A Parallel finite element Multigrid solver for incompressible Navier Stokes equations

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Technology

IN

Computational Science

by

Meesala Raviteja

Department of Computational and Data Sciences

Indian Institute of Science

BANGALORE – 560 012

JUNE 2016

Dedicated to my Parents

Acknowledgements

I would like to thank my advisor Dr. Sashikumaar Ganesan for his able guidance during the course of the project. This thesis would have not been possible without the timely suggestions and technical inputs he provided. Working with him has immesely helped me grow acadmecially and professionally.

I express my sincere gratitude towards all the faculty members of the Department. Their teaching and support have continuously inspired me to passionately persue research. I am highly thankful to the SERC facility at IISc. for granting the access and support whenever necessary to work with Cray XC40 and the other machines available to smoothly conduct my thesis work .

I would also like to thank all my colleagues (Jagannath, Birupaksha, Bhanu Teja, Shweta, Satish, Vijay Kumar) for making sure working in the lab was a rewarding experience. The journey at IISc would have been incomplete if not for the interactions and discussions I had with my classmates.

This endeavour would not have been possible without the support of my family. Their constant support and encouragement helped me perform to the best of my abilities.

Abstract

Fluid flows are generally modelled using the Navier Stokes Equations(NSE). The model can also be applied in the simulation of various practical applications such as weather, ocean currents, air flow over aircrafts etc. . As analytical solutions to these equations are hard to come by, numerical techniques such as finite element methods(FEM), finite volume methods(FVM), finite difference methods(FDM) etc. are used to solve these problems. This involves solving a system of algebraic equations, where the size of the problem increases with the desired level of accuracy in the solution.

FEM is the numerical approach adopted in this work. An effecient solution to these system of equations require the realization of robust iterative techniques. Initially, in this report, the saddle-point problem, that arise in a NSE system is discussed. An iterative technique based on a Krylov subspace method, GMRES, is adopted. The coupling of this technique with a multigrid method is considered, in order to improve the rate of convergence. A study is performed, through a set of simulations, on the rate of convergence over the variants of multigrid methods considered. Based on this, the scalability of the chosen algorithms is observed through the proposed parallel algorithm (a flat MPI implementation). Ascertaining the complexity involved in the iterative method adopted, we finally consider the performance of the technique on two test problems, a steady state problem and a time dependent problem by scaling their size.

Contents

List of Figures

Chapter 1

Introduction

1.1 Finite element methods

Fluid flows are one of the most commonly encountered physical phenomena. The Navier-Stokes equations (4.1) mathematically describe the continuum of the flow. These equations are obtained by extending the Newtons laws of motion for fluids. Thus, these represent the conservation of momentum for fluid flows. Coupled together with the continuity equation, that capture the conservation of mass, these equations enable us to solve for the fluid velocities and pressure. However, analytical solutions are not easily obtained for these set of equations. The finite element method provides a numerical framework, to solve these equations within a desired range of accuracy. While there are other known numerical techniques, FEM is preferred owing to its effeciency in solving for complex geometries. FEM reduces the given problem into a system of equations, generally linear in nature, that need to be solved. Based on the complexity of the problem and level of accuracy required in the solution, a scalable implementation of the algorithms becomes necessary. As one of the key componenents of the FEM based solution is the linear solver, that is required to solve a large system of linear equations, it plays a major role in realizing a scalable implementation of the FEM approach.

Linear solvers can be broadly classified as

- Direct Solvers
- Iterative Solvers

Direct Methods involve multiple stages in the process of solving the system. The major steps can be summarized as

1. Re-ordering to reduce fill in
2. Symbolic factorization, based on the pattern of non-zeros
3. Numerical factorization
4. Solve

These methods are preferred when the system is solved for multiple right-hand-side vectors as the first 3 steps need not be repeated.
Iterative techniques on the other hand operate by reducing the residual of the system in multiple steps. The effeciency of the technique depends on the rate of the convergence to the solution. Thus even though the complexity is O($\mathcal{N}$) for a single step, where $\mathcal{N}$ is the size of the system, there might not be a bound on the number of iterations required for convergence. Multigrid methods, in this context, provide a far more effecient and a scalable framework to obtain solution for the considered system of equations. The advantage of a multigrid algorithm lies in its ability to accelerate the convergence of an iterative solver.

1.2 Related work

In the previous implementations of FEM in our group, an effecient parallel multigrid algorithm has been implemented recently in [2] and [5] for the time dependent Convevction-diffusion systems, referred as CD systems henceforth. The current work is foucssed on solving the Navier Stokes equations of incompressible flows for a set of test problems using FEM as the numerical framework wtih a Multigrid algorithm as a solver.

The key difference between these two systems (CD and NSE) arises when the weak formulation of the systems are considered. A CD system results in a set of linear equations, in terms of unkowns at nodal points. In the NSE system the weak formulation gives rise to a non-linear equation. This can be linearized, by adopting either a fixed point method, newton method or a picards method. However, the final matrix-system that needs to be solved still differs from the one obtained in a CD-3D problem. While simpler iterative techniques, chosen as smoothers, in the multigrid algorithm, provide convergence in a CD-3D system, the choice of smoothers is not trivial in the case of NSE system. This is due to the equations arising out of the continuity equation, resulting in a non-diagonal dominant system. Partial pivoting becomes necessary if a direct technique is to be used and the complexity steadily increases as the number of pressure dof's or the size of the system, in general, increases. In [1] Rehman et.al discuss various re-ordering techniques that could be used, in order to adopt simpler iterative techniques, such as SILU. These techniques do provide an alternative if the implementation were to be independent of using a finite element package. However, there could be significant communication costs involved in using a renumbering strategy, when a parallel implementation is seeked. One way to avoid re-ordering or pivoting is by choosing a Krylov subspace method, as discussed in [3], with GMRES being the most popular among them. However, the accuracy of the solution becomes dependent on the refinement level of the grid. This leads to the idea of using a Flexible-GMRES method, where the residual vector is preconditioned using a multigrid algorithm. Vanka smoothers, as discussed in [3], is observed to be an effecient smoothing operator for the multigrid approach. Their application are well known not only in the field of CFD, but also in Computational Solid Mechanics [4].

1.3 Outline

The structure of the thesis is as follows. In Chapter 2 the parallel data structures used for the current implementation in ParMooN are discussed. Also the necessary nomenclature required is mentioned. A comparison performed over solvers, (multigrid in ParMooN, PasTix, MUMPS) implemented in ParMooN, is presented for a scalar test problem in Chapter 3. Chapter 4 discusses the NSE system and the multigrid methods explored for the current work. In Chapter 5 the set of experiments performed are presented. Finally in Chapter 6 the performance of the developed parallel technique is tested by considering a Steady state problem and a time dependent problem. The conclusions are presented in Chapter 7.

