PARALLEL COMPUTATION OF THE NAVIER-STOKES EQUATIONS USING IMPLICIT FINITE VOLUME METHOD

Masoud Darbandi* and Araz Banaeizadeh*

*Department of Aerospace Engineering
Sharif University of Technology, P.O. Box 11365-8639, Tehran, Iran
e-mail: darbandi@sharif.edu, web page: http://sina.sharif.edu/~darbandi/
araz@mehr.sharif.edu

Key words: Finite Volume, Finite Element, Parallel Computing, Implicit Method, Sparse Matrix Solver.

Abstract. The implicit methods are not restricted to small time step applications and provide superiority enough to choose them for solving steady-state CFD problems. However, a major drawback with the implicit methods is the need for solving the set of linear algebraic equations. In solving the big CFD problems, the matrix is very huge and the implicit workers may get frustrated in solving such huge matrices. The parallel computation of CFD problems is a major advancement in CFD which smoothes the path toward solving complex and huge problems. The essence of parallel computing is more consistent with the explicit methods than that of implicit methods. This is why the parallel computing research remarkably focuses on the explicit methods. However, in this research, our concern is on implicit solution of the incompressible Navier-Stokes equations. The extended method is a finite volume element method which utilizes the advantages of implicit algorithms. The resulting sparse matrices are solved in two manners of fully-coupled and semi-coupled. Different sparse solver algorithms are utilized to solve the flow problem using either fully-coupled or semi-coupled algorithms. The performance of the solvers are then investigated on a distributed computing environment. The study shows that a fully coupled algorithm still performs higher capabilities on parallel machines in comparison with the segregated algorithm. Further improvement in semi-coupled performance needs specific consideration in its algorithm.
1 INTRODUCTION

One major problem with the implicit solution of CFD problems is the dependency of nodes which appear in a computational molecule. In a second-order finite-difference approach, there are five nodes in each non-boundary computational molecule when the 2D Navier-Stokes equations are treated. To reduce the dependency from a five-diagonal matrix to a three-diagonal one, one classic approach is to employ Alternating Direction Implicit (ADI) algorithm [1]. In this algorithm, the five-node molecule is revised into two three-node molecules which are set and solved in the $x$ and $y$ directions. The solution can be accelerated by introducing a relaxation parameter in the formulation. An optimized value for this parameter needs numerical experimentation. The ADI method is still utilized as a practical aspect to solve the innovate implicit algorithm [2]

In the finite element approach, the dependency issue returns to the type of chosen elements, whether they are triangular or quadrilateral, and the type of grid distribution. In another words, the size of computational molecule directly depends on the number of elements which share a node. This story is almost true for all finite element methods. However, for a quadrilateral element choice (which is the case study in this work) the number of nodes involves in each row of the assembled matrix is at most nine irrespective of being interior or boundary volumes. This statement is similarly applicable to both finite-volume and finite-element-volume approaches. Generally speaking, the assembled matrices resulted from either a finite element or a finite volume method are nine-diagonal matrices if quadrilateral element is selected. However, there are many zeros between the non-zero elements of matrix. To reduce the computational efforts in solving the assembled matrix, numerous strategies have been developed and examined. One of the most widely algorithm is the line-by-line algorithm. In this algorithm, each of the flow governing equations is separately treated in a manner which produces a tridiagonal matrix. The generated tridiagonal matrices are then solved in different coordinate directions. Under-relaxation is required to control the convergence speed [3]. Sticking on this strategy, there are two major problems with the continuity equation which behaves different than the transport equations. Firstly, the continuity equation represents the pressure field while there is no physical appearance of the pressure terms in that. This problem can be removed by substituting the continuity equation with the pressure Poisson equation. Secondly, the continuity equation performs no directionality even by switching to the pressure Poisson equation. For a five-diagonal matrix, the strongly implicit procedure of Stone [4] can be regarded as a basic step toward solving the difficulty. As was mentioned, in the finite-volume-based approaches, the number of nodes in a computational stencil increases to nine. To improve the solution cost, there have been attempts to extend the Stone’s algorithm to nine-diagonal matrices [5]. Peric [6] also reduces the nine-diagonal matrix to a seven-diagonal one by eliminating the diagonals which belong to the nodes lying in sharp corners of a computational molecule. The above literature somehow indicates the oriented study in at most 2nd-order finite volume approaches. The increase to higher order
approaches automatically enforces higher number of non-zero diagonals. For example, a fourth-order finite volume approach requires at least five non-zero diagonals to show a suitable performance [7].

