1}^{+}$,

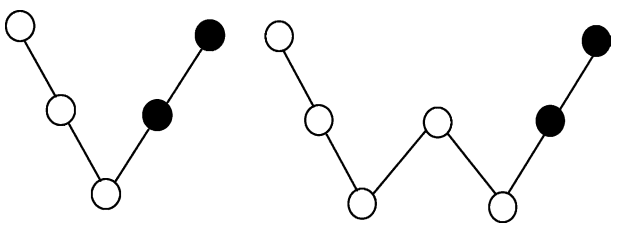


Fig. 3. V-cycle and W cycle for three grid levels. ○ denotes flow field computation on a particular level and ● denotes interpolation of corrections of flow field variables.

and the initial value that was transferred from the finer grid, $\mathbf{W}_{h+1}^{(0)}$, and this correction is transferred to the fine grid and added to the solution on that grid, i.e.,

$$\mathbf{W}_h^+ = \mathbf{W}_h + I_{h+1}^h(\mathbf{W}_{h+1}^+ - \mathbf{W}_{h+1}^{(0)}), \tag{43}$$

where I_{h+1}^h is an *interpolation* operator from the coarse grid to the fine grid and $\mathbf{W}_h^+$ is the updated solution. In order to improve the efficiency for the simulation of viscous flows, the viscous terms are only evaluated on the fine grid and not evaluated on the coarser grids. Since the coarser grids are only used to cancel the dominating low frequency errors, this treatment does not affect the accuracy of the solution. The upwind-biased interpolation scheme is set to first-order at the coarser levels where the left–right states of Eq. (30a) and (30b) are taken as the values of the two nodes of the edge to calculate the flux associated with the edge.

4.3.1. Inter-connectivity relationship between meshes

Before the flow field variables and residuals are transferred from the fine grid to the coarse grid or the corrections are interpolated from the coarse grid back to the fine grid, it is necessary to determine in which coarse cell each fine node is located and vice versa. In order to reduce pre-processing time, the flow field domain is decomposed into a number of square zones, in which searching for a particular node is only done within the related square zones, instead of searching the whole flow field.

The algorithm for inter-connectivity relationship between meshes is based on the concept of dot product of two vectors: the unit normal vector oriented *inward* from an edge, $\vec{n}$, and the vector $\vec{p}$ which points from the centre of the edge to a node under consideration. The dot product of these two vectors, $(p_n)^{\text{nedg}}$, for the three edges is shown as follows:

$$(p_n)^{\text{nedg}} = \left\{\vec{p}\right\}.\left\{\vec{n}\right\} = \{p_x p_y\}.\begin{Bmatrix} n_x \\ n_y \end{Bmatrix} = \left(p_x n_x + p_y n_y\right)^{\text{nedg}} \geqslant 0, \quad \text{nedg} = 1 \text{ to } 3, \tag{44}$$

where the superscript nedg is the edge number of the cell. And $(p_n)^{\text{nedg}}$ must be positive for all the three edges if the node under consideration falls within the cell as depicted in Fig. 4.

The search for nodes within the boundary edges will continue after the search for nodes within the cells are performed if the user specifies that the boundary is curved. Two criteria are used to determine if a node is within the boundary edge. The first criterion is depicted in Fig. 5(a) where $\vec{n}$ is the *outward* normal vector of the boundary edge and $\vec{p}$ is a vector pointing from the centre of the boundary edge to the node considered. If the product is greater than or equal to zero, then the node is considered to be within the edge, as shown in the following:

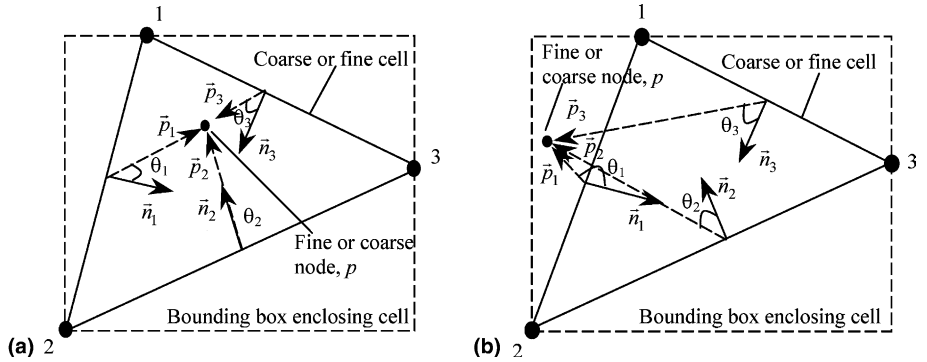


Fig. 4. (a) Node falls within a cell using dot product; (b) Node does not fall within a cell.

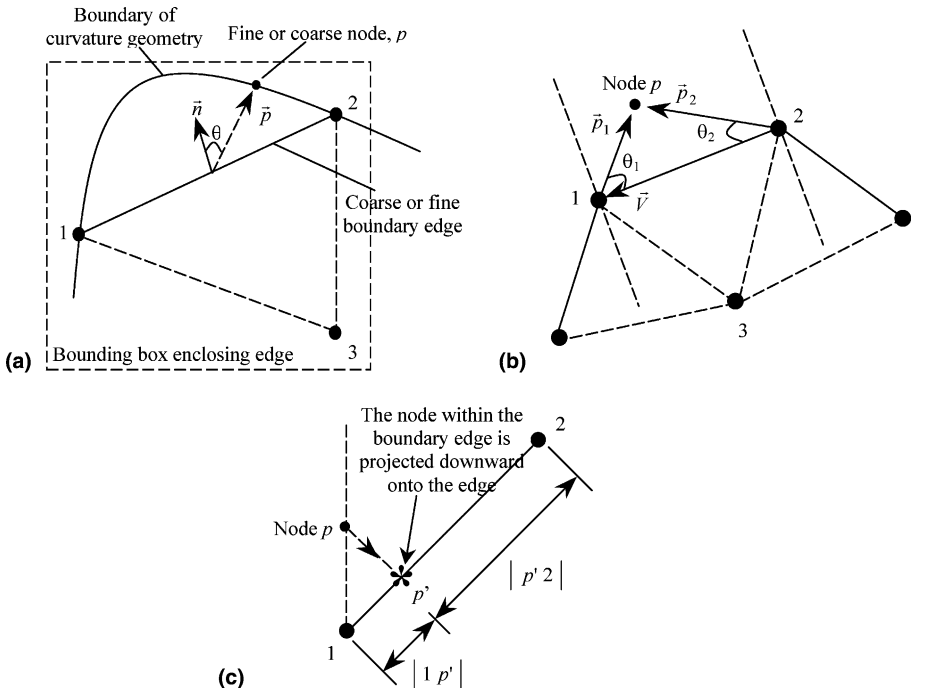


Fig. 5. (a) Fine node within the coarse boundary edge; (b) Different sign computed between $\vec{p}_1.\vec{v}$ and $\vec{p}_2.\vec{v}$; (c) Projected lengths for the transfer operator algorithms.

$$(p_n) = \left\{\vec{p}\right\}.\left\{\vec{n}\right\} = \left\{p_x p_y\right\}.\begin{Bmatrix} n_x \\ n_y \end{Bmatrix} = \left(p_x n_x + p_y n_y\right) \geqslant 0. \tag{45}$$

In order to ensure that vector $\vec{p}$ is not pointing to a node that is very far away from the boundary edge, the boundary edge will be enclosed within a bounding box. The second criterion is depicted in Fig. 5(b) where the dot product of both $\vec{p}_1.\vec{v}$ and $\vec{p}_2.\vec{v}$ must be different in sign. If the node being tested fulfills these two criteria, then the node is projected downward onto the boundary edge and the projected lengths between this projected node and the two edge nodes are computed for the transfer operator alogorithms, as illustrated in Fig. 5(c).

4.3.2. Mesh-to-mesh transfer operators

There are two classes of mesh-to-mesh transfer operators being implemented in the present study, which are *restriction* operators and *prolongation* operators. These operators follow the approach presented [12] for data transfer within the domain and incorporate the new technique developed here for a curved boundary.

4.3.2.1. Flow-field-variable transfer operators. The *restriction* transfer operator, T_h^{h+1}, that transfers the flow field values from the fine grid to the coarse grid is given as follows:

$$\mathbf{W}_1 = \frac{A_a \mathbf{W}_a + A_b \mathbf{W}_b + A_c \mathbf{W}_c}{A_a + A_b + A_c}. \tag{46}$$

Lower case letters denote fine grid nodes and Arabic numbers denote coarse grid nodes for all the figures in this section. The flow field values at the coarse node 1, which is contained in the fine cell formed by nodes a,

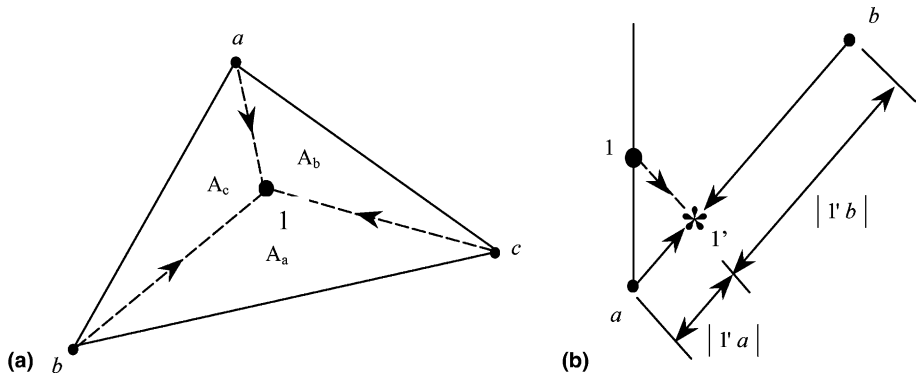


Fig. 6. (a) Transfer of flow field values from the fine mesh to the coarse mesh; (b) transfer of variables from fine nodes to the coarse node at the boundary.

b and c, is a weighted average of the values at those nodes as shown in Fig. 6(a). A_a, A_b, and A_c are the areas of the corresponding triangles opposite to the nodes.