Chapter 2

ParMooN

ParMooN [17] is the parallel version of a Finite Element Package MooNMD (Mathematics and Object oriented Numerics in Magdeburg) [7], written entirely in C++. The object oriented nature eases the implementation of new problems. It is used in solving 2D or a 3D Convevction-diffusion and Navier-Stokes equations that arise in Computational Fluid Dynamics such as [8] [11]. Earlier parallel multigrid implementation of Scalar systems have been discussed in [5] [2]. The current implementation is performed using this package. Some of the key componenents involved in realizing a FEM based solution to a CD or NSE system, using ParMooN, are:

2.1 Domain decompostion

The initial stage of the implementation involves importing the geometry (or) the domain of the problem. The whole domain is discritized into triangluar/rectangular elements in case of a 2D domain, or hexahedral/brick/tetrahedral elements in case of a 3D domain. This process is done using internal routines available in ParMooN. Distributing the domain of the problem here helps in achieving coarse grain Parallelism. Several strategies can be thought out to partition the domain across the processes. If the domain is distributed more or less equally (w.r.t the number of dofs), we can achieve good load balancing in computation. Another strategy could be to try and minimize the "interface" area that results due to partitioning. This can affect the

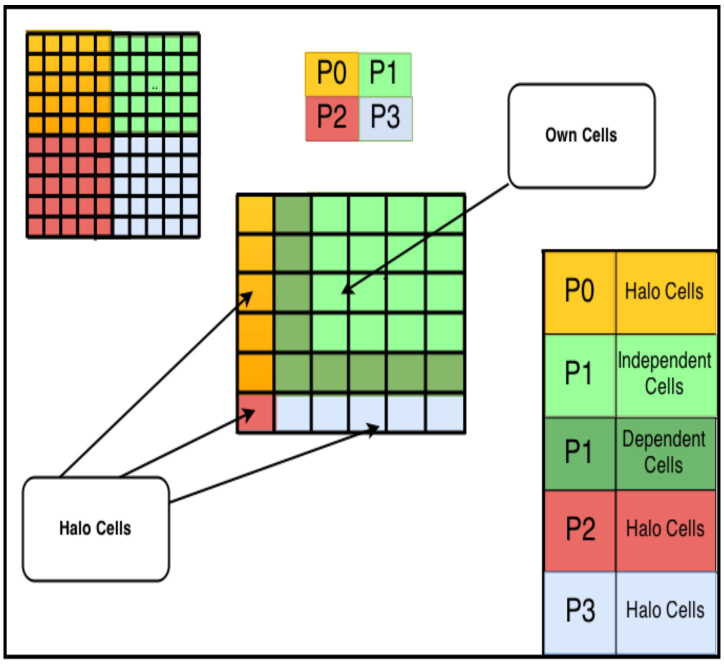

Figure 2.1: Cell Nomenclature in the subdomain of the process one (P1)

amount of communication that takes place across the processes. One among the most popular softwares used for this task is METIS/ParMETIS [9]. Each process is alloted a 'sub-domain' over which the computation is performed. Further refinement can be performed parallely by each process over their corresponding subdomains.
The sub-domain is further decomposed into discrete finite elements referred henceforth as cells. The geometrical properties of a cell, such as the number of vertex,edges and faces, depend on the choice of mesh discretization (Hexahedral, tetrahedral etc.). In a finite element implementation the information of the neighbourhood of a cell is of importance especially in assembling the system matrices. This leads to the requirement of a 'cell nomenclature' in the case of a parallel implementation.

Cell nomenclature: Each process contains a collection of cells. A cell's neighbour are the other cells in the collection which share either a vertex, edge or a face. In the parallel implementation, the neighbours of the cells at the interface of the subdomain which belong a neighbouring subdomain are also considered in the collection. Hence, if a cell belongs to the subdomain of the process it is termed as a own cell and otherwise a Halo Cell. As the cell becomes the basic unit of computation, each cell is uniquely mapped to each process(subdomain id) Own cells are further classified as dependent cell, if they have a Halo cell as a neighbour, and independent cells otherwise. This nomenclature is utilized in the construction of the DOF(nodes of the Finite element spaces) nomenclature.

2.2 Construction of the stencil & system matrix

The choice of the finite element spaces determines the stencil(pattern of non-zeros) of the system matrix. The FESpace and FEfunctions are two important classes in ParMooN, that capture this information to generate, the number of 'Degrees of Freedom' over which the solution is seeked and the system matrix. Other tasks performed during construction include numbering the DOF's[Degrees of Freedom] uniquely across the cells and assemling the system matrix by traversing the individual cells sequentially. These are performed in the parallel implementation using the below mentioned classes:

2.3 ParFEMapper

An important aspect of the parallel implementation is the DOF nomenclature. A precursor to this step is to generate a unique reference for a given DOF across processes. Once the global DOF id is obtained, the nomenclature is constructed. The nomenclature, illustrated with the figure, is as follows.

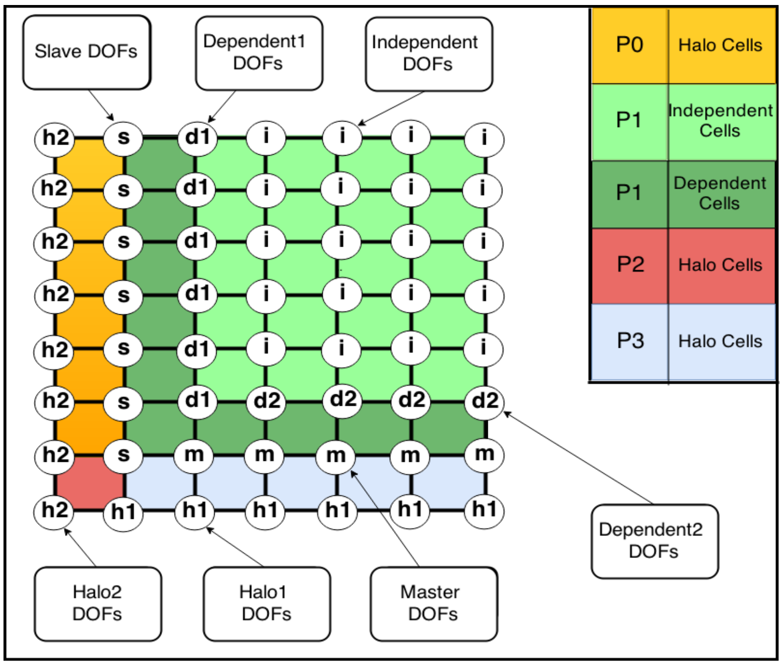

Figure 2.2: DOF Nomenclature for Q_1 finite element on the process P1

The DOF's belonging to a Halo cell become the halo/dependent DOFs. The DOFs belonging to both the halo cells and own cells form the interface across processes and form the set of interface DOFs. DOFs in own cells having a support with interface DOFs become the dependent DOFs, while the rest become the 'independent' Dofs. The interface DOFs are further classified to master('m') and slave('s'). This is required for the communication step, as each DOF information is communicated only by one process. Further, the dependent DOFs can be classified based on their support with the interface. Dependent type1 DOFs (d1) have a support with atleast one slave DOFs, while type2 (d2) have a support only with master DOFs. Further, the Halo DOFs can be classified similarly. Halo1(h1) DOFs are the ones with support on atleast one of the master DOFs, while Halo2(h2) DOFs are the one with support only with slave DOFs. Different compute strategies, especially in a multigrid technique, can be tested using the labels available on the DOFs.

2.4 ParFECommunicator

ParFECommunicator constructs the interface map of each process, which is required to relay and recieve information to and from the neighbouring processes. The map is constructed based on the distribution of master-slave DOFs by the ParFEMapper and the Halo/dependent DOFs present in each process. These maps also come into play when the restriction and prolongation operations are carried out in the multigrid approach.

2.5 Solvers

Currently ParMooN consists of (i)MUMPS, a parallel direct solver, (ii)UMFPACK, a sequential direct solver. Parallel Geometric Multigrid solvers are also available for solving scalar problems such as CD-3D. Parallel Geometric multigrid solvers for the NSE system were implemented as a part of the current work.

