Transaction B: Mechanical Engineering
Vol. 17, No. 6, pp. 443–455

SCIENTIA IRANICA

Rayan: A Polyhedral Grid Co-located Incompressible Finite Volume Solver (Part I: Basic Design Features)

M. Sani[1] and M.S. Saidi[1,*]

Abstract. *In this work, basic design features of Rayan are documented. One of the new design features presented in this work is the way Rayan handles polyhedral grids. Grid definition is combined with the definition of the structure of the sparse coefficient matrix, thereby releasing a considerable part of the memory used by the grid to store otherwise required* faces belonging to the cell *part of the connectivity description. The key idea is to use a uniform way for creating the structure of the coefficient matrix from the grid connectivity description and to access that data when computing the elements of the coefficient matrix. This saving requires many modifications to the computational algorithm details, which are addressed. Computational method features include a SIMPLE-based pressure-velocity coupling and co-located variable arrangement in which all flow variables are stored at cell centers, and mass fluxes are stored on face centers. Also handling convective and diffusive fluxes is described. The throughput is benchmark validated and shows second order truncation properties, both in time and space.*

Keywords: *Arbitrary polyhedral; Unstructured; Unsteady; Incompressible; Co-located.*

INTRODUCTION

Numerical algorithms for flow simulation in complex geometries have evolved from Cartesian grid solvers to multi block [1-3], hybrid [4,5] and current state-of-the-art polyhedral grid solvers [6-12]. Solvers with polyhedral grid capability are appealing in complex geometries. Arazgaldi [13] presents a cavitating propeller modeled using polyhedral grids. Polyhedral grids offer higher degrees of flexibility, not only in mesh generation, but also in grid adaption, grid fusion (in multi grid) and overset grid applications. There are a variety of unstructured grid finite volume strategies. The first choice to make is between cell-centered [8,11,14-17] or vertex-centered schemes [14,18-20]. This work is just concerned about the cell-centered methods. The variable arrangement is also a matter of consideration; without going into the details, we use the widely used co-located variable arrangement.

Another important thing about the unstructured grid is the way its connectivity is defined. There is no counterpart for connectivity in structured grids because it is implied by indexing. Unfortunately, connectivity requires the major part of the memory consumed for unstructured grid handling. In FEM applications it is preferred to use element-based or edge-based connectivity (see for example [21]). The element-based (cell-based) connectivity describes the connectivity as a *nodes belonging to cell* relationship. For FVM, however, because of the need to compute fluxes over the faces of each cell, face-based connectivity is preferred. Face-based connectivity also allows for uniform implementation of the arbitrary polyhedral grids, because it does not need any special treatment for different cell topologies. The face-based connectivity has two parts: *nodes belonging to a face* and *faces belonging to a cell (F2C)*. A modified version of the face-based connectivity adds a Left and Right Cell (LRC) relationship for each face. This inclusion adds to the storage cost but removes the search overhead for interpolation to face centers. The connectivity description method chosen directly affects the way elements of the coefficient matrices are evaluated.

There is extensive literature concerning discretization of the terms (like convection term) in the governing integral equations, their accuracy and their

1. *Center of Excellence in Energy Conversion, School of Mechanical Engineering, Sharif University of Technology, Tehran, P.O. Box 11155-9567, Iran.*
*. *Corresponding author. E-mail: mssaidi@sharif.edu*

Received 5 January 2010; received in revised form 11 July 2010; accepted 16 August 2010

stability. At the same time, efficient use of the computational hardware was always of concern. Nowadays, with lower hardware costs, simulation of finer details and more complex physics are possible. Even coupled solvers with much larger coefficient matrices than segregated ones are becoming attractive [22]. The trend is to consume all resources available to capture more details and handle more realistic cases. This encourages continuous development of more efficient computational methodologies.

In this work, we first describe one methodology for discretizing incompressible flow equations on polyhedral grids (a technology frontier by itself). Then, we show that there is a potential for memory saving by changing the strategy of computing coefficient matrix elements. The proposed modification removes the need for *F2C*, which is a heavy memory burden in the face-based connectivity description. Assuming that there are N_{cells} cells in the domain and, on average, N_f faces per cell, the memory consumed by *F2C* is equal to $N_{\text{cells}} \times N_f$ integer values. If, on average, there are N_v vertices per face and a total of N_{faces} in the domain, the total memory occupied by the connectivity is $N_{\text{faces}} \times N_v + N_{\text{cells}} \times N_f + 2 \times N_{\text{faces}}$ integers (the last term describes the *LRC* requirement). Table 1 shows the memory share of the *F2C* relative to the total memory required by the connectivity and grid (node coordinates plus connectivity) for different unstructured grids filling two- and three-dimensional unit cavities with cell edge sizes equal to 0.05. The share is always larger than 22% and the saving potential is considerable.

The major contribution of this work is to propose a method to combine features of the face-based connectivity and structure of the sparse coefficient matrix for evaluating coefficient matrix elements without imposing search or storage overheads and without requiring the *F2C* part of the connectivity. To completely remove the need for storing *F2C*, many details like the way the gradient or geometric entities are computed need reconsideration, which is also addressed.

In what follows, first the details of a polyhedral grid discretization is given. This shows why *F2C* is required in common practice. Then, by defining cell-based and face-based jobs, it is shown that the need for *F2C* can be removed. In the next section, geometric entities and elements of the coefficient matrix are computed without invoking *F2C* data. Finally, some benchmark problems are solved to test the integrity of the methodology and validate its correct truncation error behavior.

POLYHEDRAL GRID DISCRETIZATION OF THE GOVERNING EQUATIONS

For the incompressible flow, the continuity and momentum equations are:

$$\int_\Omega \rho \vec{V}.d\vec{S} = 0, \tag{1}$$

$$\frac{\partial}{\partial t}\int_\Omega \rho\phi dV + \int_\Gamma \phi(\rho\vec{V}.d\vec{S}) = S_p + \int_\Gamma D_\phi \frac{\partial\phi}{\partial n}dS + \int_\Omega q_{\phi_V} dV, \tag{2}$$

where ϕ could be any transported scalar including velocity components (u, v, w), q_{ϕ_V} is the volumetric source term and S_p represents the pressure term, which is only applicable to the momentum equations as:

$$\vec{S}_p = -\int_\Omega \vec{\nabla} p dV = -\int_\Gamma p d\vec{S}. \tag{3}$$

Sign and Naming Conventions

Before going into the details of discretization, it is convenient to put forward some conventions. There are two cases where the face direction is required, with regards to a cell and independent. With regards to a cell, j is used for the face number and the positive direction is always outwards from the cell. In independent addressing, the face vector, $\vec{S}_f$, is defined by the order of its vertices (nodes) and the right hand rule. In this case, f is used to address the face number. There are two cells sharing a face, P_{0f} and P_{1f}, chosen so that the face vector, $\vec{S}_f$, points from the left cell, P_{0f}, to the right cell, P_{1f}. Figure 1 illustrates these conventions.

Implicit Time Discretization

For integration in time, we follow the implicit second order three time levels method. The unsteady equa-

Table 1. Comparison of the memory share of *F2C* for different unstructured grids in unit square (cube in 3D). Grid edge size is set equal to 0.05 for all of the cases. Single (double) denotes single (double) precision.