Back to the first major problem with the continuity equation, Schneider and Raw [8] develop a fully implicit finite-volume-based finite-element method which eliminates the need for the pressure Poisson equation and takes the pressure role directly into the continuity equation. Their approach is fully implicit and produces a 27-diagonal matrix in treating the 2D Navier-Stokes equations. The solution of the resulting matrix have been numerously investigated through utilizing different bandit and sparse matrix solvers [9]. Unfortunately, the task for solving the large scale CFD problems leads to a demand for high performance computational methods capable of solving the large sparse nonlinear systems. The aerospace industry similar to many other industries is in an increased need for parallel solvers on parallel computers [10]. To be in the leading edge of high performance computational research, the aforementioned algorithm has been also extended in order to use the advantages of parallel computing on multiprocessors or multicomputers. In this regard, different ordering strategies have been investigated in the context of the employed sparse matrix algorithm [11]. In this work, we suitably extend the investigation from a fully coupled 27-diagonal matrix to three semi coupled 9-diagonal matrices. Then the created matrices are consequently solved on a single machine or parallel distributed machines. There are several open source libraries for parallel programming. One of them is PETSC developed by PETSC team [12] which provides great capabilities to its users. Moreover, it is possible to link PETSC with the other programs such as mesh generators, mesh decomposers, and linear solvers. The PETSC solver also enables the users to solve the system of equations using either iterative or direct solvers. In case of utilizing iterative solvers, choosing appropriate Krylov Subspace Procedure KSP and preconditioning PC techniques is so crucial for solving ill-posed matrices. Several different preconditioning techniques can be employed in Jacobi, SOR, Incomplete LU (ILU), Parallel ILU (Euclid accessed through the Hyper interface), LU and Parallel LU (SPOOLES [13], SuperLU-DIST [14], MUMPS [15] algorithms in PETSC. The best algorithms for solving sparse systems resulted from aerodynamics applications is often preconditioned iterative solvers which provide suitable nested hierarchy of tunable algorithms. However, the iterative-based solvers rely on the diagonal dominance of the equation system to guarantee convergence to the solution. A single equation that is not diagonally dominant can mitigate the otherwise economic virtue of certain iteration-based solvers. Therefore, in the finite-element formulation employed in this work, such equation can emerge, particularly if the solution matrix is not diagonally dominant [16]. Therefore, among the above mentioned preconditioning techniques available in PETSC, only SPOOLES, SuperLU-DIST and MUMPS lead to the direct solution of the system of algebraic equations which are suitable to be used in this work. In this work, the performance of the new algorithm is investigated and reported with respect to the fully coupled algorithm using SuperLU-DIST and MUMPS. Additionally, the additional requirements to enhance the algorithm’s
performance are addressed.

2 GOVERNING EQUATIONS

The current sparse matrix solvers are examined by solving the two-dimensional incompressible Navier-Stokes equations. The governing equations consist of the conservation statements for mass and momentums. The governing equations are given by

$$\nabla \cdot (\rho \vec{V}) = 0$$ (1)

$$\nabla \cdot (\rho \vec{V} u) = -\frac{\partial p}{\partial x} + \nabla \cdot (\mu \vec{V} u)$$ (2)

$$\nabla \cdot (\rho \vec{V} v) = -\frac{\partial p}{\partial y} + \nabla \cdot (\mu \vec{V} v)$$ (3)

where $\vec{V} = u\vec{i} + v\vec{j}$ and $p$, $\rho$, and $\mu$ represent velocity, pressure, density, and the molecular viscosity, respectively.