On the curved boundary, the flow field values are transferred from the fine nodes of the edge formed by vertices a and b onto the coarse node $1'$. The *restriction* transfer operator, T_h^{h+1}, is based on the following formula:

$$\mathbf{W}_1 = \frac{|1'\ b|\mathbf{W}_a + |1'\ a|\mathbf{W}_b}{|a\ b|}. \tag{47}$$

The flow field values at the coarse node $1'$, which is projected downward onto the fine edge enclosed by nodes a and b, is a weighted average of the values at those nodes as shown in Fig. 6(b). $|1'\ a|$ and $|1'\ b|$ are the projected lengths between the fine nodes and the projected coarse node. $|a\ b|$ is the length of the fine edge.

4.3.2.2. Residual transfer operators. The transfer of residuals from the fine nodes to the coarse nodes is based on an area-weighted contribution calculation as depicted in Fig. 7(a). The final transferred residual at a coarse grid node will be the summation of the contributions from all the fine nodes located within all the coarse cells surrounding this particular coarse grid node. For the three nodes of a coarse grid cell, the residual contributions they receive from a single fine grid node inside the cell are calculated as

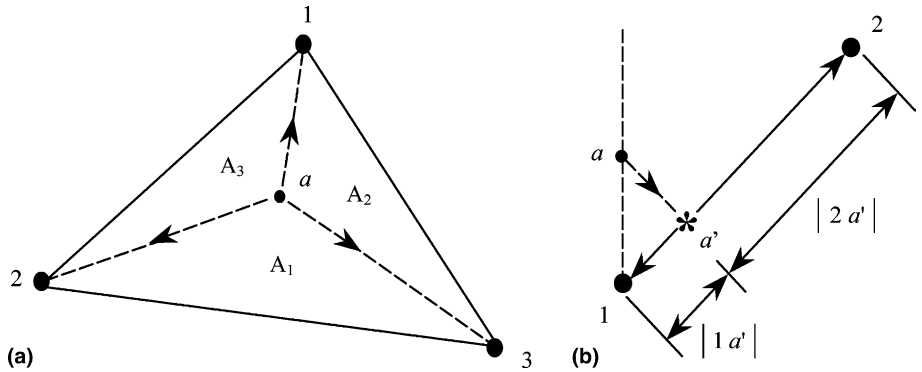


Fig. 7. (a) Transfer of residuals from the fine mesh to the coarse mesh; (b) transfer of residuals from the fine node to the coarse nodes at the boundary.

$$\begin{aligned}\tilde{R}_1 &= \tilde{R}_1 + \frac{A_1\tilde{R}_a}{A_1+A_2+A_3},\\ \tilde{R}_2 &= \tilde{R}_2 + \frac{A_2\tilde{R}_a}{A_1+A_2+A_3},\\ \tilde{R}_3 &= \tilde{R}_3 + \frac{A_3\tilde{R}_a}{A_1+A_2+A_3},\end{aligned} \tag{48}$$

where A_1, A_2 and A_3 are the areas of the corresponding triangles opposite to the nodes. It is easy to show that this transfer is conservative in the sense that the total fine mesh residuals are equal to the coarse mesh ones.

On a curved boundary the residual contributions from fine node a', which is projected onto a coarse edge enclosed by coarse nodes 1 and 2, are distributed to the coarse nodes 1 and 2 using a length-weighted contribution calculation. The transfer operator, Q_h^{h+1} is given as

$$\begin{aligned}\tilde{R}_1 &= \tilde{R}_1 + \frac{|2\ a'|\tilde{R}_{a'}}{|1\ 2|},\\ \tilde{R}_2 &= \tilde{R}_2 + \frac{|1\ a'|\tilde{R}_{a'}}{|1\ 2|}.\end{aligned} \tag{49}$$

This is shown in Fig. 7(b) for a fine node within a coarse boundary edge. It can be seen that the final residual at the coarse nodes 1 or 2 is actually the summation of all the contributions from those fine nodes projected onto the two edges that share the coarse node. This also ensures that the residual transfer is conservative on the boundary.

4.3.2.3. Correction transfer operators. Prolongation operators are used to transfer corrections of the flow field variables from the coarse mesh to the fine mesh, which is illustrated in Fig. 8. The corrections on a coarse mesh are calculated as follows:

$$\mathrm{d}\mathbf{W}_{h+1} = \mathbf{W}_{h+1}^{+} - \mathbf{W}_{h+1}^{(0)}. \tag{50}$$

The corrections are then transferred to the fine mesh by the *prolongation* operator, I_{h+1}^{h}:

$$v_h = I_{h+1}^{h}\,\mathrm{d}\mathbf{W}_{h+1}.$$

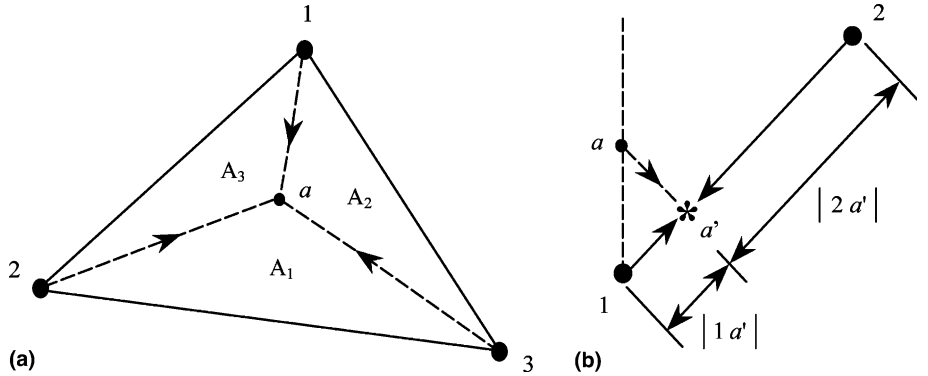


Fig. 8. (a) Transfer of corrections from the coarse mesh to the fine mesh; (b) transfer of corrections from the coarse nodes to the fine node at the boundary.

According to Fig. 8(a), the correction of the flow field variables transferred from coarse nodes 1, 2 and 3 to fine node a is a weighted average of the corrections at 1, 2 and 3 and the expression for the transferred correction is

$$(v_a)_h = \frac{A_1(\mathrm{d}\mathbf{W}_1)_{h+1} + A_2(\mathrm{d}\mathbf{W}_2)_{h+1} + A_3(\mathrm{d}\mathbf{W}_3)_{h+1}}{A_1 + A_2 + A_3}, \tag{51}$$

where A_1, A_2 and A_3 are the areas of the corresponding triangles opposite to the nodes 1, 2 and 3, respectively.

The corrections of the flow field variables are transferred from the coarse nodes of the edge formed by vertices 1 and 2 to the fine node a' as shown in Fig. 8(b). The transfer operator, I^h_{h+1}, is derived as follows:

$$(v_{a'})_h = \frac{|2\ a'|(\mathrm{d}\mathbf{W}_1)_{h+1} + |1\ a'|(\mathrm{d}\mathbf{W}_2)_{h+1}}{|1\ 2|}. \tag{52}$$

Thus, the correction at the fine node a', which is projected downward onto the coarse edge formed by nodes 1 and 2, is also a weighted average of the values at the two coarse nodes. $|1\ a'|$ and $|2\ a'|$ are the projected lengths between the coarse nodes and the projected fine node. $|1\ 2|$ is the length of the coarse edge.

5. Parallel implementation

The motivation for this section is to develop algorithms and strategies that would enable Navier–Stokes solvers based on *unstructured grids* to exploit the computational advantages offered by SGI Origin 2000 based on silicon graphics scalable shared-memory multiprocessing (S2MP) architecture. This work is based on the method of multigrid domain decomposition (MG-DD) method and the single program multiple data (SPMD) programming paradigm.

5.1. Parallelization strategy

The serial single grid and multigrid codes have been parallelized using a concept known as single program multiple data (SPMD) based on MG-DD. This technique involves generating the multi-level grids and decomposing the problem domain into a set of S partitions (sub-domains) that may be distributed over P processors in the parallel machine. Each processor runs the same program and operates only on its part of the problem with the *ghost* nodes flow variables and nodal gradients communicating with other processors through the exchange of data using the message-passing interface (MPI) [18]. The purpose of the message passing between the processors is to maintain consistency with the original sequential solver. An overlapping mesh-partitioning technique is employed in this work so as to build the control volume for finite-volume formulation. The flowchart presented in Fig. 9 depicts the parallelization strategy used in this work and $kt_{\max}$ and $t_{\max}$ denote the maximum time-steps and the maximum time for running, respectively.

5.2. Domain decomposition

Domain decomposition of a mesh into a set S of sub-domains that maybe allocated to a set of P processors involves finding a partition of the mesh so that the amount of computational time on each processor is almost equal. METIS [22,23], a software package for partitioning unstructured meshes, large irregular graphs and computing fill-reducing orderings of sparse matrices, is employed to produce the partitions for the finest level of grids used in this work.