***F2C* Share**	**Quad.**	**Tri.**	**Hexa (3D)**	**Tetra (3D)**	**Wedge (3D)**
Connectivity	32%	33%	30%	28%	26%
Grid (Single)	27%	29%	26%	27%	24%
Grid (Double)	24%	26%	24%	26%	22%

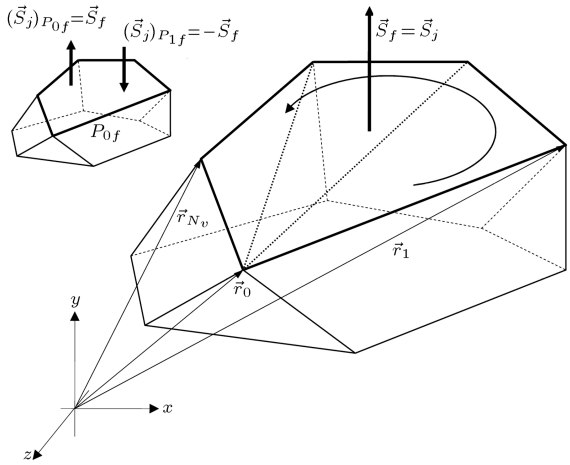


Figure 1. Polyhedral cell terminology. Also shown is the division of the highlighted face to triangles for geometric computations.

tions are symbolically rewritten as:

$$\frac{dy}{dt} = H, \tag{4}$$

where y represents the volume integral in Equation 2 and H contains all other terms. Discretization gives:

$$\frac{3y^n - 4y^{n-1} + y^{n-2}}{2\Delta t} = \left(\frac{dy}{dt}\right)^n + O(\Delta t^2)$$
$$= H^n + O(\Delta t^2). \tag{5}$$

Applying the above time discretization, Equation 2 could be discretized in space for a polyhedral cell, P_0, having N_j faces to:

$$a_{P_0}\phi_{P_0} + \sum_j^{N_j} a_{P_j}\phi_{P_j} = b_{P_0}, \tag{6}$$

in which a_{P_0}, a_{P_j} and b_{P_0} carry the effects of implicitly or explicitly discretized integrals.

Volume Integrals

The volume integrals are discretized to second order using the mid-point rule:

$$\int_{\Delta V_{P_0}} \psi dV \approx \psi_{P_0}\Delta V_{P_0}. \tag{7}$$

Convection Terms and Mass Flux Computation

Convection surface integrals are discretized using the mid-point rule as:

$$\int_j \phi(\rho\vec{V}.d\vec{S}) \approx \dot{m}_j\phi_j, \tag{8}$$

where ϕ_j represents the face center value and mass flux is assumed positive out of the cell. To avoid pressure checker-boarding, the mass flux in Equation 8 is obtained from the standard Rhie-Chow [23] (momentum-based) interpolation between the cells which have face f in common as:

$$u_{n_f} = \frac{\dot{m}_f}{\rho S_f} = \overline{(\vec{u})}_f.\hat{n}_f$$
$$+ \overline{\left(\frac{\Delta V}{a_0}\frac{\delta P}{\delta n_f}\right)}_f - \overline{\left(\frac{\Delta V}{a_0}\right)}_f \left(\frac{\delta P}{\delta n_f}\right)_f, \tag{9}$$

where $\overline{(.)}_f$ means any interpolation to the face center, like CDS, a_0 is the main diagonal element in the discretized momentum equation of the corresponding cell, and $\delta(.)/\delta n_f$ is the discretized face normal derivative operator.

To find an approximation to ϕ_j for Equation 8, any upwind scheme could be used. To promote the numerical stability of the solver, a deferred correction strategy [24] is used in this work. In this strategy, the standard First Order Upwind (FOU) is used for computing the coefficient matrix, which gives it good iteration properties. To promote accuracy to second order, the difference between FOU and the standard Second Order Upwind method (SOU) is added to the right hand side (i.e., lagged one iteration). The converged solution corresponds to SOU, while the coefficient matrix behaves as good as FOU.

For FOU, the upwind cell must be identified first. The upwind cell, P_U, is determined using the direction of the mass flux relative to face normal vector ($\vec{S}_f$). The value of ϕ_{P_U} is assumed to be convected to the face center ($\phi_j^{\text{FOU}} \approx \phi_{P_U}$). For SOU correction, the value of ϕ at the face center is interpolated using ϕ_{P_U} and $(\vec{\nabla}\phi)_{P_U}$. The face center value is approximated to second order as:

$$\phi_j^{\text{SOU}} \approx \phi_{P_U} + (\vec{\nabla}\phi)_{P_U}.(\vec{r}_j - \vec{r}_{P_U}). \tag{10}$$

Diffusion Terms

Diffusion surface integrals are discretized using:

$$\int_j D_\phi \frac{\partial\phi}{\partial n} dS \approx D_\phi \frac{\delta\phi}{\delta n_j} S_j. \tag{11}$$

To approximate the normal derivative, virtual points are used. The virtual point in cell P_{0f} related to face f is defined as the normal projection of the cell center on the face normal:

$$\vec{r}_{P'_{0f}} = \vec{r}_f + [\hat{n}_f.(\vec{r}_{P_{0f}} - \vec{r}_f)]\hat{n}_f, \tag{12}$$

where $\hat{n}_f$ is the unit normal vector of face f (Figure 2). The virtual point in cell P_{1f} is defined as the mirror image of the virtual point in P_{0f} with respect to

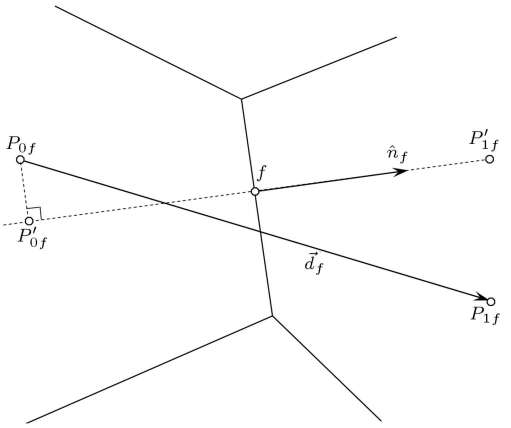


Figure 2. Definition of the virtual points.

the common face. The line connecting virtual points P'_{0f} and P'_{1f} is orthogonal to the face, and passes through the face center just at the mid-point of the line. Therefore, to second order:

$$\frac{\delta\phi}{\delta n_f} \approx \frac{\phi_{P'_{1f}} - \phi_{P'_{0f}}}{|\vec{r}_{P'_{1f}} - \vec{r}_{P'_{0f}}|} = \frac{\phi_{P'_{1f}} - \phi_{P'_{0f}}}{|\vec{d}'_f|}, \tag{13}$$

where values at virtual points are obtained using gradient-based interpolation, e.g.:

$$\phi_{P'_{0f}} \approx \phi_{P_{0f}} + (\vec{\nabla}\phi)_{P_{0f}}.(\vec{r}_{P'_{0f}} - \vec{r}_{P_{0f}}).$$

Pressure-Velocity Coupling

Continuity equation is used with SIMPLE algorithm to derive the pressure correction equation. For this algorithm, currently available velocity components and pressure are assumed to be predictions to the correct values, requiring a set of corrections which are related to each other as:

$$\vec{u}'_{P_0} \approx -\frac{\Delta V_{P_0}}{a_0}(\vec{\delta}P')_{P_0}, \tag{14}$$

where $\vec{\delta}$ is the discrete gradient operator. Obtaining the required mass flux correction from the velocity corrections, and substituting in the discretized continuity equation gives the pressure correction equation for cell P_0 as:

$$\sum_j^{N_j} \left(\rho \overline{\left(\frac{\Delta V}{a_0}\right)}_j (\vec{\delta}P')_j.\vec{S}_j \right) = \sum_j^{N_j} (\dot{m}_j), \tag{15}$$

where $(\vec{\delta}P')_j.\vec{S}_j$ is obtained using Equation 13 and:

$$(\vec{\delta}P')_j.\vec{S}_j = \frac{\delta P'}{\delta n_j} S_j. \tag{16}$$

This leads to a linear system of equations of the form:

$$c_{P_0} P'_{P_0} + \sum_j^{N_j} c_{P_j} P'_{P_j} = d_{P_0}. \tag{17}$$

Boundary Condition Application

To avoid special treatment for cells near boundaries, ghost cells are used outside the physical domain. Therefore, each boundary face, f, is surrounded by one real cell, P_{0f}, and one ghost cell, P_G. The ghost cell center is defined as the mirror image of the virtual point inside the real cell, P'_{0f}, with respect to the boundary face. With this choice, the face center value and the face normal derivative are easily computed to second order as:

$$\phi_f \approx \frac{\phi_{P_G} + \phi_{P'_{0f}}}{2}, \tag{18}$$

$$\frac{\partial\phi}{\partial n_f} \approx \frac{\phi_{P_G} - \phi_{P'_{0f}}}{|\vec{r}_{P_G} - \vec{r}_{P'_{0f}}|}. \tag{19}$$

Boundary conditions are considered as the governing equations for the ghost cells.

