Performance Test of PETSc Library for the solution of Fully Coupled Velocity-Pressure Formulation for an Unstructured Finite Volume RANSE Solver

by

Hasnat Jamil

Supervisor: Michel Visonneau

A thesis submitted in partial fulfillment for the
degree of Erasmus Mundus Master of Science

in the
Computational Mechanics
Equipe Modélisation Numérique, CNRS
ECOLE CENTRALE DE NANTES

May 2010

Abstract

This research includes a numerical description of a coupled system of momentum and pressure equation and performance analysis in solving the system. The linear system of the coupled method is very hard to solve and time consuming compared to the segregated method (SIMPLE). It requires high memory also. So in this research, two solvers are used to check the performance in solving the coupled system. One is a linear solver included in ISIS-CFD, developed by EMN (Equipe Modélisation Numérique), CNRS at Ecole Centrale de Nantes. And another one is, Portable, Extensible Toolkit for Scientific Computation (PETSc) which is a set of data structures and routines for the scalable (parallel) solution of scientific applications modelled by partial differential equation.

Acknowledgements

I would like to express my gratitude to all of my group members to give me the opportunity to complete this research. I am deeply indebted to my supervisor Dr. Michel Visonneau, the Head of the CFD team (CNRS), Ecole Centrale de Nantes, whose support, stimulating suggestions and encouragement guided me all the time of research and writing this paper. Without his help, it would have been difficult.

I am very grateful to Dr. Patrick Queutey and Dr. Ganbo Deng for their relentless assistance in going through the ups and downs of the road. Their effective discussion and enthusiasm motivated me a lot.

I also want to thank Dr. Emmanuel Guilmineau, Dr. Kunihide Ohashi and Dr. Jeroen Wackers who lend their helping hand in the hours of need.

At the end, I am thankful to my family for their support and love throughout this period.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Strongly Coupled System

During the recent decades, most of the methodologies which have been developed to solve the Navier-Stokes equation, mostly rely on the Segregated or Decoupled method. The segregated method is based on the succecive solution of the momentum and pressure equation, instead of solving the whole global linear system of velocity and pressure unknowns. The momentum equation provides the velocity unknowns for a known pressure field and the pressure equation which derived from the mass equation, provides the pressure unknowns for the velocity field determined before. There are many external algorithm based on the prediction and correction phases developed to get an interative solution of the linear coupled system (SIMPLE, PISO, etc.). In the segregated method, the linear coupling is never solved to machine accuracy within a non-linear iteration, because it is considered that it would be waste of time to solved the coupled linear system where the non-linearities have not converged.

Although Segregated method has some certain advantages like, it requires less memory since it does not require to build the coupled linear system, it requires less time by avoiding the complete solution of the coupled system in a non-linear iteration; but most of the time, the under-relaxation is strongly needed for pressure, velocity, etc. Also the use of a pseudo-time derivative is mandatory (for steady flows) to create a diagonal dominance necessary for the segregated coupling method. So, because of under-relaxation factors (seldomly higher than 0.5) and pseudo-transient terms, the segregated approach is usually very slow to converge.

On the other hand, the strongly coupled system considers the coupling between the momentum and pressure equation and this whole global linear system is solved at once in every non-linear iteration. By using fully-coupled system, one may expect following benefits:

- It requires less number of non-linear iterations (compared to segregated method) as it delts the linear system in a coupled way.
- Higher under-relaxation parameter (>0.6) can be used for fully coupled method.
- The pseudo-time derivative is not necessary (unlike segregated method).
- It requires higher memory to build the coupled system as the system is ill conditioned and unsymmetric; but now a days the memory is becoming cheaper and thus this requirement doesnt give that much of a challenge.
- The linear system may take more time to be solved but availability of new powerful linear systems give opportunities to use fully coupled method; thanks to the use of more powerful preconditioning.

1.2 ISIS-CFD

The ISIS-CFD is an incompressible flow solver developed by the EMN (Equipe Modélisation Numérique), CNRS at Ecole Centrale de Nantes. It uses the incompressible unsteady Raynold-averaged Navier Stokes equation (RANSE). The solver is based on the finite volume method to build the spatial discretization of the transport equations. The face-based method is generalized to two-dimensional, rotationally-symmetric, or three-dimensional unstructured meshes for which nonoverlapping control volumes are bounded by an arbitrary number of constitutive faces. The velocity field is obtained from the momentum conservation equations and the pressure field is extracted from the mass conservation constraint, or continuity equation, transformed into a pressure-equation.In the case of turbulent flows, additional transport equations for modeled variables are solved in a form similar to the momentum equations and they can be discretized and solved using the same principles. The whole system is solved by Segregated or decoupled method (SIMPLE). Besides, the velocity-sudo-velocity-pressure coupled system is also implemented in ISIS-CFD for both 2D and 3D cases.[1]

The accuracy and robustness of the ISIS-CFD have been demonstrated in many international workshop, classical benchmarks and EU research projects. It is now available as a part of computing suite FINE^{TM}/MARINE commercialized by Numeca.

Now a days, one of the most important concern in CFD simulation is to reduce the computation time. The coupled system, which we are dealing with, is a huge, ill-conditioned matrix and it requires more time and more memory to be solved. So, although the coupled method is implemented in ISIS-CFD, it is important to analyse the performance. In this research

[1]Most of this paragraph is taken from the theoretical manual of FINE^{TM}/MARINE

we are going to check the performance of ISIS-CFD solver in solving coupled system and compare it with the PETSc.