2.6 Multigrid in ParMooN

ParMooN supports two types of Geometric Multigrid approaches. One approach is to use a hierarchy of meshes, where the each mesh represents a level in the multigrid algorithm. The mesh with the least number of cells becomes the coarsest level and the finer mesh , obtained by refining the mesh at the lower level, represent the higher levels. The other approach is to use higher order finite elements, on a fixed mesh. The lowest order elements represent the coarsest level and the higher order elements represent the higher levels. In the current implementation we adopt a mixed strategy. A lower order finite element is used for all the coarser levels. On the finest mesh, two levels are introduced one with the lower order finite element and the othe a higher order finite element. The entire multigrid algorithm is captured in the figure below

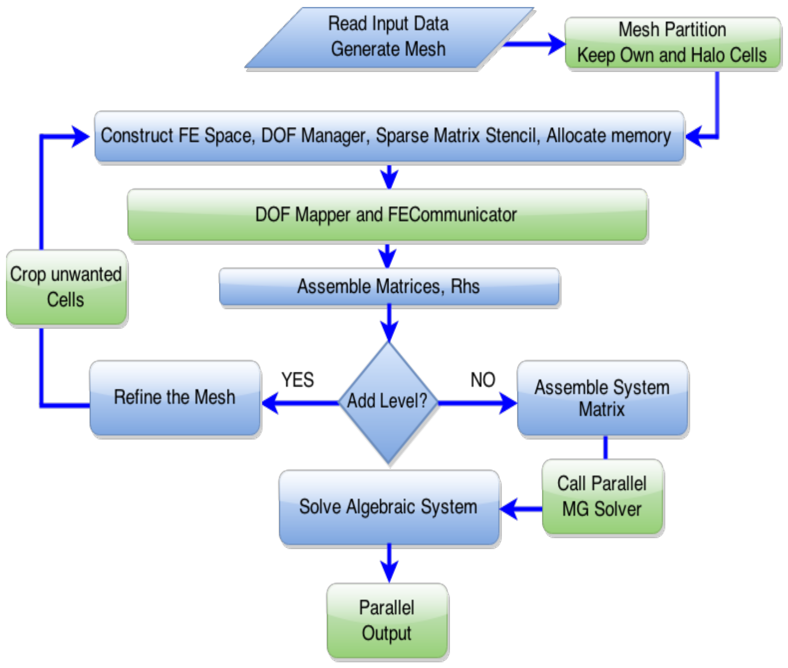

Figure 2.3: Flowchart of the Multigrid Solver in ParMooN

Restriction and Prolongation operators: The restriction and prolongation operators determine the effeciency of the multigrid implementation. In [13], Schieweck developed these operators for transfer of update from any arbitrary finite element level to the other. These have been implemented in ParMooN. In the case of parallel implementation, one needs to apply these operators with the help of the communicators for the interface/Halo DOF's. The implementation utilizes the knowledge of Own cells and Master DOF's apart from the parallel communicators discussed above. Further, when multigrid methods are adopted for non-linear problems, such as the NSE, an additional restriction operation is required to assemble the system matrices at all levels. This needs to be performed whenever the solution is updated iteratively.

Chapter 3

A comparison of solvers

A comparison of parallel solvers is considered in this chapter. MUMPS, a parallel direct solver was already implemented in ParMooN. Pastix, another direct solver, is implemented as a part of the current work.

3.1 PasTix - implementation

PasTix is a high performance parallel direct solver developed at INRIA [15]. While ParMooN has all its (distributed) local matrices in CSR format, PasTix requires the input to be given in a CSC format. However, considering the symmetry available in the system and the global ids available in ParMooN, the interface implemented for PasTix causes little overhead when considering the total time taken to solve the system.

3.2 Test results

The test problem considered here is a 3D poisson equation.

$$\begin{aligned} -\triangle u &= f \text{ in } \Omega := (0,1)^3 \\ u &= 0 \text{ on } \partial\Omega \end{aligned}$$

Two types of geometric grids are considered here. Grid1 consists of 262144 cells and a P1 order finite element is chosen on this grid, resulting in 274625 DOFs. Grid 2 consists of 32768 cells and a P2 order finite element is chosen with Grid 2. Both grids contain 274625 DOFs. The expirement was performed on the Tyrone cluster at SERC [16].

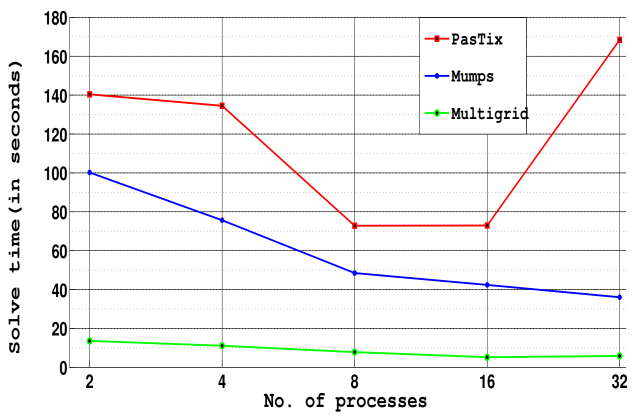

Figure 3.1: Solve time with a P1 element on Grid1

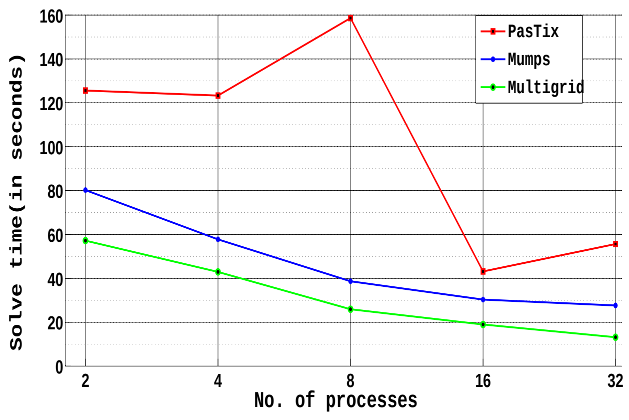

Figure 3.2: Solve time with a P2 element on Grid2

As can be seen from figures 3.1 and 3.2, the multigrid solver performs better among the considered direct solvers . Also observed is that the compute time involved is lower when higher order elements (P2 element) are considered, with the same system size. This is because of a better rate of convergence(fewer number of iterations) with higher order finite elements.

However, this has an adverse effect on the direct solvers as higher order elements relatively decrease the sparsity of the system. This could result in increased fill-ins in the factorization step and thereby higher compute time with direct methods.

A more comprehensive study of the multigrid algorithms for scalar problems in ParMooN can be found in [2] and [5].

Chapter 4

Multigrid for NSE

4.1 NSE

The Navier Stokes equation, is given as follows:

$$\rho\Big(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\Big) - \nabla\cdot S(\mathbf{u},p) = \rho\mathbf{f} \tag{4.1}$$

In the above equation ρ denotes the density, **u** denotes the velocity field, ∇ is the divergence operator, S denotes the stress tensor, **f** denotes the body force. Further, for an incompressible flow we have:

$$S(\mathbf{u},p) = -p\mathbb{I} + 2\mu\mathbb{D}(\mathbf{u}) \tag{4.2}$$

where p - denotes the pressure, $\mathbb{I}$ the identity matrix, μ denotes the viscosity of the fluid, $\mathbb{D}(\mathbf{u})$ the rate of strain tensor. Assuming the flow to be Newtonian, the strain tensor can be expressed as:

$$\mathbb{D}(\mathbf{u}) = \frac{1}{2}\Big(\nabla\mathbf{u} + \nabla\mathbf{u}^T\Big) \tag{4.3}$$

The continuity equation, for an incompressible fluid, is given as:

$$\nabla\cdot\mathbf{u} = 0 \tag{4.4}$$

Using the above equations, for the case of an incompressible newtonian fluid, (4.1) can be written as:

$$\rho\Big(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\Big) + \nabla p - \mu\big(\Delta\mathbf{u}\big) = \rho\vec{f} \tag{4.5}$$

The above equation is known as laplacian form (form II) , while equation (4.1) is known as stress tensor form (form IV) of the Navier stokes equation. For the stationary problem, the time derivative term is ignored.