3 DOMAIN DISCRETIZATION

The solution domain is broken into a huge number of quadrilateral elements. The elements fully cover the solution domain with no overlapping. Figure 1 shows a small part of the solution domain. Nodes are located at the corners of elements and are shown by circles. The nodes are the locations of the unknown variables. Each node belongs

![Figure 1: A part of the solution domain illustrating four elements, one complete finite volumes, sixteen sub-volumes, and eight cell faces.](image)
to four neighboring elements. There are four quadrilaterals which enclose node $P$ in Fig. 1. To utilize the benefits of cell-centered schemes, each element is divided into four quadrilaterals by the help of its medians. The median is demonstrated by dashline in this figure. The cells are then constructed from the proper assemblage of these sub-quadrilaterals. As is seen, irrespective of the shape and distribution of the elements, each node is surrounded by a number of sub-quadrilaterals. The proper assemblage of neighboring sub-quadrilaterals around any non-boundary node creates a complete cell.

4 COMPUTATIONAL MODELLING

To utilize the advantages of finite element volume methods, the governing equations are initially integrated over an arbitrary volume, e.g., the shaded area or the cell face shown in Fig. 1. The employment of Gauss divergence theorem to the governing equations leads to

$$\int_A \rho \vec{V} \cdot d\vec{A} = 0$$

$$\int_A u(\rho \vec{V}) \cdot d\vec{A} = -\int_A p \: dA_x + \int_A (\mu \vec{\nabla} u) \cdot d\vec{A}$$

$$\int_A v(\rho \vec{V}) \cdot d\vec{A} = -\int_A p \: dA_y + \int_A (\mu \vec{\nabla} v) \cdot d\vec{A}$$

The above integrals are evaluated over the surface which encloses each cell. The surface area is indicated by $A$. The above equations are suitably discretized using finite difference scheme and finite element interpolations. In the above expressions, $d\vec{A} = dA_x \hat{i} - dA_y \hat{j}$ is a normal vector to the edges of cell. Using this definition, the above integrals can be evaluated by summation over the faces that enclose the cell center, i.e.,

$$\sum_{i=1}^{n_s} [\rho(u \: dA_x + v \: dA_y)]_i = 0$$

$$\sum_{i=1}^{n_s} [(\rho \vec{u} \: dA_x + \rho \vec{v} \: dA_y)]_i = -\sum_{i=1}^{n_s} (p \: dA_x)_i + \sum_{i=1}^{n_s} \left[ \mu \left( \frac{\partial u}{\partial x} \: dA_x + \frac{\partial u}{\partial y} \: dA_y \right) \right]_i$$

$$\sum_{i=1}^{n_s} [(\rho \vec{u} \: dA_x + \rho \vec{v} \: dA_y)]_i = -\sum_{i=1}^{n_s} (p \: dA_y)_i + \sum_{i=1}^{n_s} \left[ \mu \left( \frac{\partial v}{\partial x} \: dA_x + \frac{\partial v}{\partial y} \: dA_y \right) \right]_i$$

where $i$ counts the number of cell faces from 1 to $n_s$. There are 8 cell faces around each cell. To linearize the governing equations, the bar over $\vec{u}$ and $\vec{v}$ indicates that these velocity components are approximated from the known magnitudes of the preceding iteration. Such approximation is essential to linearize the nonlinear momentum convection terms. The rest of procedure is to relate the cell face magnitudes (identified by lower case letters such as $u$, $v$, and $p$ variables) directly to the nodal magnitudes (identified by upper case letters such as $U$, $V$, and $P$ variables) which represent the locations for the unknown
variables of the current algorithm. A simple idea for treating the right-hand-side terms is to use the finite element shape functions $N_j = 1 \ldots 3$. The treatment results in