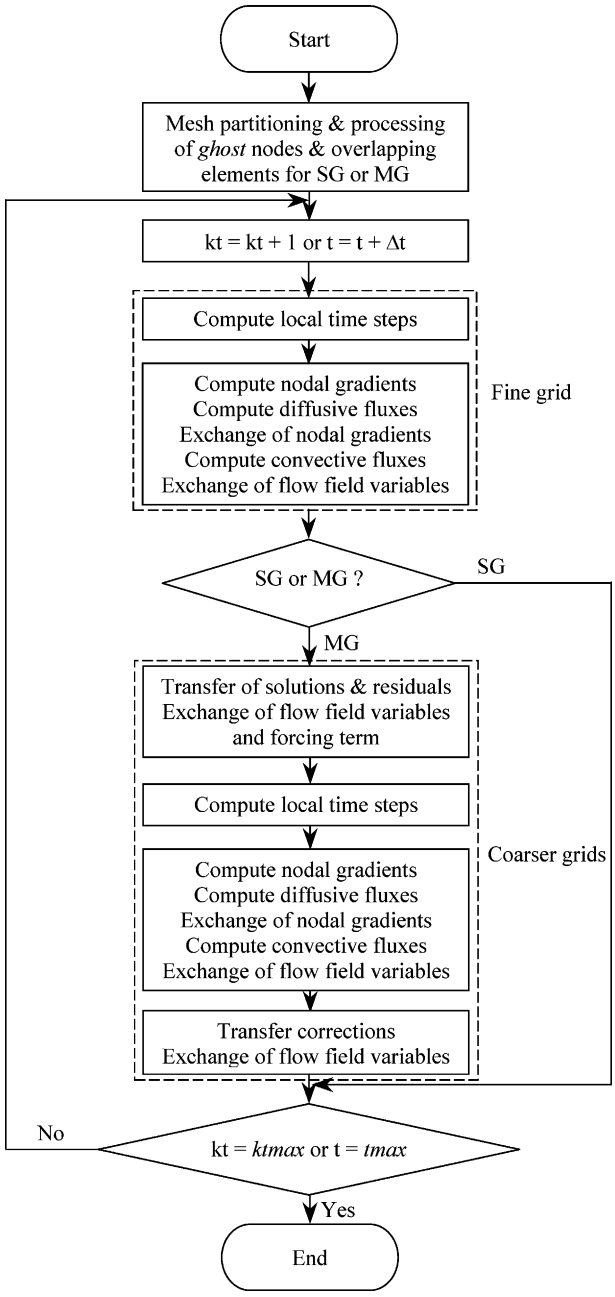


Fig. 9. Flowchart depicting current parallelization strategy.

5.2.1. Identification of ghost nodes and overlapping elements

The nodes and elements that are allocated uniquely to a processor are referred to as core mesh components in this work and each processor calculates the flow field variables and nodal gradients for it. Each sub-domain is enclosed with a layer of *ghost* nodes and overlapping elements, which overlap the sub-domains along the inter-processor boundaries as depicted in Fig. 10. These *ghost* nodes contain flow field

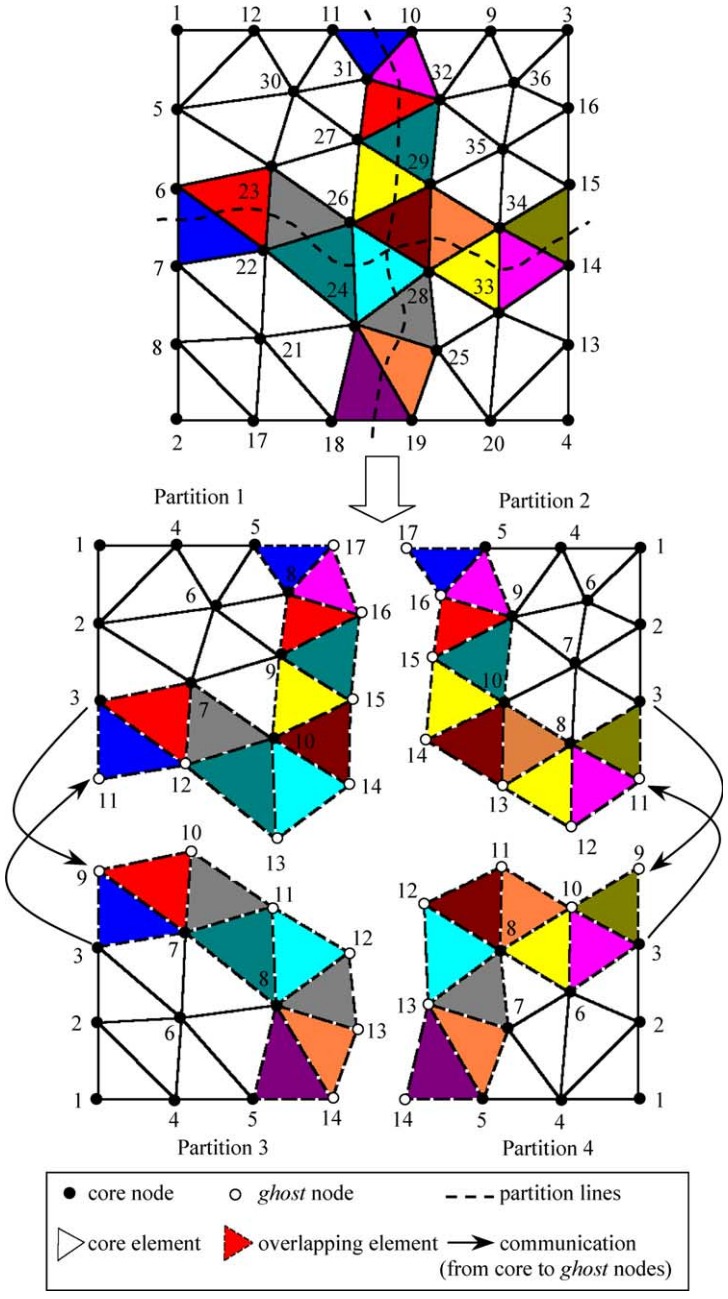


Fig. 10. Mesh decomposed into four partitions showing each sub-domain extended with a layer of *ghost* nodes and overlapping elements.

variables and nodal gradients from neighboring sub-domains that are required for the solution of variables within the sub-domain so as to maintain a solution consistent with the original serial code and no computation is performed on the ghost nodes in the sub-domain under consideration. Communication between these core and *ghost* nodes is based on MPI and the data flow direction is always from core nodes to *ghost* nodes.

An algorithm has been developed to identify the "*ghost nodes*" and overlapping elements and to write the individual grid files with local numbering for each partition. Basically, it consists of three essential steps:
1. Identify *ghost* nodes and overlapping elements between the neighbouring sub-domains.
2. Generate individual grid files with local numbering.
3. Establish data structure for communication between core and *ghost* nodes.

The main concept of this algorithm is that those elements along the inter-processor boundaries and with nodes having different partition numbers are considered as overlapping elements which are cut through by partition lines. And those nodes that formed these elements are a mixture of core and ghost nodes. Basically, the *ghost* node of a partition is the mirror image of the core node of its neighbouring partitions. For example, as shown in Fig. 10, the *ghost* node number 11 of partition 3 is the mirror image of core node number 10 of partition 1 and vice versa. Therefore, making use of this correlation between the core and *ghost* nodes, the rest of the *ghost* nodes can be identified so as to build the control volume for the core node. The concept of this developed algorithm for 2D can be readily extended to 3D and the partitioning of 3D mesh are much more complex than the illustrated 2D mesh. Fig. 10 shows that the irregular *partition lines* bisect the 2D mesh whereas irregular *partition faces* bisect the 3D mesh and the mesh elements are tetrahedral instead of triangles.

5.2.2. Data structure for the exchange of ghost node variables

The communication between the core and *ghost* nodes requires a data structure for each sub-domain, which holds the nodes and processor number to be sent and received. Each processor reflects its re-numbered sub-domain as a complete mesh consisting of 1 to n_e elements and 1 to n_p nodes where n_e and n_p are the local numbers of elements and nodes, respectively. In this work, communication and writing of variables to files are based on local numbering rather than global numbering and therefore, no translation back from local to global numbering is necessary.

With reference to the grid decomposed in Fig. 10, the data structure established for communication between the core and *ghost* nodes is shown in Table 1 and it only shows the data structure for partition 1 and the data structure for the rest of the partitions will follow exactly the same fashion as partition 1. Part of the code for reading the data structure is shown as follows for a processor, *myid*,

```
read (nunit,*) nsenrev
do id = 1,nsenrev
   read(nunit,*) igver,nprtn,icore,npart,ighst
   if (nprtn.eq.myid) then
      nsend(npart,mg) = nsend(npart,mg) + 1
      sendvert(npart,nsend(npart,mg),mg) = icore
   else
      nrecv(nprtn,mg) = nrecv(nprtn,mg) + 1
      recevert(nprtn,nrecv(nprtn,mg),mg) = ighst
 endif
enddo
```

where **nsenrev** is the total number of nodes to be sent and received, **sendvert** is a three-dimensional array that contains the locally numbered core nodes, **icore**, which it must *send* to the partition number, **npart**, for each level of multigrid, mg, and **recevert** is similar to the array **sendvert** except that it contains the locally numbered *ghost* nodes, **ighst**, which it must *receive* from the partition number, **nprtn**.