4.2 System matrix

We use the Standard Galerkin method of FEM for the problem considered. We end up with a non-linear system of equations, due to the convective term $((\mathbf{u}\cdot\nabla)\mathbf{u})$. This is linearized by replacing the convective term with $(\mathbf{u}_h\cdot\nabla)\mathbf{u}_{h+1}$, i.e. we iteratively solve (4.1) for $\mathbf{u}_{h+1}$, by using $\mathbf{u}_h$, in the convective term, obtained from the previous iteration. The initial guess is assumed to be $\vec{0}$ [the zero vector].
Key choices that govern the stencil(or the sparsity pattern) of the system matrix are: the boundary conditions necessary to solve the given system of PDE's, and the order of the finite element used. However, the most basic representation of the matrix, irrespective of the above choices, in a 3D case, obtained using (4.1) is as follows:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & B_1^T \\ A_{21} & A_{22} & A_{23} & B_2^T \\ A_{31} & A_{32} & A_{33} & B_3^T \\ B_1 & B_2 & B_3 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ p \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ 0 \end{bmatrix}$$

Here A_{ij} represents square-submatrix obtained from the bilinear product of basis functions from the Velocity space. B_i represent a rectangular-submatirx obtained from the bilinear product of basis functions, one from the pressure space and the other from the velocity space. We observe that A_{ii} needs to be reassmbled, whenever **u** has been updated.

4.3 Saddle point problems

Saddle point problems can be observed in many applications related to computational sciences. A summary of these can be found in [14]. The class of saddle point problems can be generalized as follows. Consider a block linear system,

$$\begin{bmatrix} A & B1^T \\ B2 & C \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

This system is classified as a saddle point system if it satisfies one or more of the following conditions:

1. A is either symmetric or the symmetric part of A is positive semidefinite
2. C is symmetric positive semidefinite
3. C is zero matrix
4. B1 = B2

It can be seen from above that the saddle point system arising out of navier stokes equations satisfy conditions 1,3 & 4. The spectral properties of this system determine the performance of the krylov methods. These are studied in [14]. It has been observed that Krylov subspace methods show a poor rate of convergence for these systems. Preconditioners need to be adopted inorder to accelerate these methods.
In the current work, GMRESR is the adopted krylov subspace method and multigrid cycle (V/W cycle) is considered for a preconditioner.

4.4 Iterative algorithm

A modified version of GMRES is adopted. This version of GMRES is popularly known as Right-preconditioned GMRESR. The algorithm [10] is described below. The A matrix is the system matrix, M represents a multigrid-preconditioner. Multiple GMRES iterations (determined by the restart parameter) are performed within a non linear iteration.

Right-preconditioned GMRES

```
Compute $r_0 = b - Ax_0, \beta||r_0||_2$ and $v_1 = r_0/\beta$
for $j$=1,..,$m$ do
    Compute $z_j = M^{-1}v_j$
    Compute w = $Az_j$
    for $i$ = 1,..,.$j$. do
        $h_{i,j} = (w, v_i)$
        $w = w - h_{i,j}v_i$
    end
    Compute $h_{j+1,j}$ = $||w||_2$ and $v_{j+1}$ = $w/h_{j+1,j}$
    Define $Z_m$ = $[z_1, .., z_m]$, $\overline{H}_m$ = $\{h_{i,j}\}_{1\leq i\leq j+1;1\leq j\leq m}$
end
Compute $y_m$ = $argmin_y||\beta e_1 - \overline{H}_m y||_2$
Compute $x_m = x_0 + Z_m y_m$
If satisfied stop, else set $x_0$ = $x_m$ and restart
```

4.5 Smoothers for multigrid

Vanka smoothers are block Gauss-Siedel methods. There are variations in their implementation depending on how the blocks are formed. The size of the block depends on the finite element order of the velocity space. Two approaches are adopted in solving these local blocks, one is Jacobian-based approach known as diagonal vanka and the other is using either a direct solver or a krylov subspace method, known as the fully stabilized vanka methods [12].

These smoothers can be applied whenever there is a coupling present in any kind of system. The ratio of pressure dof's and velocity in the block would be higher in the cell based approach, in comparison to the nodal based approach. This could play a role in the convergence of the individual methods. We now consider the algorithms of these smoothers. Let S represent the system matrix, 'u' represents the solution vector, 'b' represents the r.h.s. Since the domain is decomposed into a collection of cells, K_i represents the i^{th} cell in the collection. 'row[dof]' represent the row corresponding to the 'dof' in S. 'nnz' represent the d.o.f. with non zeros in a given row. 'u_{update}' refers to the smoothing update to 'u'. $'damp'$ is the damping factor applied to the added update. Loc_S represent the local block formed. Loc_b represent the local rhs.

Cell vanka

```
for i=1,..,$N_{Cells}$ do
    Get cell: $K_i$
    Get $DOF_i$ := {$V_{dof}$ and $P_{dof}$ in $K_i$}
    for dof in $DOF_i$ do
        Get nnz := {non-zeros in row[dof]]} in S
        Reset index: 0
        for x in nnz do
            if x in $DOF_i$ then
                UpdateMatrix($Loc_S[index]$,S[i,x])
            end
            else
                UpdateRHS($Loc_b[index]$,u[x],S[i,x])
            end
            Increment index: 1
        end
    end
    Solve($u_{update}$,$Loc_S$ , $Loc_b$)
    Update u[dof] := u[dof] + (damp)*($u_{update}$)
end
```

In a parallel implementation of cell vanka, the Halo Cells in the collection are ignored from each process. However, there is a dependency of the interface DOFs, that are part of Dependent Cells, across processes. These dependencies could be resolved by communicating the values of these DOFs once a sweep is performed across all own cells in a given process.

Nodal vanka

```
for i=1,..,$N_{PDOf}$ do
    Get $row[p_{dof_i}]$ in S
    Get $DOF_i$ := {$V_{dof}$ in non zeros of $row[p_{dof_i}]$ & $p_{dof_i}$}
    for dof in $DOF_i$ do
        Get nnz:= {non-zeros in row[dof]]} in S
        Reset index: 0
        for x in nnz do
            if x in $DOF_i$ then
                UpdateMatrix($Loc_S[index]$,S[i,x])
            end
            else
                UpdateRHS($Loc_b[index]$,u[x],S[i,x])
            end
            Increment index: 1
        end
    end
    Solve($u_{update}$,$Loc_S$ , $Loc_b$)
    Update u[dof] := u[dof] + (damp)*($u_{update}$)
end
```

The key step of parallel implementation of a Nodal Vanka smoother is the collection of DOFs considered that are connected to a pressure DOF. The Halo (pressure/velocity) DOFs can be entirely ignored from smoothing at a given process and only the value provided by the neighbouring process can be considered. A similar strategy can be adopted for the interface(slaves in particular) DOFs.

4.6 Complexity of the multigrid Algorithm

Computational complexity The steps involved in the iterative technique adopted is to perform a GMRES operation followed by a multigrid V/W cycle. Within a multigrid cycle, iterative sweeps are performed at each of the levels considered. Assume $'v'$ number of multigrid cycles are performed within a GMRES iteration. Let $'PRS'$ be the number of pre-smoothing steps performed on each level, before performing a restrict operation and let $'POS'$ be the number of post-smoothing steps performed, after prolongate operation. Generally these two are chosen to be equal. Additionally one could perform multiple local sweeps $'L'$, before performing a communication update, in the case of a parallel implementation. The complexity of a smoothing step is ~$C * N_p$, where N_p is the number of pressure DOFs that arise in the system and $'C'$ is the complexity involved in solving these small blocked systems. Also the total number of DOFs N is ~$O(N_p)$. The restriction and prolongation operations considered are $O(N)$.