Gradient Computation

The above outlined discretization assumes a gradient vector to be available. It is computed using the standard least square method as outlined in [18]. Since, near any cell center, $\vec{r}_{P_0}$, the value of ϕ can be approximated to second order as:

$$\phi(\vec{r}) \approx \phi_{P_0} + (\vec{\nabla}\phi)_{P_0}.(\vec{r} - \vec{r}_{P_0}), \tag{20}$$

neighbor cell center values, at $\vec{r}_{P_j}$, can be obtained from:

$$\begin{aligned}\phi_{P_j} &= \phi_{P_0} + (\vec{\nabla}\phi)_{P_0}.(\vec{r}_{P_j} - \vec{r}_{P_0}) \\ &= \phi_{P_0} + (\vec{\nabla}\phi)_{P_0}.\vec{d}_j.\end{aligned} \tag{21}$$

Rearranging for the gradient gives:

$$(\vec{\nabla}\phi)_{P_0}.\vec{d}_j = \phi_{P_j} - \phi_{P_0}. \tag{22}$$

This is always an over determined system of equations for gradient components, since in three (two) dimensions, there are at least four (three) neighbors for each cell but three (two) components of the gradient are to be solved for. Following the standard least square procedure, the gradient components could be obtained as:

$$(\vec{\nabla}\phi)_{P_0} = D^{-1} \sum_j d_j^T (\phi_{P_j} - \phi_{P_0}), \tag{23}$$

where, j runs over all of the neighbors of P_0, d_j^T is the single column notation for $\vec{d}_j$ and D is a symmetric three by three (two by two in 2D) geometry-dependent matrix defined as:

$$D = \sum_j d_j^T d_j = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix}. \tag{24}$$

The inverse of D is computed analytically as:

$$D^{-1} = \frac{1}{adf + 2bce - ae^2 - dc^2 - fb^2} \begin{bmatrix} df - e^2 & ce - bf & be - cd \\ ce - bf & af - c^2 & bc - ae \\ be - cd & bc - ae & ad - b^2 \end{bmatrix}. \quad (25)$$

Having geometry dependent D available, the gradient vector could be obtained from Equation 23. Since the computation of the gradient requires the knowledge of the neighbor values of ϕ, whenever the gradient is encountered in the discretization, it is lagged. This means that the values from the last iteration are used for the computation of the gradient. At convergence, this lagging will not affect the solution, but it may slow down the convergence.

Coefficient Matrices and Their Storage

Following the routines outlined above, a linear system of equations of the form of Equation 6 is obtained for each of the velocity components. The coefficients of the system are given in Table 2. It is evident that the coefficient matrix is sparse with compact support (stencil extended just to immediate neighbors). The pressure correction discretized equation has the same structure with coefficients given in the same table.

For the finite volume method described above, non-zero elements are related to the discretization of the volume and surface integrals. Volume integrals affect the main diagonal elements, while surface integrals, because of the required interpolations, affect both main and off-diagonal elements. Every row in the coefficient matrix (row i) corresponds to the discretized transport equation for cell number i. Therefore, non-zero elements on the ith row are just on the main diagonal or at the column numbers corresponding to the cell number of the neighbors of the ith cell.

Sparse coefficient matrices can be economically handled with a common Compressed Row Storage format (CRS). In CRS, non-zero members of the coefficient matrix are sequentially stored in a one-dimensional array, a, swapping the matrix row by row. The column number of every non-zero element encountered is also stored in another one-dimensional array called $\text{col}_{\text{index}}$. For every row, a pointer to the beginning element of the row in a is stored in another one-dimensional vector called row_{ptr}. This means that $\text{row}_{\text{ptr}}[i+1] - \text{row}_{\text{ptr}}[i]$ is the number of non-zeros on the ith row of the original matrix. Non-zero values corresponding to that row are stored in $a[\text{row}_{\text{ptr}}[i]]$ to $a[\text{row}_{\text{ptr}}[i+1]-1]$. In the data chunk related to a row, the order is not important. For convenient access to the main diagonal element, we use the first place of the data chunk to store the main diagonal element.

Usually, row_{ptr} is extended by a single element, which stores the total number of non-zeros for the sake of uniformity of application. The interdependence of these three one-dimensional arrays and the grid is illustrated in Figure 3. It is worth noting that since non-zeros correspond to the neighbors, the structure of the coefficient matrix could be obtained from the grid

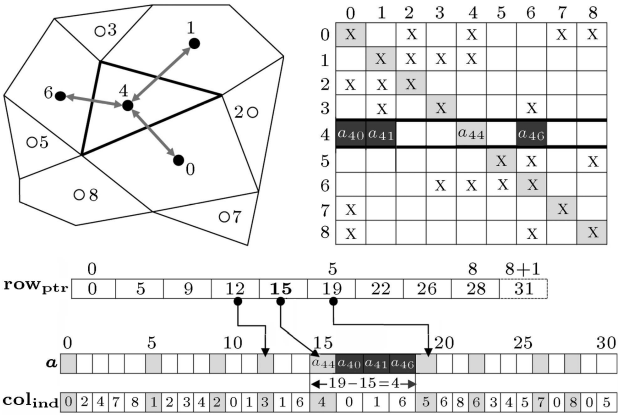


Figure 3. The structure of the sparse coefficient matrix and its relation to the grid topology.

Table 2. The elements of the coefficient matrix resulted from the implicit discretization of the generic scalar transport Equation 6 and pressure correction Equation 17 on cell P_0.

a_{P_0}	$\frac{3(\rho\Delta V)_{P_0}}{2\Delta t} + \sum_j^{N_j} \left(\max(\dot{m}_j, 0) + D_{\phi_j} \frac{\lvert\vec{S}_j\rvert}{\lvert\vec{d}'_j\rvert} \right)$
a_{P_j}	$\min(\dot{m}_j, 0) - D_{\phi_j} \frac{\lvert\vec{S}_j\rvert}{\lvert\vec{d}'_j\rvert}$
b_{P_0}	$\frac{4(\rho\Delta V)_{P_0}(\phi)_{P_0}^{n-1} - (\rho\Delta V)_{P_0}(\phi)_{P_0}^{n-2}}{2\Delta t} + q_{\phi P_0}\Delta V_{P_0}$ $+ \sum_j^{N_j} \left(D_{\phi_j} \frac{\lvert\vec{S}_j\rvert}{\lvert\vec{d}'_j\rvert} \left((\vec{\nabla}\phi)_{P_j} . (\vec{r}_{P'_{P_j}} - \vec{r}_{P_j}) - (\vec{\nabla}\phi)_{P_0} . (\vec{r}_{P'_{P_0}} - \vec{r}_{P_{P_0}}) \right) \right)^{\text{old}}$ $+ \sum_j^{N_j} \left(\max(\dot{m}_j, 0)\phi_{P_0} + \min(\dot{m}_j, 0)\phi_{P_j} - (\dot{m}_j\phi_j)^{\text{SOU}} \right)^{\text{old}}$
c_{P_j}	$\left(\frac{\rho\Delta V}{a_0} \right)_j \frac{\lvert\vec{S}_j\rvert}{\lvert\vec{d}'_j\rvert}$
c_{P_0}	$-\sum_j^{N_j} c_{P_j}$
d_{P_0}	$\sum_j^{N_j} \dot{m}_j$

information without constructing the two-dimensional matrix itself (and paying for the associated high memory costs).