On most parallel computers, moving data from one processor to another takes more time than moving or manipulating data within a single processor. To prevent a program from being slowed down excessively, many parallel computers allow users to start sending or receiving several messages and to proceed with other operations. Message passing is done asynchronously using the buffered

Table 1
Communication between core and *ghost* nodes for partition 1 depicted in Fig. 10

Global node number	Send		Receive	
	Partition number	Local node number (core node)	Partition number	Local node number (*ghost* node)
igver	**nprtn**	**icore**	**npart**	**ighst**
11		5	2	17
31		8	2	16
27		9	2	15
26	1	10	2	14
26		10	3	11
23		7	3	10
6		3	3	9
26		10	4	12
10		5		17
32	2	9		16
29		10	1	15
24		8		13
22	3	7		12
7		3		11
28	4	8		14

mpi_bsend( ) routine in this work. This allows the messages to be placed in the buffer until it is delivered. Once all the flow field variables or nodal gradients have been sent, the processes are ready to receive variables or nodal gradients from the respective processes. Messages are received by the standard blocking **mpi_recv( )** routine. In order to reduce the communication time, the flow field variables (ua( )) or nodal gradients are packed into a single array (sendsol( )) before sending it to the respective partitions, as depicted in part of the code shown below. The flow field variables or nodal gradients are unpacked from the single array (recvsol( )) after being received from the respective partitions.

```
      do iv = 1,nsend(npart,mg)
         is = sendvert(npart,iv,mg)
         do ivar = 1,nvar
            iq = iq + 1
            sendsol(iq) = ua(ivar,is,mg)
         enddo
      enddo
      call mpi_bsend(sendsol,(nsend(npart,mg)*nvar),
     & mpi_real,npart,myid,mpi_comm_world,ierr)
      call mpi_recv(recvsol,(nrecv(npart,mg)*nvar),
     & mpi_real,npart,npart,mpi_comm_world,istatus,ierr)
      do iv = 1,nrecv(npart,mg)
         is = recevert(npart,iv,mg)
         do ivar = 1,nvar
            id = id + 1
            ua(ivar,is,mg) = recvsol(id)
         enddo
      enddo
```

5.3. Multigrid parallelization

The MG-DD approach is adopted in this study. This means that the non-nested multigrids are independently generated first. Then domain decomposition of the finest grid is performed, which is followed by decomposition of the various coarse levels of grids guided by the finer grid partitions. This is achieved by using the fine grid partitions to infer the coarse level partitions (i.e., the coarse grid is to inherit its partition from that of the corresponding fine grid) and load balancing in the coarse mesh is reasonably well ensured. In the multigrid process, the flow field variables and residuals of the coarse grid nodes are obtained directly from their corresponding fine grid nodes while ghost nodes are updated by inter-processor communication. A two level multigrid and two sub-domains are used to describe the procedure of partitioning the coarse grid using the fine grid. The main idea about this multigrid parallelization algorithm is that the fine grid is partitioned into two sub-domains according to the algorithm developed for single grid depicted in Section 5.2. And both the maximum and minimum values in the x- and y-direction ($x_{\min}$, $x_{\max}$, $y_{\min}$ and $y_{\max}$) of each partition for the fine grid are noted. With these dimensions, an imaginary bounding box enclosing the sub-domains is formed as shown in Fig. 11. The main purpose of these bounding boxes is to identify the coarse nodes that fall within these boxes according to the fine grid partitioned including those nodes beyond the sub-domains boundary. The algorithms depicted in Section 4.3.1 is used to search for those actual coarse nodes that fall within the fine grid sub-domain and those nodes that fall beyond the sub-domain boundary (i.e., shaded portion as depicted in Fig. 11) are ignored. After classifying the respective coarse nodes according to which partition they belong to, the *ghost* nodes and overlapping elements are identified using the algorithm depicted in Section 5.2.1. The individual grid file for the partitioned coarse grid and data structure for communication is then generated.

5.4. Measuring performance

Both speed-up and efficiency are commonly used to measure the performance of a parallel code. The run time of the original serial code is used as a measure of the run time on one processor. In this study, the run time starts after mesh partitioning for either single grid or multigrid, identifying *ghost* nodes and writing local grid files for different partitions have been performed, and up to the time when writing all the results to the respective files have been undertaken. Both CPU time and wall-clock time are used to record the run time. The main difference between them is that CPU time is the recorded time when the processor performs

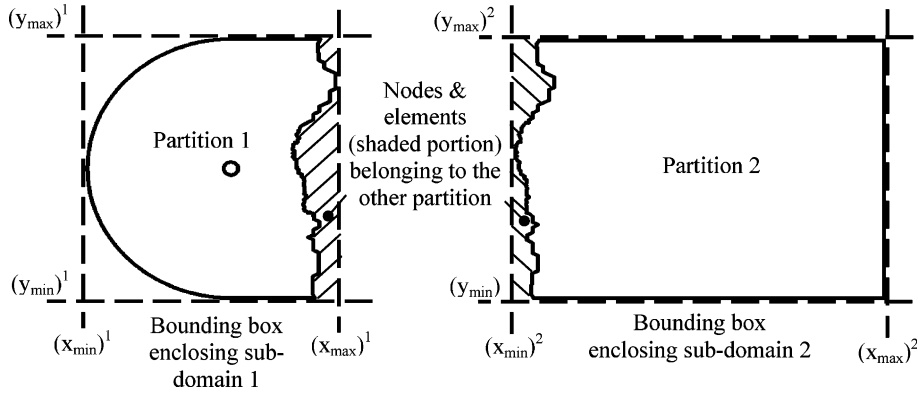


Fig. 11. Fine grid partitions to infer the coarse grid partitions.

a calculation whereas wall-clock time includes communication time and idling time when the processor idles while waiting for other processors to communicate or other jobs to be processed. The wall-clock time is used to represent the total run time in this study since it includes the idling time and communication time. Computation time is the time spent on computing and communication time is the total time spent on exchange of flow field variables, nodal gradients and forcing terms.

The parallel speed-up S_p is the ratio of the run-time on one processor t_1 to the run-time on P processors, t_p [24] with only one user running a single job on the computers:

$$S_p = \frac{t_1}{t_p}. \tag{53}$$

If the parallelization is 100% efficient, then $S_p = P * 100\%$, but this is rarely the case as both communication and load balancing adversely affect the performance of parallel computation. In this work, the total simulation wall-clock time, t_p, is used as the run-time to compute the speedup and it is the maximum wall-clock time among all the processors:

$$t_p = \max\left(t^j_{cpu} + t^j_{idle}\right), \tag{54}$$

where j is the processor number, t^j_{cpu} and t^j_{idle} are the CPU time and idling time for processor j, respectively. And the CPU time for processor j is defined as follows:

$$t^j_{cpu} = t^j_{comp} + t^j_{comm}, \tag{55}$$

where t^j_{comp} and t^j_{comm} are the computation and communication time for processor j, respectively. The overall communication time is defined as the maximum communication time among all the processors:

$$t_{comm} = \max\left(t^j_{comm}\right) = \max\left(\sum_j t^{FFV}_{comm} + t^{NG}_{comm} + t^{FT}_{comm}\right), \tag{56}$$

where t^{FFV}_{comm}, t^{NG}_{comm} and t^{FT}_{comm} are the communication time for exchanging of flow field variables, nodal gradients and forcing terms, respectively. And the representative computation time of a problem is defined as

$$t_{comp} = t_p - t_{comm}. \tag{57}$$

The parallel efficiency is sometimes used as the performance measure for a parallel code. Parallel efficiency E_p is simply the ratio of the parallel speed-up S_p to the number of processor P [24]

$$E_p = \frac{S_p}{P}, \tag{58}$$

where $0 \leqslant E_p \leqslant 1$.

6. Boundary and initial conditions

At the solid wall, a *no-slip* condition is imposed for viscous flow by setting the flow velocity equal to that of the body. A uniform velocity profile is given as the free stream boundary condition and the pressure at the free stream is calculated while pressure at the down stream boundary is fixed at a constant value and the velocity at the down stream boundary is calculated. The flow field values are set to the free-stream values at the start of the computation.

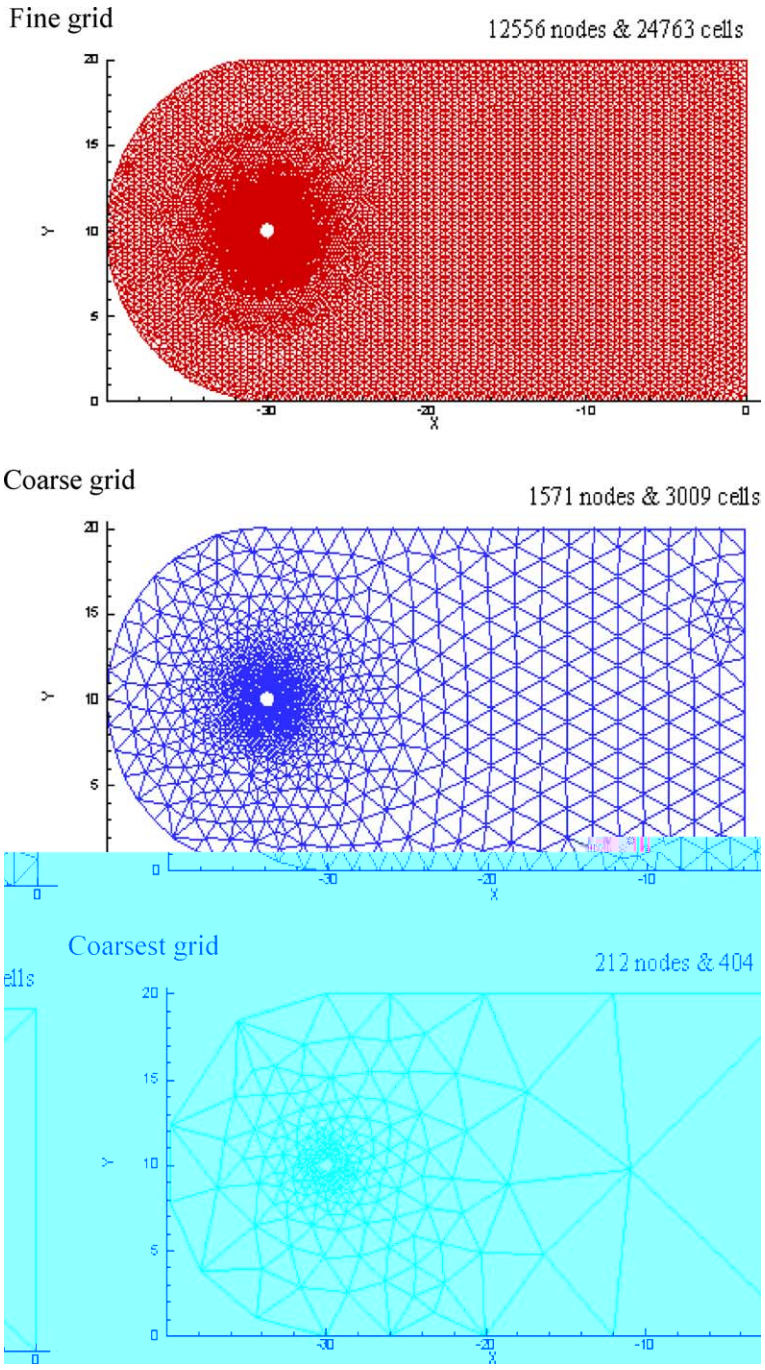


Fig. 12. The sequence of grids used in the three-level multigrid computations.