Hence, the total complexity in a multigrid sweep per level would be : $C * v * (2 * PRS) * L * O(N_l)$, where N_l is the total number of DOFs in a given level. In the case of 3D problems, coarser levels have ~$N_F/8$ DOFs, where N_F is the number of DOFs on a finer mesh. This indicates that the total DOFs across all levels is bounded by $O(N_F)$ Hence the total complexity can be approximated to be : $C * v * (2 * PRS) * L * O(N)$. The parameters v, PRS, L can be chosen appropriately to affect the rate of convergence. $'C'$ can be controlled based on the choice of the solver of the local systems

Communication Vs Computation For the sake of analysis of the implementation we adopt a simple cubic domain. The partition is assumed to be uniform, i.e. each process obtains a sub-cube of same volume. Consider a cube of side length $'A'$ as our physical domain. The toal number of nodes N (number of nodes) ~A^3. Assume the domain is partitioned across K processes. This gives a (sub)cube of volume (A^3/K). This gives a side length of $(A/K^{1/3})$ ~$(N/K)^{1/3}$ for the subcube. Since the interface determines the communication, we consider the faces of the cube which constitute the interface. Surface area of (sub)cube ~c*$(N/K)^{2/3}$. This approximation holds for every sub-cube (having varying number of faces as interfaces).

Every cube now shares a boundary(edges or faces or corner vertices) with atmost 26 neighbouring sub-cubes. Since the interaction of the sub-cube with these neighbours is bounded by a constant , and the information is only required locally we can assume that the communication complexity is directly proportional to the calculated area of the interface ~$(N/K)^{2/3}$.
Considering the ratio of computation to communication, we have that : Computation ~$c * (N/K)$, Communication ~$(N/K)^{2/3}$. The ratio we obtain is ~O($(N/K)^{1/3}$). Consider an embarassingly parallel program, with communication $O(1)$, the ratio is ~$O(N/K)$. This implies that the problem becomes I/O bound at a quicker rate than an embarrassingly parallel program. However, with a c_1 fold increase in size of the problem (through a higher level of refinement), we can scale the problem to c_1 times more processes. Also, if we can increase compute alone by a factor of c_2, we can again scale it to c_2 times more processes before it becomes I/O bound.

Chapter 5

Experiments

The proposed multigrid preconditioners are tested for convergence with the following test example. The physical domain ($\Omega \subset \mathbb{R}^3$) considered is a circular channel.

$$\Omega := \{ \big[r * cos(\theta), r * sin(\theta), k) \big] \} \tag{5.1}$$

where, $\{0 \leq \theta \leq 2\pi, 0 \leq k \leq 10, 0 \leq r \leq 1\}$

The stationary NS equation considered, in the non-dimensional form, is:

$$\Big((\mathbf{u} \cdot \nabla)\mathbf{u} \Big) - \frac{\nabla(\mathbb{D}(\mathbf{u}))}{Re} + \frac{\nabla p}{Re} = \vec{f} \tag{5.2}$$

Re - represents the reynolds number of the flow. In the current experiment: Re = 1, $\vec{f}$=0.

In the numerical simulations performed, 2 types of grid are used. Grid1 consists of 16787 DOFs and Grid2 consists of 124195 DOFs. Discontinuous finite element space(P1) was considered for pressure and continuous velocity space (P2). The same geometric grids were considered (Grid 3, with the same geometry as Grid1 with 15468 DOFs and Grid 4, same geometry as Grid 2 with 112724 DOFs.) for the continuous finite element space (continuous pressure(P1) and velocity(P2)). Further, the maximum GMRES iterations considered within a linear iteration is 25, unless the residual reduces by a factor of 100. 3 smoothing steps were considered. On the coarse grid the smoothing is stopped when the residual drops by a factor of 10. A maximum of 20 iterations were allowed otherwise. The 'W' cycle was used as the multigrid cycle. The non linear iteration is terminated when the residual (euclidean norm of the defect vector) is less than $1E-06$. All experiments in this section were conducted on a a 32 (8 NUMA nodes with 4 cores per node) core Linux workstation with 2.4 Ghz AMD Opteron processor and a memory of 128 GB.

5.1 Cell Vanka Vs Nodal Vanka

In this section we test the performance of the two vanka smoothers discussed earlier.

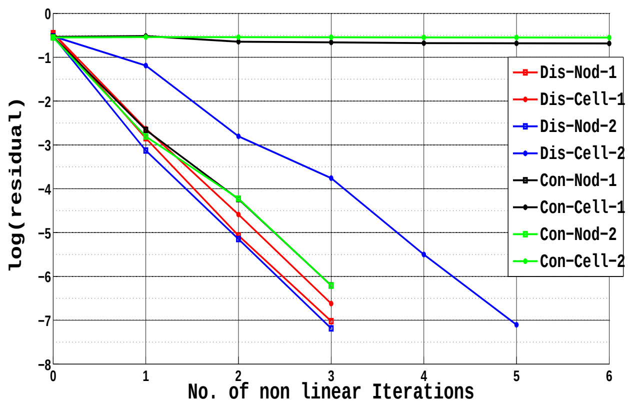

Figure 5.1: Residual along non-linear iterations for Cell vanka & Nodal vanka

As can be observed from the figure 5.1, with the discontinuous element schemes with both Nodal and Cell vanka converge. However for continuous element, Cell vanka doesn't converge. Further, it can be observed that with the increase in the refinement of the grid, Cell vanka converges comparatively slower than Nodal vanka.

5.2 Parallel variants with continuous, discontinuous pressure space

The parallel scalability of the mulitgrid algorithm is considered here. The strategy adopted for parallelism is as follows. The outer GMRES routine is trivial to parallelize, as each step consists of only a matrix vector product. In the case of multigrid smoothers, the smoothing is performed by choosing one pressure dof/cell at a time. Each process picks only the pressure DOFs that are labelled as 'master'. The halo DOFs are omitted as they explicitly belong to the neighbouring sub-domain and are picked by the neighbouring process. However, the slave pressure DOFs can also be considered to improve the rate of convergence as they belong to the same sub-domain. The velocity DOF's can also be chosen similarly while constructing the local system. It was observed that coarse grid solver diverges when the 'Master only' strategy was used. The plot (figure 5.2)obtained here is for continuous(con) and discontinuous(dis) finite element types mentioned above, using the master-slave strategy. Nodal vanka is the smoother considered throughout.

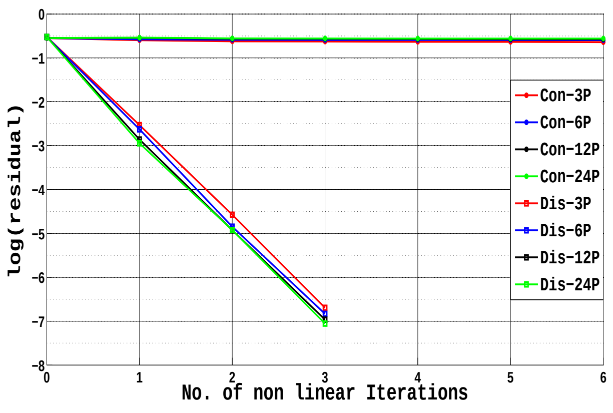

Figure 5.2: Residual at the end of each non-linear iteration with Continuous, Discontinuous pressure space

From figure 5.2, it can be observed that continuous finite elements are stringent to parallelism, with the smoothers considered. However, the algorithm scales well with little effect on convergence in the case of discontinuous finite elements. For the rest of the experiments ahead, we limit our choice to the Nodal Vanka smoother using the discontinuous finite elements.