Methods to Compute Coefficient Matrix Elements

One easy way to compute the transport coefficient matrix is to loop over all of the cells. For every cell (corresponding row in the coefficient matrix), computation of the main and off diagonal elements requires at least the knowledge of the volume of the cell, area of its faces and cell number of its neighbors. The volume of the cell is readily available because the cell number coincides with the row number of the coefficient matrix. The area of the faces requires the cell to know its faces. This information is taken care of using the *face belonging to cell (F2C)* part of the connectivity. Neighbors of the cell are stored as *Left and Right Cell (LRC)* in the face-based connectivity. Therefore, if the cell knows its faces and if every face knows its adjacent cells, neighbor information is available to the cell.

Of course, although the above methodology provides sufficient means to evaluate the elements of the coefficient matrix, it is by no means the most efficient way. It requires double computation of the face fluxes (once per each cell sharing the face). It also requires the storage of the bulky *F2C* in the grid connectivity. Another problem with the method is finding the element on row number i with column number j in a sparse matrix. Fortunately, the order of storage of faces in *F2C* for each cell could be exploited to remove this search overhead. This order is not available in discretization without *F2C*. This work describes a way to circumvent this search problem.

To avoid double computations (per face), one can first loop over all of the faces of the domain and compute the implicit and explicit fluxes per face. The values cannot be directly inserted into the coefficient matrix unless a search is undertaken to find column number j on row number i in the sparse system. This time the order in *F2C* could not be exploited because looping over the faces of the domain makes it impossible to know the sequential place of the face number in the ordered *F2C* list of the corresponding cells, unless a search is carried out. Now that the direct insertion requires a search overhead, one may store the computed fluxes per face. This requires a storage space proportional to the number of faces. By looping over the cells, these values could be retrieved (instead of computed) and used to fill the coefficient matrix without searching (exploiting the order in *F2C*).

To summarize, the three methods presented above require either added computational cost (double computations per face or search overhead to locate the element in the spare matrix) or memory overhead (flux storage per face and *F2C*).

CELL-BASED AND FACE-BASED JOBS

A Cell-Based (CB) job is defined as a job which requires a loop over the cells in the computational domain. The first process of evaluating the coefficient matrix defined previously is an example of a CB job. Similarly, a Face-Based (FB) job is defined as a job which requires a loop over the faces in the computational domain.

As described previously, computing elements of the coefficient matrix in a row by row (CB) manner not only requires *F2C*, but also demands that the fluxes over the faces (surface integrals) be computed twice (once for each cell sharing the face). To remove this double computation overhead, one can first define a FB job in which all of the fluxes are evaluated and stored per face. Then using a same CB job as before, the elements of the coefficient matrix are evaluated. This time, since the fluxes are available per face, they are just retrieved. This although removes the double computation overhead, requires a storage space equal to the number of faces for the fluxes. To summarize, the aforementioned methodology is carried out with following steps:

1. Loop over all of the faces in the domain. For each face:
   (a) Use *LRC* to interpolate to the face center and compute fluxes.
   (b) Store the flux for the face.
2. Loop over all of the cells of the domain. For each cell:
   (a) Compute the effect of the volume integrals and update the main diagonal element.
   (b) Use *F2C* to find its faces. Loop over the faces of the cell. For each face:
      i. Retrieve the value of fluxes stored for the face.
      ii. Insert the main diagonal element share from the face.
      iii. Insert the off-diagonal element share from the face by retrieving the related neighbor cell number from *LRC*.

It is worth noting that the right hand side of each equation is treated in the same way as the main diagonal element.

DISCRETIZATION WITHOUT *F2C*

The other way, proposed in this work, is to evaluate and accumulate all of the fluxes in the FB job. This not only removes the need for the flux storage, but also removes the need for *F2C*. The task is carried out as:

1. Loop over all of the faces in the domain. For each face:
   (a) Use *LRC* to find neighboring cells and interpolate to find face fluxes.
   (b) Directly insert the share of the face flux to the main and off diagonal elements of the coefficient matrix.
2. Loop over all of the cells of the domain and insert the effect of volume integrals on the main diagonal elements.

The key step is Step 1b. Directly inserting the share removes the need for storage of the fluxes for later use. It also removes the need for storage of *F2C*, since the CB job of Step 2 has nothing to do with faces of the corresponding cell and, therefore, does not require *F2C*. For inserting the shares in Step 1b, one needs to know the insertion location in the coefficient matrix stored in CRS format (the same idea applies to other sparse storage mechanisms).

For face number f, there are two neighbor cells, namely, P_{0f} and P_{1f}. The flux over this face contributes to the main diagonal coefficient of the equations at row numbers P_{0f} and P_{1f}. It also contributes to the element located at column number P_{1f} on row number P_{0f}, and the element located at column number P_{0f} on row number P_{1f}. For full matrix storage, access to these elements is trivial. Of course, this is not the case when sparse storage is exploited (which is the economical way of handling unstructured-grid real-word problems).

The main diagonal element on row number i is stored in $a[\mathrm{row}_{\mathrm{ptr}}[i]]$ and is easily accessible. The off-diagonal element on row number i and column number j can be found as follows. The easiest, but not the most efficient, way is to search over the data chunk stored in $\mathrm{col}_{\mathrm{ind}}[\mathrm{row}_{\mathrm{ptr}}[i]+1]$ to $\mathrm{col}_{\mathrm{ind}}[\mathrm{row}_{\mathrm{ptr}}[i+1]-1]$ for j. The number of data stored there is equal to the number of faces cell number i has. Therefore, this is a low cost search for each cell. But it should be carried out many times for the whole grid, which imposes a search overhead. This overhead can be avoided as proposed below.

The key idea is that the order of cell numbers (column numbers in $\mathrm{col}_{\mathrm{ind}}$) depends on the way the structure of the coefficient matrix is created from the grid connectivity description. If the same procedure is used for accessing that data, there is no need for a search. The structure of the coefficient matrix is to be generated from a face-based grid description. Since the connectivity is to be described as *node belonging to face* and *Left and Right Cells (LRC)* of the face, one-dimensional arrays, a, $\mathrm{col}_{\mathrm{ind}}$ and $\mathrm{row}_{\mathrm{ptr}}$, are to be formed just using *LRC*.

The first step is to reserve memory for them. The length of $\mathrm{row}_{\mathrm{ptr}}$ is equal to the number of rows of the coefficient matrix (equal to the number of cells) plus one: a and $\mathrm{col}_{\mathrm{ind}}$ have equal lengths. Since they hold non-zero elements, their lengths are equal to $N_{\mathrm{cells}} + 2N_{\mathrm{faces}}$. N_{cells} accounts for the main diagonal elements and $2N_{\mathrm{faces}}$ represents the total number of off-diagonal non-zero elements. Off-diagonal elements (cross-links) are the result of the inter-connection of the equations. For face number f, there is an off-diagonal element on row number P_{0f} and column number P_{1f}, and another off-diagonal element on row number P_{1f} and column number P_{0f}. Therefore, there are $2N_{\mathrm{faces}}$ off-diagonal non-zero elements.