$$p_i = \sum_{j=1}^{4} N_{ij} P_j$$

$$\left. \frac{\partial \phi}{\partial z} \right|_i = \sum_{j=1}^{4} \frac{\partial N_{ij}}{\partial z} \Phi_j$$

where $p_i$ identifies the magnitude of $p$ at the mid-point of $i$th edge of the cell face. The $j$ notation counts the node numbers of an element where the $i$th cell face is located inside it. Additionally, the variable $z$ represents either $x$ or $y$ coordinates and $\phi$ (and $\Phi$) represents either $u$ (and $U$) or $v$ (and $V$) velocity components. As was mentioned, lower and upper case letters are used to represent cell face and nodal magnitudes, respectively.

The above approximations end the pressure and diffusion term treatments at any cell face of $i$. However, more sophisticated expressions are required to treat the convection terms. In fact, the treatment should not disregard the convection-diffusion physics and concept. To respect the correct physics of the convection, Reference [17] employs an upwind-based scheme (known as a physical influence scheme) within quadrilateral and triangular elements. The reference shows that a physical-based treatment of the $x$-momentum governing equation can result in

$$\phi_i = \sum_{j=1}^{4} \alpha_{ij} \Phi_j + \sum_{j=1}^{4} \beta_{ij} P_j + \gamma_i$$

where $\alpha, \beta,$ and $\gamma$ represent matrix, matrix, and vector coefficients, respectively. The above statement indicates that $\phi (\equiv u, v)$ at cell face can be approximated by the proper assemblage of $\Phi (\equiv U, V)$ and $P$ influences. In fact, this approximation can be regarded as a pressure-weighted upwind scheme. As is observed in Eq.(12), contrary to the problems raised in the Introduction section, the pressure field is not obtained through solving the pressure Poisson equation in this study. In fact, the continuity preserves its original identification and the direct contribution of the pressure terms are forced through the mass flux statements at the cell faces.

The substitution of Eqs. (10-12) in Eqs. (7-9) provides a set of algebraic equations for each cell. It is given by

$$
\begin{bmatrix}
  c_{ij}^{uu} & c_{ij}^{uv} & c_{ij}^{up} \\
  c_{ij}^{vu} & c_{ij}^{vv} & c_{ij}^{vp} \\
  c_{ij}^{pu} & c_{ij}^{pv} & c_{ij}^{pp}
\end{bmatrix}
\begin{bmatrix}
  U_j \\
  V_j \\
  P_j
\end{bmatrix} =
\begin{bmatrix}
  d_i^p \\
  d_i^v \\
  d_i^p
\end{bmatrix}
$$

where $i$ and $j$ count the global node numbers, i.e., $i, j = 1 \ldots N_node$. It should be mentioned that the matrix is a diagonal matrix which is normally generated in implicit
finite element methods. Therefore, it is strongly sparse and need sparse solution strategies. The coefficients in the global assembled matrix is identified by \( c \). The first letter in each superscript depicts the type of equation, i.e., \( p \), \( u \), and \( v \) indicate continuity, x-momentum, and y-momentum equations, respectively. The second letter in the superscripts indicates which unknown the coefficient belongs to. The right-hand-side vector is shown by \( d \).

\[
\begin{bmatrix}
\vdots & \vdots & \vdots \\
\vdots & c_{ij}^{uu} & \vdots \\
\vdots & \vdots & \vdots 
\end{bmatrix}
\begin{bmatrix}
U_j \\
V_j \\
P_j 
\end{bmatrix}
= \begin{bmatrix}
\vdots \\
d_i^u - c_{ij}^{uv} V_j - c_{ij}^{up} P_j \\
\vdots \\
\vdots 
\end{bmatrix}
\] (14)

\[
\begin{bmatrix}
\vdots & \vdots & \vdots \\
\vdots & c_{ij}^{vv} & \vdots \\
\vdots & \vdots & \vdots 
\end{bmatrix}
\begin{bmatrix}
U_j \\
V_j \\
P_j 
\end{bmatrix}
= \begin{bmatrix}
\vdots \\
d_i^v - c_{ij}^{vu} U_j - c_{ij}^{vp} P_j \\
\vdots \\
\vdots 
\end{bmatrix}
\] (15)