5.3 Diagonal Vanka and Full Vanka

The solve step of the local systems, involved in the smoother, could be carried out in several ways. Since the size of the system is considerably small (in the range of 28 - 350), one could try out direct as well as iterative methods. Quickest approach of all could be to adopt a Jacobi smoother, with a single iteration. This approach is otherwise known as diagonal Vanka. There is as always the choice to solve the system exactly using direct solvers. Since the size of the system varies with the choice of finite elements (hexahedral or tetrahedral or P3 or P2 elements), one could control the use of direct solvers for smaller systems (< 100 is chosen here) and choose a krylov method for larger systems with a limit on the dimension of the krylov subspace (20 is chosen currently). This approach is otherwise known as Full Vanka. We now observe the rate of convergence and the compute time for these two variants.

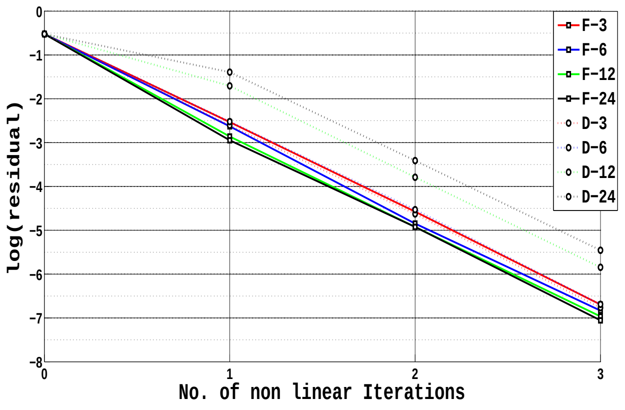

Figure 5.3: The rate of convergence for full and Diagonal Vanka

It can be observed in figure 5.3 that diagonal vanka takes more non-linear steps to converge. However, this may not reflect the actual compute time involved. This can be seen in figure 5.4.

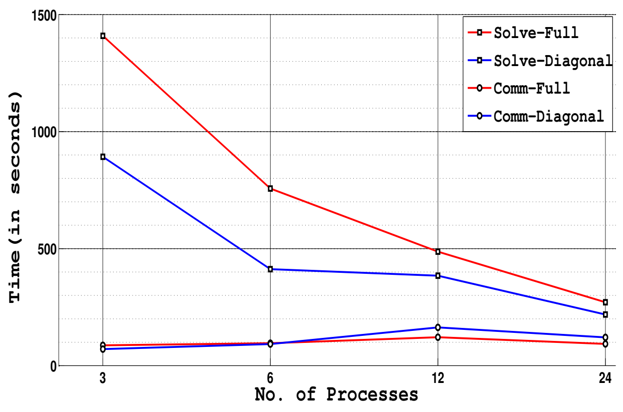

Figure 5.4: Comparison of Solve and Communication time for Diagonal and Full Vanka

Diagonal vanka is faster wrt. total compute time in comparison to full vanka. However, the overall communication time involved increases with the increase in the number of processes. The ratio of these (communication/solve) could help understand the rate at which these algorithms tend to become communication bound.

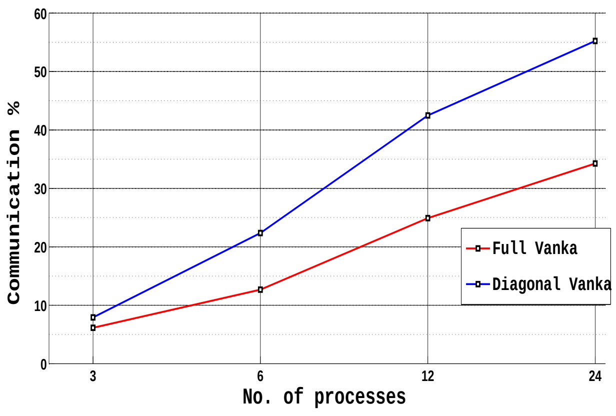

Figure 5.5: Ratio of Communication & Total Solve for Diagonal and full vanka

From the graph in figure 5.5, it could be inferred that diagonal vanka becomes I/O bound at a faster rate than full vanka. This implies that one could scale a given problem with full vanka to comparitively more number of processes. This might help in reducing the compute time of full vanka further.

5.4 Multilevel multigrid approach

Two different types of multigrid methods have been considered in this section. In the first approach ('MG0'), the same type of finite element space is chosen for all the hierarchy of meshes considered. In the second approach('MG1') the coarser level meshes adopt a lower order finite element space. At the finest level, a higher order space is also considered. Figure 5.6 captures the rate of convergence for both these approaches. Only Grid2 has been considered for the simulations.

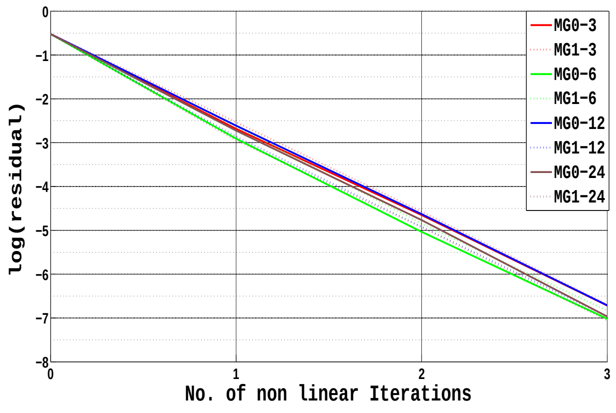

Figure 5.6: The rate of convergence for MG0 and MG1 approaches

From the plot in figure 5.6, it is observed that the rate of convergence appear to be similar. The difference in these schemes can be realized by considering the compute times of the individual methods.

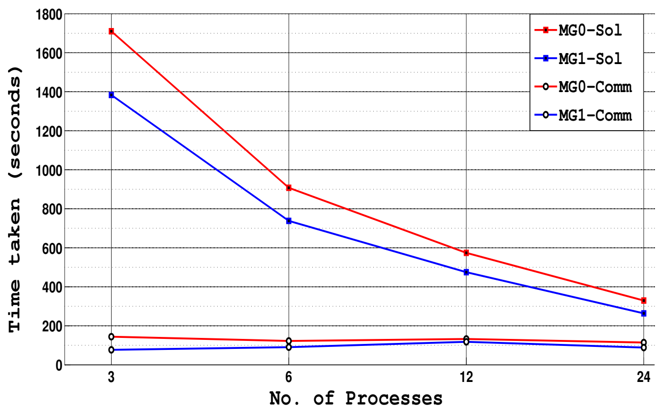

Figure 5.7: Plot for Solve and communication time involved for MG0 and MG1

From figure 5.6 and 5.7 it can be seen that the compute times are faster for 'MG1' type approach. This is expected as the coarser meshes have relatively lesser amount of DOFs in comparison to the 'MG0 ' approach. Also, this gain can be seen across processes. This implies that both these algorithms scale more or less similarly. Hence, a clear choice could be made to decrease compute time without sacrificing scalability and accuracy with the 'MG1' type approach.

5.5 Optimization

Reordering Compute A simpler way to increase computations keeping the communications constant is by introducing the idea of 'local sweeps' within each process. From the analysis in section 4.6 it can be obtained that this would decrease the overall communication ratio by atmost 20% . These local sweeps could affect convergence of the algorithm in adverse way as well. However, since the algorithm of the smoother here doesnt assume a fixed ordering in which the smoothing is to take place we exploit this in the reordering scheme here. Each pressure DOF is updated twice before the interface,halo DOFs are updated. The DOFs are coloured based on their distance from the interface. In the first sweep, the dofs are updated from the interface to the innermost region. The second sweep is carried out in the reverse order(referred as R1 in plot). Another strategy could be to reverse the order of what is proposed above (reffered as R2 in plot). These are compared in the plot here, with a default strategy (R3 in plot) i.e. both the sweeps are carried in the default DOF ordering (interface first and the rest next) and a strategy with a single sweep(R0 inplot).