The next step is to fill $\mathrm{row}_{\mathrm{ptr}}$ and $\mathrm{col}_{\mathrm{ind}}$, which together define the structure of the sparse coefficient matrix. For $\mathrm{row}_{\mathrm{ptr}}$, since it stores the offset of the first element of each row from the first non-zero in the coefficient matrix, we first need to count the non-zeros on each row and then accumulate them:

1. Put one in every element of $\mathrm{row}_{\mathrm{ptr}}$ accounting for main diagonal elements.
2. Count off-diagonal elements: Loop over the faces of the domain. For every face, f, use *LRC* to find neighboring cells (P_{0f} and P_{1f}) and increment the values stored in $\mathrm{row}_{\mathrm{ptr}}[P_{0f}]$ and $\mathrm{row}_{\mathrm{ptr}}[P_{1f}]$. The reason is that every face means a cross-link between its left and right cells and this adds to the number of elements on the corresponding rows. After this loop, every $\mathrm{row}_{\mathrm{ptr}}[i]$ contains the number of non-zeros on the ith row of the coefficient matrix.
3. Accumulate these values towards the end of the $\mathrm{row}_{\mathrm{ptr}}$ in a loop over the elements of $\mathrm{row}_{\mathrm{ptr}}$ in order to find the required offset. This is done in a loop from the second element of $\mathrm{row}_{\mathrm{ptr}}$ onwards; for every element, i, setting $\mathrm{row}_{\mathrm{ptr}}[i]$ to $\mathrm{row}_{\mathrm{ptr}}[i] + \mathrm{row}_{\mathrm{ptr}}[i-1]$.
4. Shift all elements of $\mathrm{row}_{\mathrm{ptr}}$ towards the end by a loop from the last element to the second one, and, for each element i replace the current value by $\mathrm{row}_{\mathrm{ptr}}[i-1]$. For the first element, zero should be used.

Now every element in $\mathrm{row}_{\mathrm{ptr}}$ contains the number of non-zeros before the corresponding row in the coefficient matrix.

To complete $\mathrm{col}_{\mathrm{index}}$, we make use of an auxiliary, one-dimensional integer array with N_{cellS} elements called *front*. Since *front* and pressure correction (P') are never required simultaneously, and since the memory for P' must be reserved and is greater than or equal to the memory required by *front*, that space could be used which removes any memory overhead. The aim of using the front is to keep the track of the filled elements in $\mathrm{col}_{\mathrm{ind}}$ for each row:

1. Main diagonal elements: In a CB job (with index i), put i in $\mathrm{col}_{\mathrm{ind}}[\mathrm{row}_{\mathrm{ptr}}[i]]$ which means that the first

element of the data chunk related to the row number i is the diagonal element with column number i. At the same time put one in the corresponding elements of $front$, which means that for every element, one column index is set (the main diagonal one). After this step, all off the elements of $front$ contain one.

2. Off-diagonal column numbers: Loop over the faces of the domain. For every face f use LRC to find neighboring cells (P_{0f} and P_{1f}). Set P_{1f} in $\mathrm{col_{ind}}[\mathrm{row_{ptr}}[P_{0f}] + \mathrm{front}[P_{0f}]]$ and increment $\mathrm{front}[P_{0f}]$ (advance the front for P_{0f}). Also set P_{0f} in $\mathrm{col_{ind}}[\mathrm{row_{ptr}}[P_{1f}] + \mathrm{front}[P_{1f}]]$ and increment $\mathrm{front}[P_{1f}]$ (advance the front for P_{1f}).

Mimicking Step 2 when computing elements of the coefficient matrix makes it possible to insert the off-diagonal shares of the elements of the coefficient matrix without search overhead.

It is now appropriate to separate the CB and FB shares of the elements of the coefficient matrix. Table 3 shows this splitting. FB and CB superscripts stand for the face-based and cell-based shares, respectively. Also it is worth noting that symbols like $(a_{P_{0f}})_{P_{1f}}$ mean the coefficient of $\phi_{P_{0f}}$ in the equation related to its neighbor with cell number P_{1f} and therefore the element on row number P_{1f} and column number P_{0f} of the coefficient matrix.

Now elements of the coefficient matrix are evaluated with:

1. Make all elements of the coefficient matrix (elements of a) zero.
2. Loop over all cells and add cell-based shares to the main diagonal and right hand side elements. For each cell put one into the corresponding element of the $front$.
3. Loop over all faces of the domain, for each face, f, retrieve P_{0f} and P_{1f} cells from LRC and add the contribution of the face to $(a_{P_{0f}})_{P_{1f}}$, $(a_{P_{1f}})_{P_{0f}}$, $(a_{P_{0f}})_{P_{0f}}$, $(a_{P_{1f}})_{P_{1f}}$, $b_{P_{0f}}$ and $b_{P_{1f}}$. After adding the shares, advance the front for P_{0f} and P_{1f} (i.e., increment the values stored in $front[P_{0f}]$ and $front[P_{1f}]$).

Step 3 requires the knowledge of the position of the elements in a. Element $(a_{P_{0f}})_{P_{1f}}$ is located at $a[\mathrm{row_{ptr}}[P_{1f}] + \mathrm{front}[P_{1f}]]$. Likewise, $(a_{P_{1f}})_{P_{0f}}$ is located at $a[\mathrm{row_{ptr}}[P_{0f}] + \mathrm{front}[P_{0f}]]$. Main diagonal element $(a_{P_{0f}})_{P_{0f}}$ is located at $a[\mathrm{row_{ptr}}[P_{0f}]]$ and $(a_{P_{1f}})_{P_{1f}}$ is located at $a[\mathrm{row_{ptr}}[P_{1f}]]$. The same method could be applied to the pressure correction

Table 3. Face-based and cell-based shares of the elements of the coefficient matrix. FB shares are from face number f.

$\left(a^{FB}_{P_{0f}}\right)_{P_{0f}}$	$\max(\dot{m}_f, 0) + D_{\phi_j}\frac{\lvert\vec{S}_f\rvert}{\lvert\vec{d}'_f\rvert}$
$\left(a^{FB}_{P_{0f}}\right)_{P_{1f}}$	$-\left(a^{FB}_{P_{0f}}\right)_{P_{0f}}$
$\left(a^{FB}_{P_{1f}}\right)_{P_{1f}}$	$-\min(\dot{m}_f, 0) + D_{\phi_j}\frac{\lvert\vec{S}_f\rvert}{\lvert\vec{d}'_f\rvert}$
$\left(a^{FB}_{P_{1f}}\right)_{P_{0f}}$	$-\left(a^{FB}_{P_{1f}}\right)_{P_{1f}}$
$\left(b^{FB}_{P_{0f}}\right)_{P_{0f}}$	$D_{\phi_j}\frac{\lvert\vec{S}_f\rvert}{\lvert\vec{d}'_f\rvert}\left((\vec{\nabla}\phi)_{P_{1f}}.(\vec{r}_{P'_{P_{1f}}} - \vec{r}_{P_{1f}}) - (\vec{\nabla}\phi)_{P_{0f}}.(\vec{r}_{P'_{P_{0f}}} - \vec{r}_{P_{P_{0f}}})\right)^{\mathrm{old}}$
	$+\left(\max(\dot{m}_f, 0)\phi_{P_{0f}} + \min(\dot{m}_f, 0)\phi_{P_{1f}} - (\dot{m}_j\phi_j)^{\mathrm{SOU}}_{P_{0f}}\right)^{\mathrm{old}} + q_{\phi_{P_0}}\Delta V_{P_0}$
$\left(b^{FB}_{P_{1f}}\right)_{P_{1f}}$	$-\left(b^{FB}_{P_{0f}}\right)_{P_{0f}}$
$\left(a^{CB}_{P_0}\right)_{P_0}$	$\frac{3(\rho\Delta V)_{P_0}}{2\Delta t}$
$\left(b^{CB}_{P_0}\right)_{P_0}$	$\frac{4(\rho\Delta V)_{P_0}(\phi)^{n-1}_{P_0} - (\rho\Delta V)_{P_0}(\phi)^{n-2}_{P_0}}{2\Delta t}$
$\left(c^{FB}_{P_{1f}}\right)_{P_{0f}}$	$\left(\frac{\rho\Delta V}{a_0}\right)_j\frac{\lvert\vec{S}_f\rvert}{\lvert\vec{d}'_f\rvert}$
$\left(c^{FB}_{P_{0f}}\right)_{P_{1f}}$	$\left(c^{FB}_{P_{1f}}\right)_{P_{0f}}$
$\left(c^{FB}_{P_{0f}}\right)_{P_{0f}}$	$-\left(c^{FB}_{P_{1f}}\right)_{P_{0f}}$
$\left(c^{FB}_{P_{1f}}\right)_{P_{1f}}$	$-\left(c^{FB}_{P_{1f}}\right)_{P_{0f}}$
$\left(d^{FB}_{P_{0f}}\right)_{P_{0f}}$	$\dot{m}_f$
$\left(d^{FB}_{P_{1f}}\right)_{P_{1f}}$	$-\left(d^{FB}_{P_{0f}}\right)_{P_{0f}}$

equation because the structure of the coefficient matrix is the same.