7. Computational results

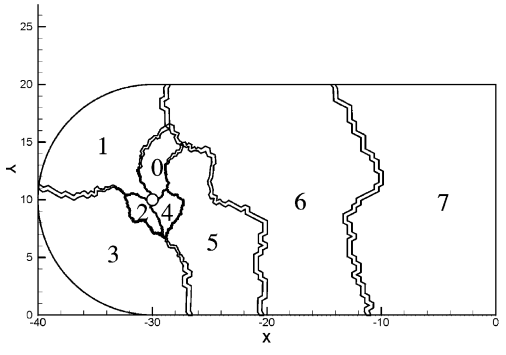
In this section, the test case employed to examine the convergence behaviour and speed-up characteristics of the parallel unstructured single grid (SG) and multigrid (MG) solvers is viscous flow over a circular cylinder at both low and high Reynolds numbers.

7.1. Steady viscous flow

In this computation, the Reynolds numbers (Re) specified is 41.0, the pseudo-compressibility coefficient β is set to 1.0 for fast convergence and the physical time-step was set to a large value. The third-order characteristics-based scheme was used in both parallel single grid and multigrid computations. A three-level multigrid using both the V- and W-cycles is used for the latter and the number of multigrid cycles per time-step was set to 100. There are 12,556 nodes and 24,763 elements in the fine grid, 1571 nodes and 3009 elements in the coarse grid, and 212 nodes and 404 elements in the coarsest grid, all of which are shown in Fig. 12. Fig. 13 shows the mesh partition information for 8 partitions in the mesh. Both serial and parallel computations were performed on an SGI Origin 2000 machine with 32 processors, 500 MFLOPS/CPU, 8 nodes and each node has four MPIs 64-Bit R10000 250 MHz RISC CPUs.

The parallel solutions obtained are used to make a qualitative comparison with those results obtained from serial codes and experimental results [25]. They agree well with both numerical and experimental solutions as depicted in Fig. 14. The performance results for both parallel SG and MG in term of speedup characteristics, efficiency of parallelization and comparison between percentage computation and communication time are shown in Figs. 15–17, and timings for all these performance results are tabulated in Tables 2 and 3. These figures give the detailed statistics of the parallel codes run on 1, 2, 4, 8 and 16 processors. In all cases, timings are measured for the single grid solver and the MG method using both V- and W-cycles and a high degree of parallelism is achieved. All the speedups obtained in this work are compared to the perfect speedup.

The plot shown in Fig. 15 reveals that the speedup increases almost linearly up to four processors, after which, it starts to deviate away from the perfect speedup. This is to be expected, since the relative ratio of communication to computation is higher as the number of processor increases and inevitably the speedup declines. But the simulation time decreases as the number of processor increases and the time taken to complete the simulation reduces accordingly. The speedups for both parallel SG and MG show a similar trend, but the computational time required to compute the same problem by parallel-MG is much less than parallel-SG. Comparing the speedups obtained for both multigrid V- and W-cycles, it is noted that the



No. of Partitions = 8			
Partition Number	Core Nodes	*Ghost* Nodes	Elements
0	1521	145	3170
1	1608	155	3314
2	1556	151	3246
3	1542	136	3166
4	1599	140	3317
5	1510	191	3193
6	1616	67	3181
7	1604	147	3292

Fig. 13. Mesh partition information for 8-partition mesh with 12,556 nodes.

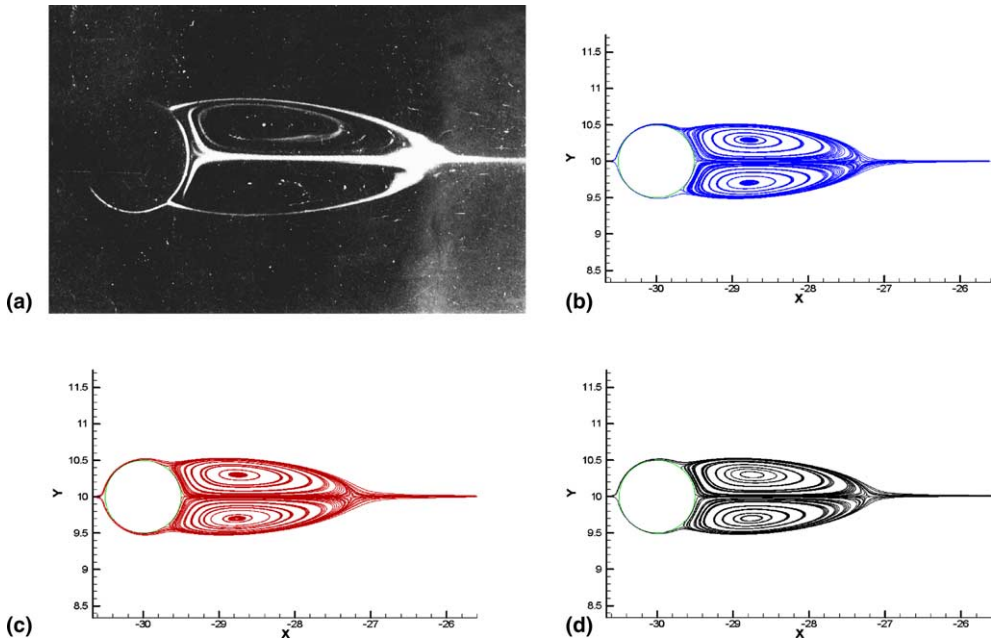


Fig. 14. Streamlines plot for flow over a circular cylinder for (a) experimental measurements [25], parallel computation for (b) single grid, (c) MG, V-cycle and (d) MG, W-cycle ($Re = 41.0$).

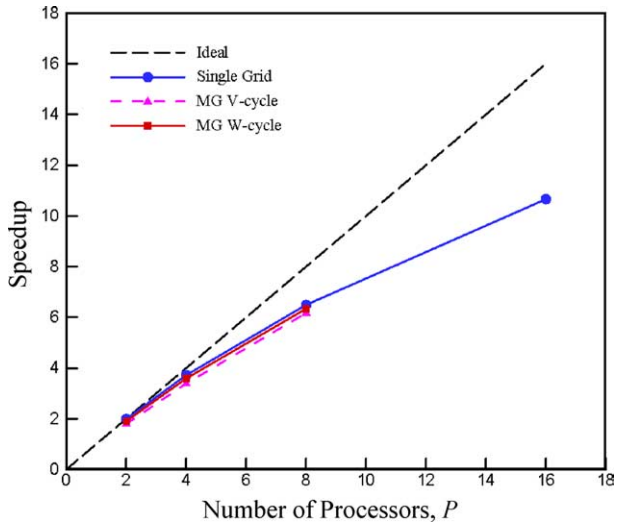


Fig. 15. Speedups for both parallel SG and MG ($Re = 41.0$).

speedup for W-cycle is slightly better than V-cycle. This may be partly due to the fact that the specified number of multigrid cycles is less for the W-cycle to compute this problem and therefore, less communication is involved during the transferring of variables and residuals.

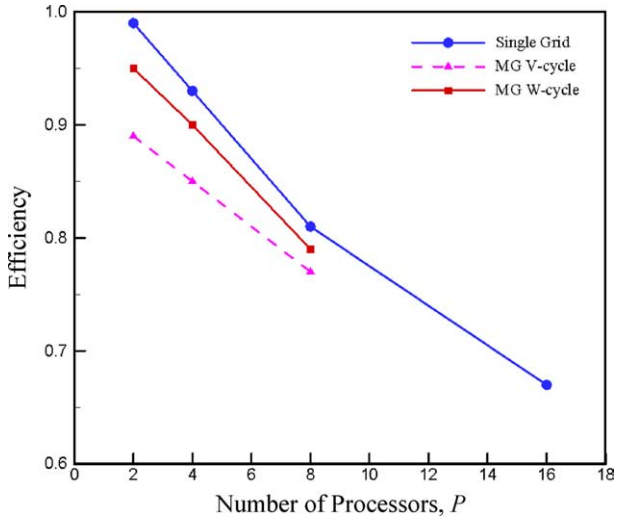


Fig. 16. Efficiency for both parallel SG and MG ($Re = 41.0$).