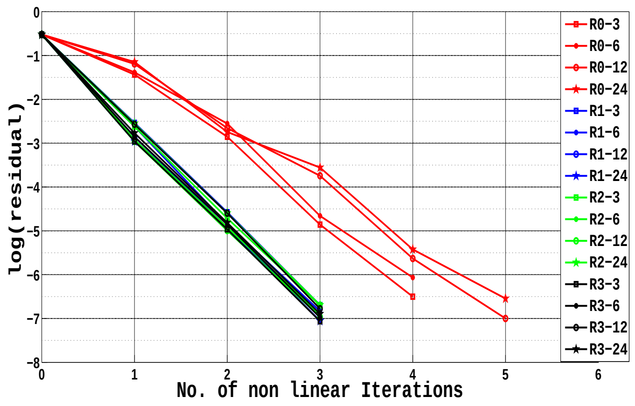

Figure 5.8: rate of convergence for re-ordering strategies

From figure 5.8, 5.9 and 5.10, it can be observed that 'local sweeps' have improved the rate of convergence (w.r.t the number of non-linear iterations). However, this may not necessarily be the case with increase in the number of 'local sweeps'.

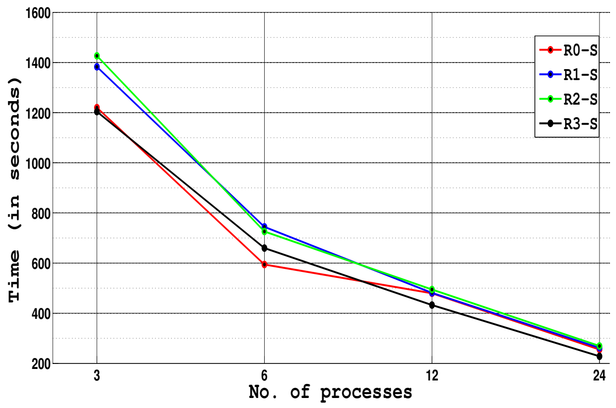

Figure 5.9: Total Solve time involved for re-ordering strategies

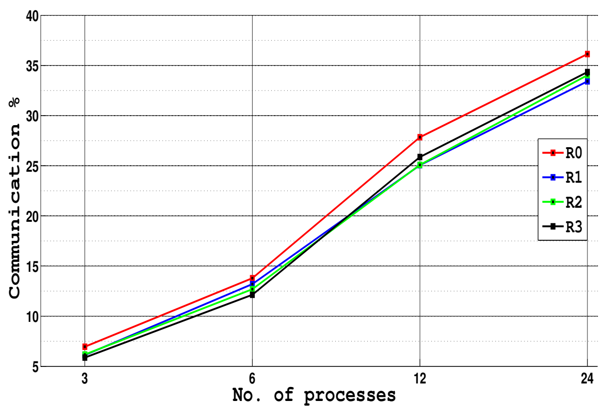

Figure 5.10: Communication-total time ratio for re-ordering strategies

The ’single sweep (R0)’ strategy has comparitively shorter solve time initially. However, the ’default strategy (R2)’ has a better solve time with higher number of processes. Also, the re-ordered methods have comparitively lesser amount of communication involved relative to the solve time.

Chapter 6

Test Problems and results

The performance of the multigrid solver is tested using two test problems. The experiments were performed on Sahasrat, a crayXC40 system, at SERC, IISc. The parameters studied here are the Speed up and the effeciency of the algorithm. The reference number of processes considered here is 24 to evaluate the same.

6.1 Steady state problem

The same example is chosen here as in Chapter 5. The mesh is refined, with 3 multigrid levels. The total size of the system considered here is 7,488,643 DOFs.

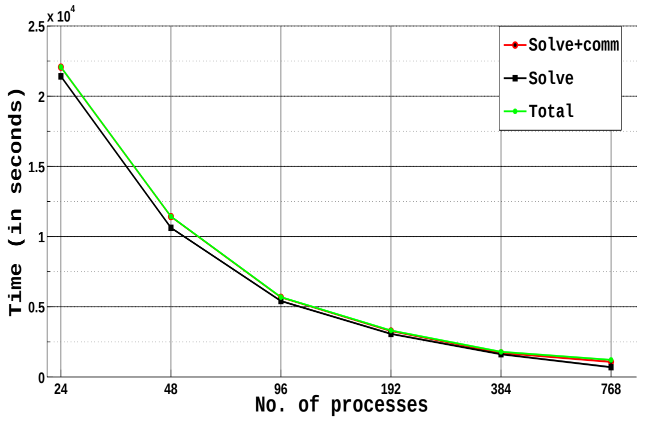

Figure 6.1: Time involved in Solve and Communication for steady state problem

It can be observed from figure 6.2 and 6.3 that the problem tends to become communication bound (30% of total time) when scaled to 768 processes. The effeciency drops drastically to around 60% with an increase from 384 -768 processors. Further increasing the number of processes could adversely affect the total compute time, as the communication at the coarser grids could consume more time than solve.

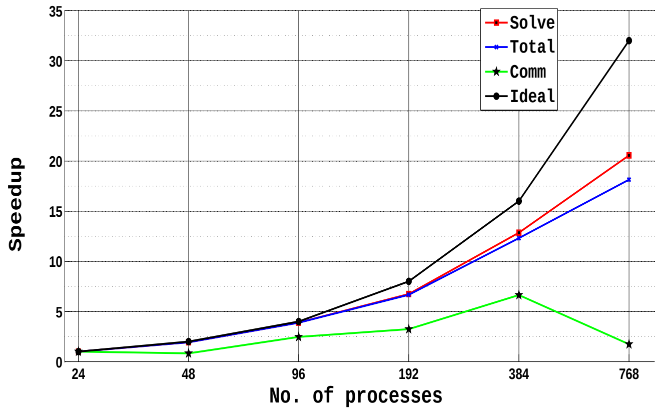

Figure 6.2: Speedup obtained for steady state problem

It can also be observed that there is an immediate jump in the amount of communication after 24 processors. This could be because of the out-of-node communication(24 cores per NODE) that begins when more than 24 processors are considered.

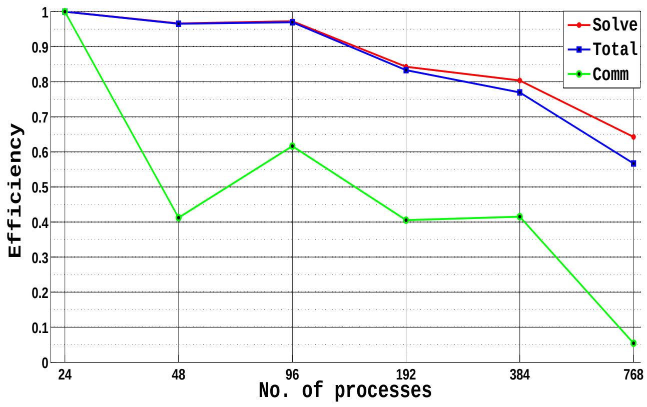

Figure 6.3: Parallel effeciency for steady state problem

6.2 Time-dependent problem

The physical domain ($\Omega \subset \mathbb{R}^3$) considered is a cubic domain.

$$\Omega := \left[0,1\right]^3 \tag{6.1}$$

The time dependent NS equation considered, in the non-dimensional form, is:

$$\frac{\partial \mathbf{u}}{\partial t} + \Big((\mathbf{u}.\nabla)\mathbf{u}\Big) - \frac{\nabla(\mathbb{D}(\mathbf{u}))}{Re} + \frac{\nabla p}{Re} = \vec{f} \text{ in } (0,T] \times \Omega \tag{6.2}$$

The boundary conditions are as follows:

$$\mathbf{u} := 0 \text{ at x = 0, x = 1, y = 0, y = 1} \tag{6.3}$$

$$\mathbf{u} := (1-x)*x*y*(1-y) \text{ at z = 0 (inlet velocity)} \tag{6.4}$$

$$n \cdot \nabla(\mathbf{u}) := 0 \text{ at z = 1} \tag{6.5}$$

The time derivative is discretized using a Crank-Nicholson scheme. Re - represents the reynolds number of the flow. In the current experiment: Re = 1, $\vec{f}$=0. The mesh is refined, with 3 multi-grid levels. The total size of the system considered here is 7,488,643 DOFs. The simulation is run for a total of 10 time steps, with a step size of 0.001 seconds.