Required Modifications When F2C Is Not Available

Computing geometric entities like cell volume needs reconsideration when *F2C* is not available. Since the cell does not know its faces, the volume could not be obtained in a CB job. Assuming that the face area and center are available (which require no modification regarding the removal of *F2C*) in CB oriented calculations, the cell volume could be obtained from Gauss's theorem as:

$$\Delta V_{P_0} = \int\limits_{\Delta V_{P_0}} dV = \frac{1}{3} \int\limits_{\Delta V_{P_0}} \vec{\nabla}.\vec{r} dV = \frac{1}{3} \int\limits_{S_{P_0}} \vec{r}.d\vec{S}$$

$$= \frac{1}{3} \sum_{j=1}^{N_j} \vec{r}_j.\vec{S}_j. \tag{26}$$

Or equivalently from:

$$\Delta V_{P_0} = \int\limits_{\Delta V_{P_0}} dV = \int\limits_{\Delta V_{P_0}} \vec{\nabla}.(x\hat{i}) dV = \int\limits_{S_{P_0}} (x\hat{i}).d\vec{S}$$

$$= \sum_{j=1}^{N_j} x_j(\hat{i}.\vec{S}_j) = \sum_{j=1}^{N_j} x_j (S_x)_j. \tag{27}$$

Cell volumes could be computed by accumulating FB shares while looping over all faces of the domain as:

$$\begin{cases} \Delta V_{P_{0f}}^{FB} = x_f (S_x)_f \\ \\ \Delta V_{P_{1f}}^{FB} = -x_f (S_x)_f = -\Delta V_{P_{0f}}^{FB} \end{cases} \tag{28}$$

Computing cell centers also requires modification. According to its definition, the x coordinate of the cell center could be computed from:

$$\bar{x}.\Delta V_{P_0} = \int\limits_{\Delta V_{P_0}} x dV = \int\limits_{\Delta V_{P_0}} \vec{\nabla}.(\frac{x^2}{2}\hat{i}) dV$$

$$= \frac{1}{2} \int\limits_{S_{P_0}} (x^2\hat{i}).d\vec{S} = \frac{1}{2} \sum_{j=1}^{N_j} (\hat{i}.\hat{n}_j) \int\limits_{S_j} x^2 dS. \tag{29}$$

For each face, the last integral could be computed [25] from the triangulation of the face (Figure 1). For each triangle, a vertex is chosen as vertex number zero ($\vec{r}_0$). The other two vertices are numbered in a right hand manner, so that their cross product is consistent with the face area vector direction. The relative location vector of these vertices is computed as $\vec{e}_i = \vec{r}_i - \vec{e}_0$. A surface parameterization of the triangle is assumed as $\vec{r}(p,q) = \vec{r}_0 + p\vec{e}_1 + q\vec{e}_2$ constrained to $0 \leqslant p,q \leqslant 1$ and $p+q \leqslant 1$. With this parameterization, the surface mapping relationship is:

$$dS = \left| \frac{\partial \vec{r}}{\partial p} \times \frac{\partial \vec{r}}{\partial q} \right| = |\vec{e}_1 \times \vec{e}_2| dpdq. \tag{30}$$

This results to:

$$(\hat{i}.\hat{n}_f) \int\limits_{S_f} x^2 ds = (\vec{e}_1 \times \vec{e}_2).\hat{i} \int\limits_0^1 \int\limits_0^{1-q} (x(p,q))^2 dpdq. \tag{31}$$

The integral could be evaluated analytically as:

$$\int\limits_0^1 \int\limits_0^{1-q} (x(p,q))^2 dpdq = \frac{1}{12}(x_0^2 + x_1^2 + x_2^2$$

$$+ x_0x_1 + x_0x_2 + x_1x_2). \tag{32}$$

The result of the integral when substituted in Equaiton 29 gives the cell center in terms of the nodal coordinates of its vertices. For a face-based application, the share of each cell from each face could be computed from:

$$\begin{cases} \bar{x}_{P_{0f}}^{FB} = \frac{1}{2\Delta V_{P_{0f}}} (\hat{i}.\hat{n}_f) \int_{S_f} x^2 dS \\ \\ \bar{x}_{P_{1f}}^{FB} = \frac{-1}{2\Delta V_{P_{1f}}} (\hat{i}.\hat{n}_f) \int_{S_f} x^2 dS \end{cases} \tag{33}$$

Therefore, cell centers could be found by looping over all faces of the domain; for each face, computing the integral in Equation 33 by substituting the coordinates of its nodes into Equation 32 and adding the shares to left and right cells according to Equation 33.

Computation of the D matrix used for evaluating the gradient and stored at cell centers also requires reconsideration. Since from Equation 24, D is equal to $\sum_j d_j^T d_j$, it could be computed in a FB job with face shares equal to:

$$D_{P_{0f}}^{FB} = D_{P_{1f}}^{FB} = d_f^T d_f. \tag{34}$$

After that D is available at cell centers, it can also be inverted using Equation 25 in a CB job.

To compute gradient field for ϕ from Equation 23 again a FB job followed by a CB job is employed. The share of each face in $\sum_j d_j^T(\phi_{P_j} - \phi_{P_0})$ is computed in a FB job and is accumulated on the memory reserved for gradient vector at cell centers by:

$$(d_j^T(\phi_{P_j} - \phi_{P_0}))_{P_{0f}}^{FB} = (d_j^T(\phi_{P_j} - \phi_{P_0}))_{P_{1f}}^{FB}$$

$$= d_f^T(\phi_{P_{1f}} - \phi_{P_{0f}}). \tag{35}$$

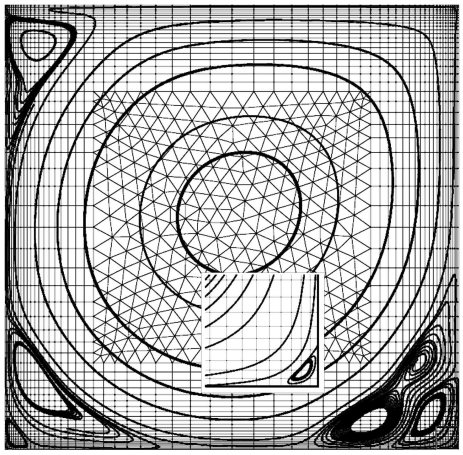
Now in a CB job, D^{-1} can be multiplied to $\sum_j d_j^T(\phi_{P_j} - \phi_{P_0})$ and stored at the same place for the gradient vector.

INTEGRITY AND ACCURACY CHECKS

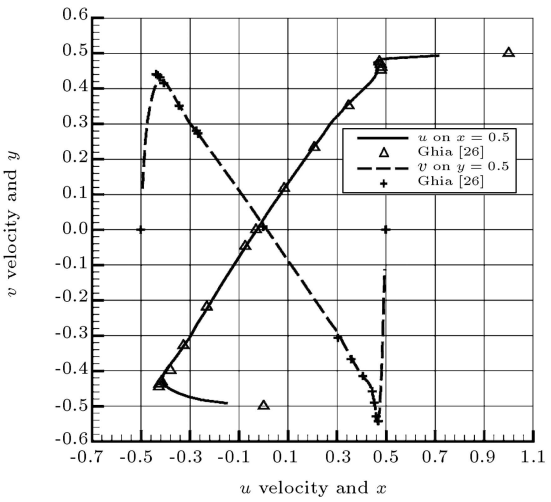
Using the methods outlined above, Rayan was developed with object oriented design. To verify the integrity and check its accuracy, some benchmark problems were solved, as reported hereafter.