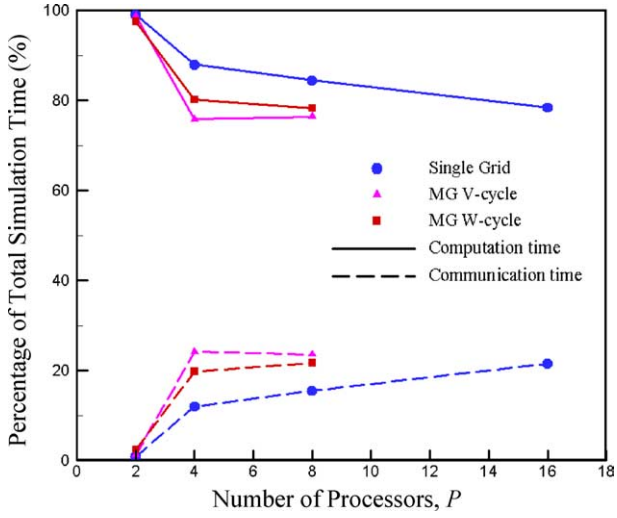


Fig. 17. Comparison between computation and communication time ($Re = 41.0$).

The performance results for parallel runs on 16 and 32 processors for both single grid and multigrid are not presented in this study partly due to the fact that the mesh size is quite small and the runs are greatly affected by other jobs running concurrently in the machine. The speedups obtained from these runs are low and are not representative of the actual performance.

The efficiency declines steadily as the number of processors increases as shown in Fig. 16. This is to be expected, since the speedup deviates away from the perfect speedup shown in Fig. 15 as the number of processor increases. Although the computation time has reduced significantly running on 16 processors, it is observed that the number of processors to achieve an optimum speedup and efficiency is 8 processors. This is because the number of users and their load varied during the simulation and by using fewer processors may even produce better efficiency out of the machine.

Table 2
Timings for sequential and parallel single grid and multigrid (V- and W-cycles) for $Re = 41$

Serial/ Parallel	No. of Processor(s)	Single grid				Multigrid (V-cycle)				Multigrid (W-cycle)			
		CPU time (min)	Wall-clock time (min)	Speedup	Efficiency	CPU time (min)	Wall-clock time (min)	Speedup	Efficiency	CPU time (min)	Wall-clock time (min)	Speedup	Efficiency
Serial	1	540.79	543.02	–	–	232.52	234.88	–	–	227.49	233.18	–	–
Parallel	2	271.52	273.25	1.99	0.99	130.28	131.40	1.79	0.89	121.57	122.50	1.90	0.95
	4	145.10	145.98	3.72	0.93	68.71	69.35	3.39	0.85	64.53	65.03	3.59	0.90
	8	83.09	83.67	6.49	0.81	37.72	38.05	6.17	0.77	36.46	36.80	6.34	0.79
	16	50.50	50.92	10.66	0.67	–	–	–	–	–	–	–	–

Table 3
Comparison between computation and communication time for parallel single grid and multigrid (V- and W-cycles) for $Re = 41$

No. of processors	Single grid			Multigrid (V-cycle)			Multigrid (W-cycle)		
	Computation time (min) (A)	Communication time (min) (B)	Total simulation time (min) (C) = (A) + (B)	Computation time (min) (D)	Communication time (min) (E)	Total simulatior time (min) (F) = (D) + (E)	Computation time (min) (G)	Communication time (min) (H)	Total simulation time (min) (I) = (G) + (H)
2	270.72 (99.07%)	2.53 (0.93%)	273.25	129.67 (98.68%)	1.73 (1.32%)	131.40	119.48 (97.53%)	3.02 (2.47%)	122.50
4	128.40 (87.96%)	17.58 (12.04%)	145.98	52.57 (75.80%)	16.78 (24.20%)	69.35	52.16 (80.21%)	12.87 (19.79%)	65.03
8	70.67 (84.46%)	13.00 (15.54%)	83.67	29.07 (76.40%)	8.98 (23.60%)	38.05	28.80 (78.26%)	8.00 (21.74%)	36.80
16	39.94 (78.44%)	10.98 (21.56%)	50.92	–	–	–	–	–	–

One of the reasons why speedup and efficiency deteriorate as the number of processor increases is the increase in communication time. This phenomenon can be analysed in Fig. 17, where there is a steep increase and decrease in percentage communication time and computation time, respectively, from two to four processors. And from four processors onwards, the rate of change is quite steady. For parallel-SG running on two processors, only 0.93% of the total simulation time is spent on communication and the rest is spent on computation, whereas, running on 16 processors, 21.56% of the time is spent on communication and the rest on computation. This increase in communication overhead is mainly due to the increasing number of adjacent sub-domains, which increases the number of messages that require transmission. Likewise, increasing the number of partitions will definitely decrease the computation time due to the fact that the load on each processor is much smaller.

The main reason for the poor scaling of parallel-MG computation, as compared with parallel-SG computation, is that the communication time for the parallel-MG computation is significantly higher than for the parallel-SG case for the same number of processors. This is because the multigrid algorithm performs additional coarse grid sweeps compared with the parallel-SG. Similarly, multigrid W-cycle spends less time on communication as compared with its counterpart, the V-cycle. This may be partly due to the fact that the specified number of multigrid cycles is less for the W-cycle to solve this problem and therefore, less communication is involved in transferring variables and residuals.

Fig. 18 shows the convergence history plot for the various numbers of partitions for single grid simulation. It can be seen that the trend for the residual reduction for the various numbers of partitions follow the same trend as the single grid and the residuals decrease to the same order of magnitude for all partitions. Based on this figure, it can be seen that the parallel strategy adopted is effective and will not affect the convergence rate of the solution regardless of the numbers of partitions used.

7.2. Unsteady viscous flow

The test case considered for high-Reynolds-number unsteady flow is the viscous flow over a circular cylinder at $Re = 200$, which is one of the most thoroughly investigated unsteady flow cases. The parameters and schemes for both the parallel single-grid (SG) and multigrid (MG) in the computations were the same as before, except that the number of pseudo sub-iterations and number of multigrid cycles per step were set differently. The non-dimensional physical time-step was set to be 0.09 for better temporal resolution. The

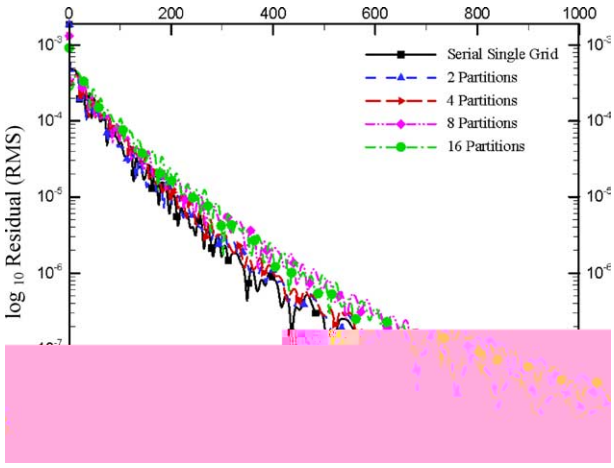


Fig. 18. Convergence history plot for various numbers of partitions (single grid).

third-order characteristics-based scheme was also used in both parallel SG and MG computations together with the dual-time-stepping scheme and the second-order temporal discretization. The pseudo sub-iterations per time-step were set to 200 for single grid, 30 V-cycles per time-step and 15 W-cycles per time step. A three-level multigrid was used to compute the flow. There are 24,226 nodes and 48,103 elements in the fine grid, 8646 nodes and 17,102 elements in the coarse grid, and 3139 nodes and 6139 elements in the coarsest grid. Especially in the fine grid the wake region is further refined in order to accurately capture the fine details of the vortex shedding phenomenon as shown in Fig. 19.

The high-Reynolds-number-flow calculations for both serial and parallel computations were performed on the SGI Origin 2000 machine. The flow was started from stationary conditions and the simulation was run until periodic shedding of vortices occurred. Fig. 20 shows the computed lift and drag coefficients on the cylinder versus non-dimensional time with both parallel SG and MG, respectively, for an 8-partition mesh. The lift coefficient, C_l, drag coefficient, C_d, and Strouhal number, St, obtained in this work are ± 0.64,

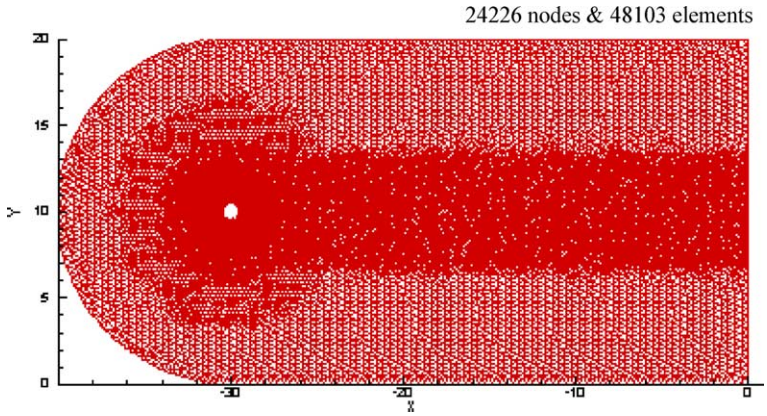


Fig. 19. Fine mesh with grid refinement in the wake region for unsteady flow calculation.