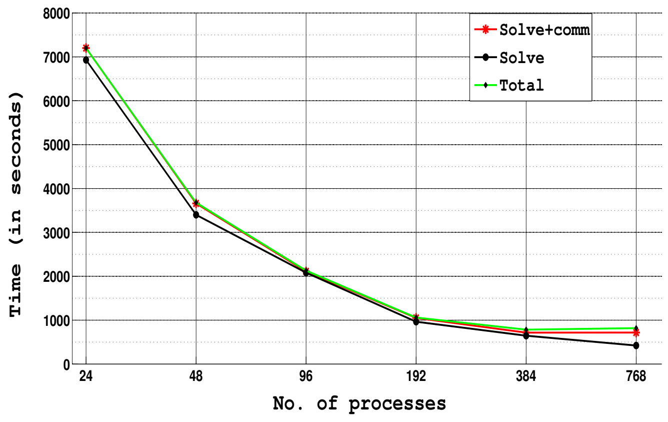

Figure 6.4: Execution time for time dependent problem

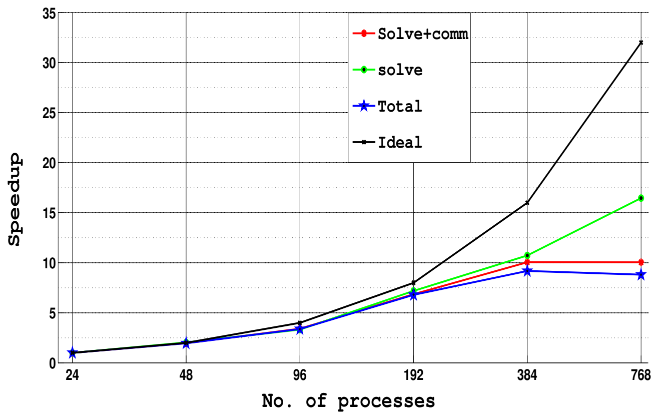

Figure 6.5: Speedup on time dependent problem

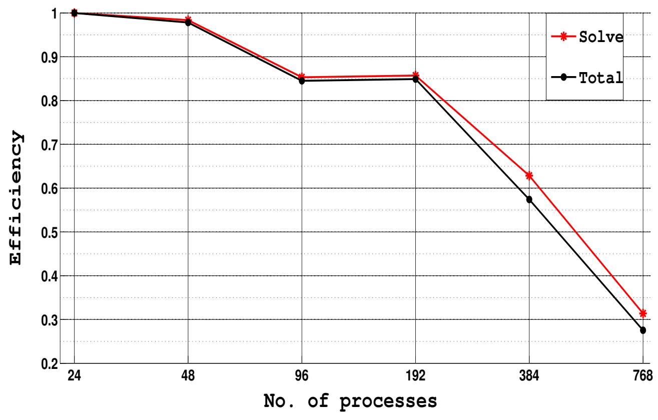

Figure 6.6: Parallel Effeciency of the time dependent problem

As can be observed from figures 6.4 and 6.5, the problem becomes I/O bound at a much faster rate than the previous problem, even though the sizes are similar. This could be because of the difference in partitioning of the domains. Also observed is that the compute times(convergence rates) are much faster for the cubic domain.

Chapter 7

Conclusions and Future Work

The parallel multigrid approach has been explored for the incompressible Navier Stokes equations. It has been observed that discontinuous pressure finite elements coupled with continuous velocity finite elements are comparitively more amicable to coarse grain parallelism than using a continuous pressure elements. While the initial experiments were performed over a circular channel, the time dependent problem is tested over a cubic domain. The results indicate a dependency of the rate of convergence with the physical geometry as well.

The algorithm becomes I/O bound at a quicker rate due to heavy communication at the coarser levels. As a part of future work, one could explore the option of paritioning each level to only an optimal number of processes to reduce the communication complexity involved. The optimal number could vary with the effeciency of the partitioning method adopted and is problem dependent. While adopting such a strategy, the same partiton could be solved by multiple processes to reduce/entirely avoid communication at the coarser levels. A complete partitioning, across all processes, could be used at the finer level. However, one could always realize more effecieny when the size of the problem is increased.

Bibliography

[1] M.Rehman and C.Vuik and G.Segal, "A comparison of preconditioners for incompressible Navier Stokes solvers", International Journal for Numerical Methods in Fluids 2008

[2] Finite Element Algorithm for Massively Parallel Computing, Shamim Abdus, MTech Thesis, SERC, Indian Institute of Science, 2015, MTech Thesis

[3] Volker John, Higher order finite element methods and multigrid solvers in a benchmark problem for the 3D NavierâĂŞStokes equations, International Journal for Numerical Methods in Fluids, 2002.

[4] Hilmar Wobker and Stefan Turek, Numerical Studies of Vanka-Type Smoothers in Computational Solid Mechanics, Advances in Applied Mathematics and Mechanics, 2002.

[5] Finite Element Simulation with Parallel Multigrid Solver, P.Badarla, MTech Thesis, SERC, Indian Institute of Science, http://www.serc.iisc.ernet.in/graduation-theses/BMPavanKumar.pdf , 2014

[6] P.R. Amestoy and I.S. Duff and J.-Y. L'Excellent, MUMPS MUltifrontal Massively Parallel Solver Version 2.0, 1998

[7] Volker John and Gunar Matthies, MooNMD-2013; a Program Package Based on Mapped Finite Element Methods, Comput. Vis. Sci., March 2004,

[8] An accurate and robustness finite element computations for three-dimensional breast cancer invasion model on realistic geometry, S.Lingeshwaran and S.Ganesan,

[9] George Karypis and Vipin Kumar, METIS – Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0, 1995

[10] Saad, Y., Iterative Methods for Sparse Linear Systems, 2003, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA

[11] Sashikumaar Ganesan and Lutz Tobiska, Arbitrary Lagrangian-Eulerian Finite-element Method for Computation of Two-phase Flows with Soluble Surfactants, J. Comput. Phys., May, 2012.

[12] Volker John and Lutz Tobiska, Smoothers in Coupled Multigrid Methods for the Parallel Solution of the Incompressible Navier-Stokes Equations, Int. J. Num. Meth. Fluids, 2000.

[13] F. Schieweck, A General Transfer Operator for Arbitrary Finite Element Spaces, 2000

[14] , Michele Benzi and Gene H. Golub and JÃűrg Liesen, Numerical solution of saddle point problems, ACTA NUMERICA, 2005.

[15] http://pastix.gforge.inria.fr/files/README-txt.html.

[16] http://www.serc.iisc.in/facilities/tyrone

[17] http://nmsc.serc.iisc.in/parmoon/