Lid-Driven Cavity Problem

The lid-driven cavity problem is a first step standard test case for which a vast numerical and experimental data base exists. To verify polyhedral treatment, a grid was created with 3962 cells (7701 faces and 3740 vertices). It was a combination of quadrilateral (near wall) and triangular (center) cells (Figure 4). The flow was solved at Reynolds numbers ranging from 10^2 to 10^4. Also shown in Figure 4 are the streamlines corresponding to $\mathrm{Re} = 10^4$. Even on such a course mesh, tertiary and quaternary vortices are captured on the left-bottom and right-bottom corners, respectively. Horizontal and vertical components of velocity on vertical and horizontal symmetry lines are compared to the benchmark values given by Ghia et al. [26]. Regardless of the coarseness of the mesh, results compare well with benchmark data, which were obtained on a 257×257 grid (17 times more nodes) using a multi-grid stream function-vorticity method.



(a) Mixed type grid used for solving cavity problem. Also shown are the streamlines at $\mathrm{Re} = 10^4$. Close-up shows successful capturing of small quaternary corner vortex.



(b) Comparison of u and v profile on the central vertical and horizontal lines with benchmark solutions of Ghia [26].

Figure 4. Benchmark validation using cavity problem at $\mathrm{Re} = 10^4$.

Decaying Vortices Problem

The decaying vortices problem, having an analytical solution, is commonly used to measure the order of accuracy of the codes. Upon substitution, it could be easily shown that the flow field defied by:

$$\begin{cases} u(x,y,t) = -cos(\pi x)sin(\pi y)e^{-2\pi^2 t/\mathrm{Re}} \\ v(x,y,t) = sin(\pi x)cos(\pi y)e^{-2\pi^2 t/\mathrm{Re}} \\ p(x,y,t) = -\frac{1}{4}(cos(2\pi x) + cos(2\pi y))e^{-4\pi^2 t/\mathrm{Re}} \end{cases} \tag{36}$$

satisfies incompressible flow equations. This problem is solved for $\mathrm{Re} = 10$ and in the range $(x,y) \in \{\frac{-1}{2} < (x,y) < \frac{1}{2}\}$ (following [4]). Analytical values are used as the boundary and initial conditions. The errors are computed at $t = 0.3$ (the time at which the maximum velocity decays to almost half of its initial value). Various norms of error are defined as:

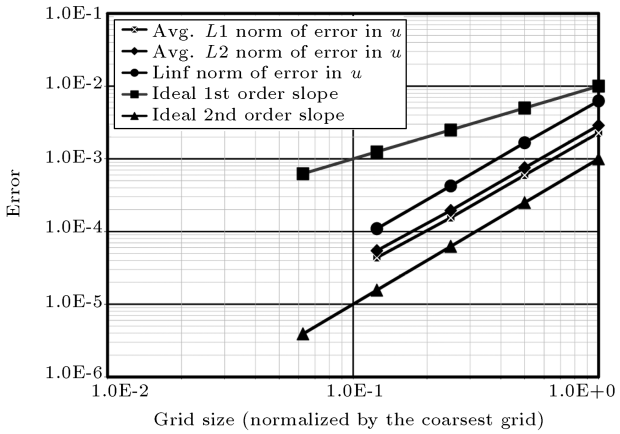
$$\varepsilon_{L1} = \frac{\sum |u_{\mathrm{computed}} - u_{\mathrm{analytic}}|}{N_{\mathrm{cells}}}, \tag{37}$$

$$\varepsilon_{L2} = \sqrt{\frac{\sum (u_{\mathrm{computed}} - u_{\mathrm{analytic}})^2}{N_{\mathrm{cells}}}}, \tag{38}$$

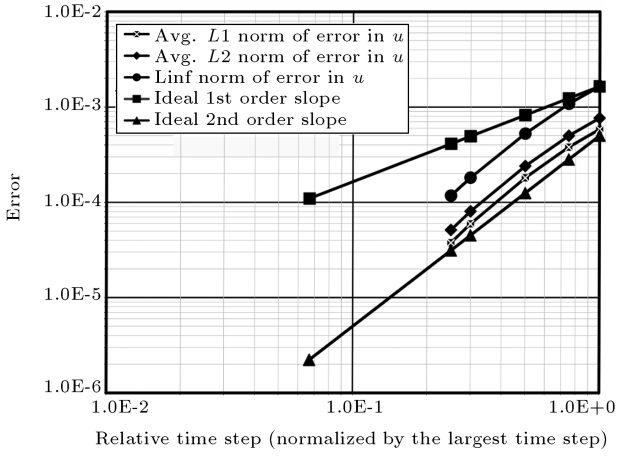
$$\varepsilon_{L\infty} = \max(u_{\mathrm{computed}} - u_{\mathrm{analytic}}). \tag{39}$$

To find the order of the space discretization truncation error, the flow is solved on systematically refined uniform grids of 10×10, 20×20, 40×40 and 80×80 cells. The time step is held constant at 0.001. Since the time step is small, almost all errors could be related to the space truncation error. Figure 5a proves that the error (described by three norms) is indeed of second order in space.

To check the truncation error in time, the problem is solved on a 113×113 uniform grid with time steps of 0.1, 0.075, 0.05, 0.03 and 0.025. Because the grid is very fine and time steps are so large, the error could be related to the time series truncation. The error is shown in Figure 5b, which proves the method is of second order in time, as well.



(a) Space truncation error



(b) Time truncation error

Figure 5. Error for decaying vortices problem with respect to the analytical solution at $t = 0.3$.

Flow Passing a Circular Cylinder

Flow normal to a cylinder possessing vortex shedding is solved at Reynolds numbers of 200 and 1000 on the grid shown in Figure 6. The mesh is a combination of clustered body conforming quad cells near the cylinder and triangular cells covering the rest of the domain. The related Strouhal number, defined as

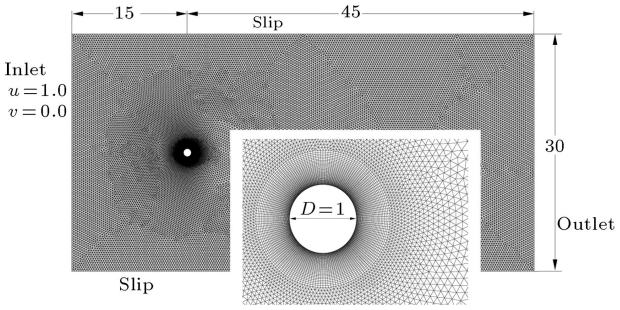


Figure 6. Domain and the mixed type grid used for the simulation of the vortex shedding problem over a cylinder.

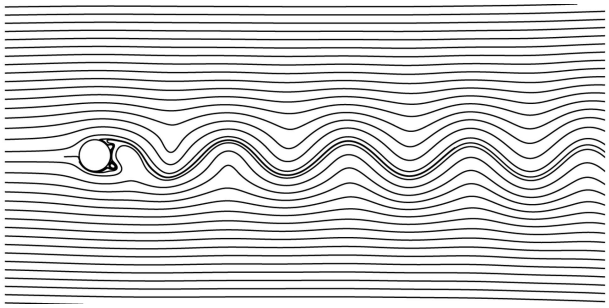


Figure 7. Streamlines for flow passing a circular cylinder at Re = 1000 and $t = 80$.

Table 4. Comparison of the St numbers for the vortex shedding from a circular cylinder.

	Manzari [27]	**He [28]**	**Present**
$\boldsymbol{St_{\mathrm{Re=200}}}$	0.20	0.1978	0.1980
$\boldsymbol{St_{\mathrm{Re=1000}}}$	0.238	0.2392	0.2381

$St = fD/V_\infty$, describes the frequency of the vortex shedding. Figure 7 show streamlines for $t = 80$ after sudden exposure to a flow with $\rho = 1$ and $\mu = 0.001$ flowing at $u = 1$. Table 4 summarizes the results for Re = 200 and 1000. As is evident, St numbers are in close agreement with Manzari [27] and He et al. [28] results, which guarantees the good behavior of the method in external flow problems.