\[
\begin{bmatrix}
\vdots & \vdots & \vdots \\
\vdots & c_{ij}^{pp} & \vdots \\
\vdots & \vdots & \vdots 
\end{bmatrix}
\begin{bmatrix}
U_j \\
V_j \\
P_j 
\end{bmatrix}
= \begin{bmatrix}
\vdots \\
d_i^p - c_{ij}^{pu} U_j - c_{ij}^{vp} V_j \\
\vdots \\
\vdots 
\end{bmatrix}
\] (16)

As was told in the Introduction section, a step-down from a fully coupled set to several semi-coupled sets needs introducing a relaxation parameter in ADI formulation. The issue is similarly true about the finite volume approaches [?]. Similarly, in the present algorithm, the convergence speed is controlled by introducing three relaxation parameters into the solution of the three sub-original set of diagonal matrices. The corrections are done using

\[
U_i^{\text{new}} = (1 - \omega_u) U_i^{\text{odd}} + \omega_u U_i^{\text{new}}
\] (17)

\[
V_i^{\text{new}} = (1 - \omega_v) V_i^{\text{odd}} + \omega_v V_i^{\text{new}}
\] (18)

\[
P_i^{\text{new}} = (1 - \omega_p) P_i^{\text{odd}} + \omega_p P_i^{\text{new}}
\] (19)

The magnitudes of the relaxation parameters depend on the chosen test cases. They are chosen around 0.7 in solving the current test cases.
5 THE RESULTS

The performance of the extended algorithm is examined by testing a standard benchmark. The test case is a cavity which its upper lid moves with a constant velocity. Because of several recirculation zones in the cavity, achieving the accurate solution is a challenge. However, in this study, we do not focus on the solution itself and its accuracy because the accuracy of the basic method employed in this research has been already investigated and reported in the preceding publications, e.g., Refs. [9]. Here, we are interested in the performance of the extended semi-coupled algorithm in comparison with that of the fully-coupled algorithm. Therefore, we concisely present the results of a simple test which is arranged at a Reynolds number of 100.

Figure 2 provides a comparison between the results of the two algorithms with those of benchmark [18]. The two plots indicate the $U$ (left) and $V$ (right) velocity profiles at the centerlines of the cavity. The results of the current algorithms are almost the same and identical with those of benchmark. The numbers of iterations required to achieve the above solution are 7 and 66 for the fully-coupled and the semi-coupled algorithms, respectively. The current solution procedures are fulfilled in sequential manners. Of course, the results of sequential and parallel processing manners should not be different at all. Our experience also agrees with the above expectation. The utilized grid has a resolution of $50 \times 50$.

Figure 3 demonstrates the convergence rate or the reduction in $U$, $V$, and $P$ residuals during the iterations in testing the above defined test case. The results have been collected by utilizing the semi-coupled algorithm. The results are partly shown in this figure.
The number of required iterations to achieve the above accuracy is 7 if a fully-coupled algorithm is examined. As was mentioned in the introduction section, a shift from a fully-coupled algorithm to a semi-coupled one enforces the need for employing relaxation parameter in order to avoid divergence or to control the convergence speed. To improve the performance, we have employed suitable under-relaxations within the semi-coupled algorithm. However, the fully-coupled algorithm does not need any relaxation parameter. In the current work, we have employed a small under-relaxation around 0.001 at the beginning of the iterations for all three $U$, $V$, and $P$ unknown variables. This low under-relaxations are soon recovered to the magnitudes around 0.6-0.8 within the next few iterations. This range of relaxation parameters is because different grid resolutions require different relaxation magnitudes. The residuals are obtained using a grid size of 50×50. As is seen in Fig. 3, some oscillatory behaviors are observed at the beginning of the iterations. However, they are smoothen within the next few iterations. Slow convergence of the pressure variable indicates that the employed strategy needs extra suitable modifications. The utilized sparse solver is MUMPS in this test case.