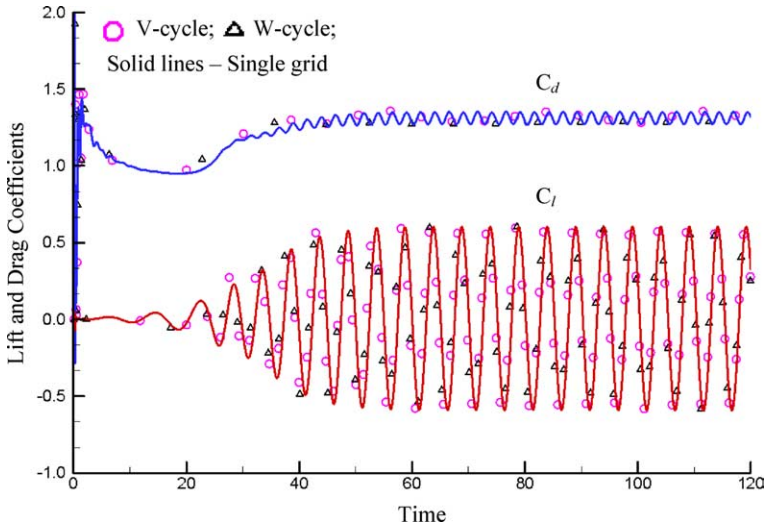


Fig. 20. Lift and drag coefficients versus time for flow over a circular cylinder ($Re = 200$).

Table 4
Comparison of results for unsteady flow over a cylinder, $Re = 200$

Reference	C_l	C_d	St
Present (parallel-multigrid)	± 0.64	1.31 ± 0.041	0.195
Liu et al. [11]	± 0.69	1.31 ± 0.049	0.192
Chan and Anastasiou [14]	± 0.63	1.48 ± 0.05	0.183
Belov et al. [10]	± 0.64	1.19 ± 0.042	0.193
Rogers and Kwak [13]	± 0.65	1.23 ± 0.05	0.185
Rosenfield et al. [30]	± 0.69	1.46 ± 0.05	0.211
Lecointe and Piquet – second order [31]	± 0.70	1.46 ± 0.04	0.227
Lecointe and Piquet – fourth order [31]	± 0.50	1.58 ± 0.0035	0.194
Lin et al. [32]	–	1.17	–
Kovaznay (Expt.) [28]	–	–	0.19
Roshko (Expt.) [29]	–	–	0.19
Wille (Expt.) [27]	–	1.30	–

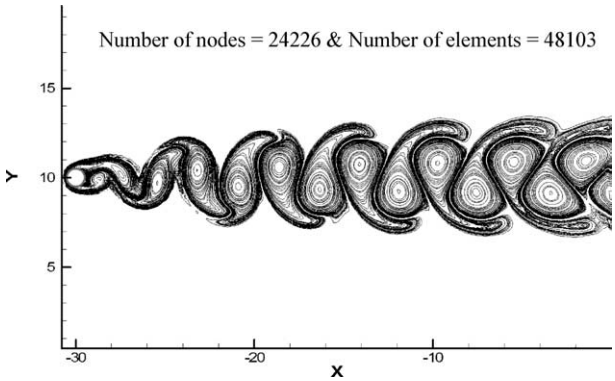


Fig. 21. Vorticity contours plot for multigrid ($Re = 200$).

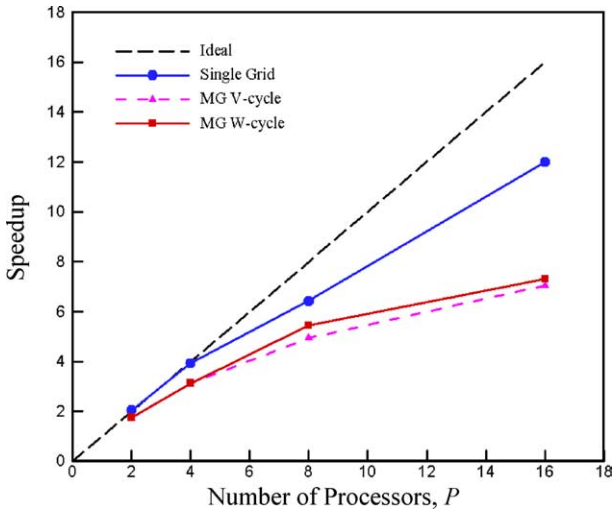


Fig. 22. Speedups for both parallel SG and MG ($Re = 200.0$).

1.31 ± 0.041 and 0.195, respectively, and they agree well with numerical solutions obtained by other researchers as well as with experimental measurements [10,11,13,14,26–32] and these results are tabulated in Table 4. Fig. 21 shows the contours of vorticity obtained by the parallel-MG method, which basically outlines the von Kármán vortex street phenomenon. The figure shows that the vortices with opposite signs are shed from upper and lower surfaces alternately, thus forming the von Kármán vortex street phenomenon.

The performance results for both parallel SG and MG are shown in Figs. 22–24, in terms of speedup characteristics, parallelization efficiency and percentage computation and communication time. Timings for all these performance results are also tabulated in Tables 5 and 6. Fig. 22 shows the speedup *vs.* number of

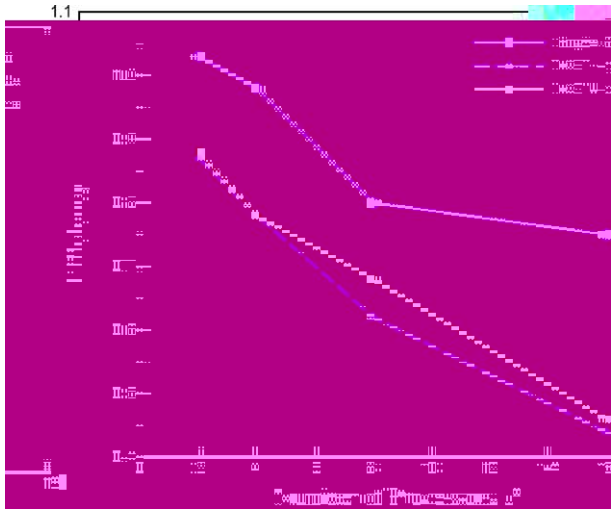


Fig. 23. Efficiency for both parallel SG and MG ($Re = 200.0$).

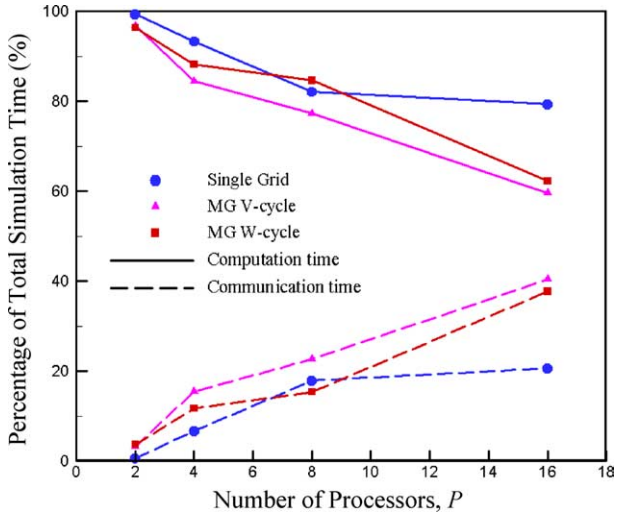


Fig. 24. Comparison between computation and communication time ($Re = 200.0$).

Table 5
Timings for serial and parallel single grid and multigrid (V- and W-cycles) to reach $t = 80.0$ using different number of sub-iterations/multigrid cycles per time-step for $Re = 200$

Serial/ Parallel	No. of processor(s)	Single grid				Multigrid (V-cycle)				Multigrid (W-cycle)			
		CPU time (min)	Wall-clock time (min)	Speedup	Efficiency	CPU time (min)	Wall-clock time (min)	Speedup	Efficiency	CPU time (min)	Wall-clock time (min)	Speedup	Efficiency
Serial	1	2140.10	2148.43	–	–	1055.07	1064.47	–	–	1081.04	1086.0	–	–
Parallel	2	1040.06	1045.57	2.05	1.03	605.65	609.63	1.75	0.87	615.37	618.57	1.76	0.88
	4	544.61	547.22	3.93	0.98	338.87	340.92	3.12	0.78	343.42	346.42	3.13	0.78
	8	332.58	334.37	6.43	0.80	213.72	215.27	4.94	0.62	198.09	199.42	5.45	0.68
	16	177.97	178.98	12.00	0.75	149.98	151.07	7.05	0.44	147.32	148.47	7.31	0.46

Table 6
Comparison between computation and communication time for parallel single grid and multigrid (V- and W-cycles) to reach $t = 80.0$ using different number of sub-iterations/multigrid cycles per time-step for $Re = 200$

No. of processors	Single grid			Multigrid (V-cycle)			Multigrid (W-cycle)		
	Computation time (min) (A)	Communication time (min) (B)	Total simulation time (min) (C) = (A) + (B)	Computation time (min) (D)	Communication time (min) (E)	Total simulation time (min) (F) = (D) + (E)	Computation time (min) (G)	Communication time (min) (H)	Total simulation time (min) (I) = (G) + (H)
2	1038.50 (99.33%)	7.07 (0.67%)	1045.57	590.18 (96.81%)	19.45 (3.19%)	609.63	595.77 (96.31%)	22.80 (3.69%)	618.57
4	510.58 (93.30%)	36.64 (6.70%)	547.22	288.04 (84.49%)	52.88 (15.51%)	340.92	305.60 (88.22%)	40.82 (11.78%)	346.42
8	274.49 (82.09%)	59.88 (17.91%)	334.37	166.42 (77.31%)	48.85 (22.69%)	215.27	168.70 (84.60%)	30.72 (15.40%)	199.42
16	142.01 (79.34%)	36.97 (20.66%)	178.98	90.04 (59.60%)	61.03 (40.40%)	151.07	92.48 (62.29%)	55.98 (37.71%)	148.47

processors, where the plot reveals that the speedup for parallel-SG shows a 2.05 speedup for 2 processors and a near perfect speedup for four processors. This near perfect speedup reflects the effectiveness of the parallelization strategy for single grid. The speedup performance starts to deteriorate as the number of processors increases, but the simulation time has decreased significantly. From the figure, it can be seen that the speedup is not that significant for the parallel-MG as compared to the parallel-SG. This is mainly due to the additional overheads of transferring variables between the coarse grids, which increases the communication time. Although the speedup is lower for the parallel-MG as compared with the parallel-SG method due to additional overheads, the overall simulation time required to compute the same problem by parallel-MG is still much less than parallel-SG. For instance, the simulation time using eight processors for the multigrid V-cycle method is 215.27 min, while the single-grid method would need 334.37 min, as found in Table 5. Comparing the speedups obtained for both multigrid V- and W-cycle, it is noted that the speedup for both two and four processors coincide with each other and from this point onwards, they start to deviate away from each other. In this test case, the speedup for W-cycle is found to be better than V-cycle. This is mainly due to the smaller communication overheads involved in W-cycle. For a fixed problem, as the number of processors increases, the computation time on each processor decreases while the communication time increases. It is mainly due to this change in the ratio of calculation to communication that leads to the drop in speed-up as the number of processors increases. From Fig. 22, it can be seen that the computation is scalable up to 16 processors. The efficiency plot shown in Fig. 23 is extremely low in this test case for multigrid methods running on 16 processors, below 50% as shown in the figure. Therefore, it is more efficient to use eight processors to simulate this problem, rather than 16 processors. On the contrary, parallel-SG proved to be quite efficient running on 16 processors, indicating an efficiency of 75%, but the optimum performance is obtained running on eight processors.