CONCLUSION

A second order in time and space polyhedral grid finite volume methodology was described, which features no need for the *face belonging to cell (F2C)* part of the connectivity. Based on this methodology, a code called Rayan was developed and tested. The idea of creating the structure of the sparse coefficient matrix and computing its elements in a uniform way was addressed. It was shown that by using this idea, *F2C* is redundant. Changes necessary to geometric entities computation that were affected by removing *F2C* were addressed. The major achievement of the work was removing the need for *F2C* without imposing any other storage or search overheads. Not storing *F2C* results in at least 20% less memory requirement for grid connectivity storage. Compared to a single cell-based loop for computing the elements of the coefficient matrix, double flux computation per face overhead is removed. The proposed methodology also removes the need for storage of the fluxes per face, if elements of the coefficient matrix are to be computed in a face-based loop followed by a cell-based loop. Also, the search overhead, attributable to the element insertion mechanism, for both cases is removed. The accuracy of Rayan was proved through a series of classical benchmark validations.

ACKNOWLEDGMENT

The cooperation of Mr. M. Zendehbad in linking the PETSc solver to the Rayan code is gratefully acknowledged. The authors also appreciate the generous software backup and support of the open source community.

REFERENCES

1. Rembold, B. and Jenny, P. "A multiblock joint PDF finite-volume hybrid algorithm for the computation of turbulent flows in complex geometries", *J. of Comput. Physics*, **220**, pp. 59-87 (2006).
2. Thakur, S. and Wright, J. "A multiblock operator-splitting algorithm for unsteady flows at all speeds in complex geometries", *Int. J. for Num. Methods in Fluids*, **46**, pp. 383-414 (2004).
3. Banerjee, S.S. "The development of a multiblocked strongly conservative finite volume solver with chimera grid capabilities for flows in complex geometries", PhD dissertation, Texas A & M University (1999).
4. Kim, D. and Choi, H. "A second-order time-accurate finite volume method for unsteady incompressible flow on hybrid unstructured grids", *J. of Comput. Physics*, **162**(2), pp. 411-428 (2000).
5. Shaw, J.A., Georgala, J.M., Childs, P.N. "General procedures employed in the generation of three-dimensional hybrid structured/unstructured meshes", NASA STI/Recon Technical Report N 95, p 19506 (1994).
6. Basara, B. "Employment of the second-moment turbulence closure on arbitrary unstructured grids", *Int. J. for Num. Methods in Fluids*, **44**, pp. 377-407 (2004).
7. Smolarkiewicz, P.K. and Szmelter, J. "MPDATA: An edge-based unstructured-grid formulation", *J. of Comput. Physics*, **206**, pp. 624-649 (2005).
8. Hadzic, H. "Development and application of a finite volume method for the computation of flows around moving bodies on unstructured, overlapping grids", PhD Dissertation, Technischen Universitat Hamburg-Harburg (2005).
9. Peric, M. "Numerical methods for computing turbulent flows", in *Introduction to Turbulence Modeling V*, VKI Lecture Series 2004-06 (2004).
10. Wright, J.A. and Smith, R.W. "An edge-based method for the incompressible Navier-Stokes equations on polygonal meshes", *J. of Comput. Physics*, **169**(1), pp. 24-43 (2001).
11. Basara, B. "A pressure correction method for unstructured meshes with arbitrary control volumes", *Int. J. for Num. Methods in Fluids*, **22**, pp. 265-281 (1996).
12. Demirdzic, I. and Peric, M. "Finite volume method for prediction of fluid flow in arbitrarily shaped domains with moving boundaries", *Int. J. for Num. Methods in Fluids*, **10**, pp. 771-790 (1990).
13. Arazgaldi, R. and Hajilouy, A. and Farhanieh, B. "Experimental and numerical investigation of marine propeller cavitation", *Scientia Iranica, Trans. B, Mech. Eng.*, **16**(6), pp. 525-533 (2009).
14. Whitlow, D. "Finite volume methods for incompressible flow", PhD Dissertation, University of California, Davis (2001).
15. Eymard, R., Herbin, R. and Latché, J.-C. "Convergence analysis of a colocated finite volume scheme for the incompressible Navier-Stokes equations on general 2D or 3D meshes", *SIAM Journal on Numerical Analysis*, **45**(1), pp. 1-36 (2007).
16. Taylor, L.K. "Unsteady three-dimensional incompressible algorithm based on artificial compressibility", PhD Dissertation, Mississippi State Univ., State College. (1991).
17. Eberle, A. "Enhanced numerical inviscid and viscous fluxes for cell centered finite volume schemes", in *Computational Fluid Dynamics Symposium*, Wesseling, P., Segal, A., Vankan, J., Oosterlee, C.W. and Kassels, C.G.M., Eds., pp. 9-12 (1991).
18. Tai, C.H. and Zhao, Y. "A finite volume unstructured multigrid method for efficient computation of unsteady incompressible viscous flows", *Int. J. for Num. Methods in Fluids*, **46**, pp. 59-84 (2004).
19. Oosterlee, C.W. "A GMRES-based plane smoother in multigrid to solve 3D anisotropic fluid flow problems", *J. of Comput. Physics*, **130**, pp. 41-53 (1997).
20. Jessee, J.P. and Fiveland, W.A. "A cell vertex algorithm for the incompressible Navier-Stokes equations on non-orthogonal grids", *Int. J. for Num. Methods in Fluids*, **23**, pp. 271-293 (1996).
21. Manzari, M.T. "Inviscid compressible flow computations on 3D unstructured grids", *Scientia Iranica*, **12**(2), pp. 207-216 (2005).
22. Darwish, M., Sraj, I. and Moukalled, F. "A coupled finite volume solver for the solution of incompressible flows on unstructured grids", *J. of Comput. Physics*, **228**, pp. 180-201 (2009).
23. Rhie, C.M. and Chow, W.L. "Numerical study of the turbulent flow past an airfoil with trailing edge separation", *AIAA Journal*, **21**(11), pp. 1525-1532 (1983).
24. Khosla, P.K. and Rubin, S.G. "A diagonally dominant second-order accurate implicit scheme", *Computers & Fluids*, **2**(2), pp. 207-209 (1974).
25. Eberly, D. "Polyhedral mass properties (revisited)", Geometric Tools, LLC (2008).
26. Ghia, U. and Ghia, K.N. and Shin, C.T. "High-resolutions for incompressible flow using the Navier-Stokes equations and a multigrid method", *J. of Comput. Physics*, **48**(3), pp. 387-411 (1982).
27. Manzari, M.T. "A time-accurate finite element algorithm for incompressible flow problems", *Int. J. for Num. Methods for Heat & Fluid Flow*, **13**(2), pp. 158-177 (2003).

28. He, J.W., Glowinski, R., Metcalfe, R., Nordlander, A. and Periaux, J. "Active control and drag optimization for flow past a circular cylinder", *J. of Comput. Physics*, **163**, pp. 83-117 (2000).

BIOGRAPHIES

Mahdi Sani is a Ph.D. candidate at the Center of Excellence in Energy Conversion, School of Mechanical Engineering, Sharif University of Technology (SUT) and is also a faculty member of the SUT international campus on Kish island. He has served as a mechanical engineer in the Iran Marine Industrial Company (locally known as SADRA) and Monenco Iran. He has also served as a CFD consulting engineer in the Pargas Iran Company His research interests include: Dynamic Meshes, Fluid/Structure Interaction, Aerosol Dynamics and Living Tissue Modelling.

Mohammad Said Saidi is the professor of mechanical engineering at Sharif University of Technology. His research interests are: Modeling and Numerical Analysis of Transport and Deposition of Aerosol Particles, Modeling and Numerical Analysis of Biofluids, Modeling and Numerical Analysis of Thermal-Hydraulics of Porous Media and Microchannels.