Figure 3 and the accompanied discussions provide the performance of the MUMPS direct sparse solver. There is a question how the performance of other direct sparse solvers would be. To provide the answer, Fig. 4 illustrates the performance of the fully-coupled and the semi-coupled algorithms using two other sparse solvers of PETSC-LU and SuperLU-DIST. To extend our conclusions for a wide range of matrix sizes, these direct solvers are used to solve the CFD problem on three different matrix sizes of 7500, 30000, and 120000 which belong to the grid resolutions of 50×50, 100×100, and 200×200, respectively. The two plots in Fig. 4 indicate that MUMPS solver is more efficient than PETSC-LU and SuperLU-DIST solvers for both types of algorithms. Irrespective of the
matrix size, MUMPS, SuperLU-DIST, and PETSC-LU can be ranked from the most to the least efficient algorithms, respectively. However, in solving larger size of matrices, SuperLU-DIST efficiency is faded more comparing with the efficiency of the two other solvers if a semi-coupled algorithm is used. It should be mentioned that the iteration times reported in Fig. 4 are just for one iteration if a full matrix is solved, or they are the sum up the times required to solve three sub-matrices in a semi-coupled algorithm.

Since MUMPS performs better speed-up on a single processor, we further discuss its performance on parallel computers. Figure 5 shows the performance of MUMPS on several types of parallel machines to solve the fully-coupled algorithm. The performance is provided for a wide range of sparse matrix sizes. This figure indicates that the sequential run time is more than the parallel run time when the matrix is large; however, the time is in favor of the sequential procedure if the matrix size is not large. On the other hand, the iteration time is almost the same on different sorts of the parallel machines. It seems the communication time overwhelsms the computational time. This is why the parallel machines do not enhance the speed-up so much.

Figure 5 only compares the speed-up within one iteration. As was mentioned in the Introduction section, the number of iterations required to achieve solution with certain accuracy is different for the fully-coupled and the semi-coupled algorithms. To present a fair comparison between the two algorithms, the overall run times required to solve the test case are reported in Fig. 6. The left plot is for a fully-coupled algorithm. It is very similar to Fig. 5. Comparing Fig. 6 (left) with Fig. 5 indicates that the number of iterations to achieve the solution with an specified accuracy is around 7 for testing different sparse matrix sizes. However, the overall run time of MUMPS is different on different
Figure 5: The required time for the MUMPS solver in order to solve a fully-coupled algorithm on parallel distributed machines.

Figure 6: The MUMPS overall run time for parallel processing using either the fully-coupled (left) and or the semi-coupled (right) algorithms.

sorts of parallel processors if a semi-coupled algorithm is utilized. Firstly, the figure emphasizes that, irrespective of the size of parallel machines, the overall solution time always increases if a semi-coupled algorithm is utilized. Secondly, the figure demonstrates that in solving small size matrices, the performance is deteriorated when the number of processors increases. It seems this is not consistent with the basic philosophy of parallel
processing. In fact, the communication time overwhelms the computational time in solving small size matrices. However, the figure shows that the speed-up improves for a larger number of parallel processors when the sparse matrix is large enough.

6 CONCLUSIONS

- The implementation of a new pressure-weighted upwinding scheme in a finite element volume context enables us to break the fully sparse matrix into several smaller sparse systems.

- The implicit sparse system is solved using different sparse solvers.

- The MUMPS direct sparse solver performs better than SuperLU-DIST and PETSC-LU solvers irrespective of the sparse matrix sizes.

- Although the run-time per iteration improves in the semi-coupled algorithm, the fully coupled algorithm provides better overall-run-time and speed-up than the semi-coupled algorithm over a wide range of matrix sizes.

- The investigation shows that the considerable communication time does not permit better speed-up in solving small sparse systems; however, the speed-up is improved if the sparse matrix size is large enough.
REFERENCES