Fig. 24 shows the percentage computation time and communication time required in various computations. The lines representing both the percentage computation time and communication time are moving towards each other as the number of processors increases. This is to be expected, since computation time and communication time will decrease and increase, respectively, inversely proportional to each other as the number of processor increases. The figure shows that the percentage communication time is much lower for parallel-SG as compared to parallel-MG running on 16 processors because no additional coarse grid sweeps are performed on the single grid code. This also explains why SG computation has much higher parallel efficiency. The communication time is extremely high for the multigrid method running on 16 processors, which accounts for almost 40% of the total simulation time as indicated in Fig. 24. Although the percentage communication time is high, the percentage computation time is getting smaller using 16 processors for the multigrid method: only 60% of the total simulation time. It can be seen from the figure that the communication overheads for the W-cycle is lower than the V-cycle and parallel-SG running on eight processors. That is why it is more appropriate and advantageous to simulate problems using the multigrid W-cycle over the other two methods.

8. Conclusions

A finite volume Navier–Stokes solver on unstructured grids using a higher-order characteristics-based upwind scheme and dual time-stepping have been extended to implement the parallel multigrid algorithm for the study of unsteady incompressible viscous flows. The parallel multigrid method is based on the multigrid domain decomposition (MG-DD), the single program multiple data (SPMD) programming paradigm and message-passing interface (MPI). The speedups and efficiencies obtained by both parallel-SG and MG solvers are reasonably good for both test cases. A maximum speedup of 12 could be achieved on 16 processors for high-Reynolds number unsteady viscous flow. The results obtained were compared with those results obtained using serial single grid and multigrid codes and the parallel solution remains the same

as those obtained by serial solvers. Moreover, the results agree well with numerical solutions obtained by other researchers as well as experimental measurements. The computation time and communication time for both test cases are quite reasonably balanced as the number of processors increased. It is observed that the optimum number of processors to simulate both test cases is 8.

Acknowledgements

This research work was supported by a research scholarship provided by Nanyang Technological University (NTU) and Agency for Science, Technology and Research (A*STAR), Singapore. The provision of computing facilities by Nanyang Centre for Supercomputing and Visualization (NCSV), NTU is acknowledged.

References

[1] M. Lallemand, H. Steve, A. Dervieux, Unstructured multigridding by volume agglomeration, Comput. Fluids 21 (1992) 397–433.
[2] D. Mavriplis, Three-dimensional multigrid for the Euler equations, AIAA J. 30 (1992) 1753–1761.
[3] D. Mavriplis, V. Venkatakrishnan, Agglomeration multigrid for two dimensional viscous flows, Comput. Fluids 24 (1995) 553–570.
[4] J. Peraire, K. Morgan, J. Peiro, Multigrid solution of the 3D compressible Euler equations on unstructured tetrahedral grids, Internat. J. Numer. Methods Engrg. 36 (1993) 1029–1044.
[5] J. Farmer, L. Martinelli, A. Jameson, Fast multigrid method for solving incompressible hydrodynamic problems with free surface, AIAA J. 32 (1994) 1175–1182.
[6] A. Rizzi, L. Erikson, Computation of inviscid incompressible flow with rotation, J. Fluid Mech. 153 (1985) 275–312.
[7] J. Peraire, K. Morgan, J. Peiro, The simulation of 3D incompressible flows using unstructured grids, in: D.A. Caughey, M.M. Hafez (Eds.), Frontier of Computational Fluid Dynamics 1994, John Wiley & Son, New York, 1994, pp. 281–301.
[8] W.K. Anderson, R.D. Rausch, D.L. Bonhaus, Implicit/multigrid algorithm for incompressible turbulent flows on unstructured grids, J. Comput. Phys. 128 (1996) 391–408.
[9] A. Chorin, A numerical method for solving incompressible viscous flow problems, J. Comput. Phys. 2 (1) (1967) 12–26.
[10] A. Belov, L. Martinelli, A. Jameson, A new implicit algorithm with multigrid for unsteady incompressible flow calculations, AIAA 95-0049, January 9–12, 1995.
[11] C. Liu, X. Zheng, C.H. Sung, Preconditioned multigrid methods for unsteady incompressible flows, J. Comput. Phys. 139 (1998) 35–57.
[12] P.T. Lin, Implicit time dependent calculations for compressible and incompressible flows on unstructured meshes, MSc Thesis, Department of Mechanical and Aerospace Engineering, Princeton University, 1994.
[13] S.E. Rogers, D. Kwak, Upwind differencing scheme for the time-accurate incompressible Navier–Stokes equations, AIAA J. 28 (2) (1990) 253.
[14] C.T. Chan, K. Anastasiou, Solution of incompressible flows with or without a free surface using the finite volume method on unstructured triangular meshes, Internat. J. Numer. Method Fluids 29 (1995) 35–57.
[15] D.J. Mavriplis, Large-scale parallel viscous flow computations using an unstructured multigrid algorithm, NASA/CR-1999-209724, ICASE Report No. 99-44, November 1999.
[16] D.J. Mavriplis, Parallel unstructured mesh analysis of high-lift configurations, AIAA Paper 2000-0923, 38th AIAA Aerospace Sciences Meeting, January 2000.
[17] I.M. Llorente, M. Prieto-Matias, B. Diskin, An efficient parallel multigrid solver for 3-D convection-dominated problems, NASA/CR-2000-210319, ICASE Report No. 2000-29, August 2000.
[18] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, The MIT Press, Cambridge, MA, 1994.
[19] D. Dirikakis, P.A. Govatsos, D.E. Papantonis, A characteristic-based method for incompressible flows, Internat. J. Numer. Methods Fluids 19 (8) (1994) 667–685.
[20] A. Eberle, Three-dimensional Euler calculations using characteristics flux extrapolation, AIAA Paper 85-0119, 1985.
[21] Y. Zhao, B.L. Zhang, A high-order characteristics upwind FV method for incompressible flow and heat transfer simulation on unstructured grids, Comput. Methods Appl. Mech. Engrg. 25 (6) (2001) 523–536.

[22] G. Karypis, V. Kumar, A parallel algorithm for multilevel graph partitioning and sparse matrix ordering, J. Parallel Distrib. Comput. 48 (1998) 71–85.
[23] G. Karypis, V. Kumar, Metis: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0, Department of Computer Science, University of Minnesota, 1998.
[24] S.P. Peter, Parallel Programming with MPI, Morgan Kaufmann, San Francisco, CA, 1997.
[25] Van Dyke, D. Milton, An Album of Fluid Motion, Parabolic Press, Stanford, CA, 1982.
[26] M. Nishioka, H. Sato, Mechanism of determination of the shedding frequency of vortices behind a cylinder at low Reynolds number, J. Fluid Mech. 89 (1978) 49–60.
[27] R. Wille, Karman Vortex Streets, in: Advances in Applied Mechanics, vol. 6, Academic, New York, 1960, pp. 273–287.
[28] L.S.G. Kovasznay, Hot-wire investigation of the wake behind cylinders at low Reynolds numbers, Proc. R. Soc. Lond., Ser. A 198 (1053) (1949) 174–190.
[29] A. Roshko, On the development of turbulent wakes from vortex streets, NACA Report, 1191, 1954.
[30] M. Rosenfeld, D. Kwak, M. Vinokur, A solution method for the unsteady and incompressible Navier–Stokes equations in generalized coordinate systems, AIAA Paper 88-0718, 1988.
[31] Y. Lecointe, J. Piquet, On the use of several compact methods for the study of unsteady incompressible viscous flow round a circular cylinder, Comput. Fluids 12 (4) (1984) 255–280.
[32] C.L. Lin, D.W. Pepper, S.C. Lee, Numerical methods for separated flow solutions around a circular cylinder, AIAA J. 14 (1976) 900–907.
[33] Y. Zhao, C.H. Tai, Higher-order characteristics-based methods for incompressible flow computation on unstructured grids, AIAA J. 39 (7) (2001) 1280–1287.
[34] Y. Zhao, C.H. Tai, F. Ahmed, Simulation of micro flows with moving boundaries using high-order upwind FV method on unstructured grids, Comput. Mech. 28 (1) (2002) 66–75.
[35] R. Lohner, CFD via unstructured grids: trends and applications, in: D.A. Caughey, M.M. Hafez (Eds.), Frontier of Computational Fluid Dynamics 1994, John Wiley & Son, New York, 1994, pp. 117–133.