1.3 PETSc

The Portable, Extensible Toolkit for Scientific Computation (PETSc) is a powerful package to develop large scale scientific application codes in Fortran, C and C++.This software has powerful set of tools for the numerical solution of partial differential equations and related problems on high-performance computers. It also supports parallel computations by employing Message Passing Interface (MPI) to communicate with multi-processor.

PETSc includes variety of libraries, each of which deals with particular family of objects (e.g. vectors, matrices, etc.) and operations. These objects and operations are derived from experience of the scientific computations. Some of the PETSc modules deals with:

- Index sets, including permutations, for indexing into vectors, renumbering, etc.
- Vectors
- Matrices (generally sparse)
- Distributed arrays (useful for parallelizing regular grid-based problems)
- Krylov subspace methods
- Preconditioners, including multigrid and sparse direct solvers
- Nonlinear solvers and
- Timesteppers for solving time-dependent (nonlinear) PDEs.

Each consists of an abstract interface (simply a set of calling sequences) and one or more implementations using particular data structures. Thus, PETSc provides clean and effective codes for the various phases of solving PDEs, with a uniform approach for each class of problems.

Chapter 2

Strongly Coupled Velocity-Pressure Formulation

2.1 Governing Equations

In a multi-phase incompressible flow of viscous fluid under isothermal conditions, the mass, momentum and volume fraction conservation equations can be written in Cartesian coordinates as (using the generalized Gauss' theorem):

$$\frac{\partial}{\partial t}\int_V \rho dV + \int_S \rho\left(\vec{U} - \vec{U_d}\right)\cdot\vec{n}dS = 0 \tag{2.1}$$

$$\frac{\partial}{\partial t}\int_V \rho U_i dV + \int_S \rho U_i\left(\vec{U} - \vec{U_d}\right)\cdot\vec{n}dS = \int_S \left(\tau_{ij}I_j - pI_i\right)\cdot\vec{n}dS + \int_V \rho g_i dV \tag{2.2}$$

$$\frac{\partial}{\partial t}\int_V c_i dV + \int_S c_i\left(\vec{U} - \vec{U_d}\right)\cdot\vec{n}dS = 0 \tag{2.3}$$

where,

V is the control volume bounded by the closed surface S moving with a velocity $\vec{U_i}$ with a unit normal vector $\vec{n}$ directed outward.

$\vec{U}$ and p represent, velocity and pressure field respectively.

τ_{ij} and g_i are viscous tensor and gravity vector respectively and I_j is a vector whose components vanish, except for the component j which is equal to unity.

c_i is the ith volume fraction for fluid i and is used to distinguish the presence ($c_i = 1$) or the absence ($c_i = 0$) of fluid i. Since volume fraction between 0 and 1 indicates the presence of mixture, the value of 1/2 is selected as a definition of the interface.

The effective flow physical properties (viscosity μ and density ρ) are obtained from each physical properties of constituent fluids (μ_i and ρ_i) with the following constitutive relations:

$$\rho = \sum_i c_i \rho_i; \qquad \mu = \sum_i c_i \mu_i; \qquad 1 = \sum_i c_i; \tag{2.4}$$

For a moving grid, the space conservation law also must be satisfied:

$$\frac{\partial}{\partial t} \int_V dV + \int_S \vec{U_d} \cdot \vec{n} dS = 0 \tag{2.5}$$

The general mass conservation Eq.(2.1) can be simplified by considering incompressible phases with constant densities ρ_i. By considering the constitutive relations (2.4) one arbitrary phase j can be isolated as (such that $\rho \neq 0$):

$$c_i = 1 - \sum_{i \neq j} c_i \tag{2.6}$$

$$\rho = c_j \rho_j + \sum_{i \neq j} c_i \rho_i = \rho_j + \sum_{i \neq j} c_i (\rho_i - \rho_j) \tag{2.7}$$

Substituting these relations (2.6) and (2.7) into the global mass conservation Eq.(2.1) yields

$$\begin{aligned} 0 &= \frac{\partial}{\partial t} \int_V \left(\rho_j + \sum_{i \neq j} c_i (\rho_i - \rho_j) \right) dV + \int_S \left(\rho_j + \sum_{i \neq j} c_i (\rho_i - \rho_j) \right) \left(\vec{U} - \vec{U_d} \right) \cdot \vec{n} dS \\ &= \rho_j \left[\frac{\partial}{\partial t} \int_V dV + \int_S \left(\vec{U} - \vec{U_d} \right) \cdot \vec{n} dS \right] + \left(\sum_{i \neq j} (\rho_i - \rho_j) \right) \left[\frac{\partial}{\partial t} \int_V c_i dV + \int_S c_i \left(\vec{U} - \vec{U_d} \right) \cdot \vec{n} dS \right] \\ &= \rho_j \left[\int_S \vec{U} \cdot \vec{n} dS \right] \end{aligned} \tag{2.8}$$

So the mass conservation simplifies as:

$$\int_S \vec{U} \cdot \vec{n} dS = 0 \tag{2.9}$$

2.2 Generic Discretisation Form

2.2.1 Discretised Momentum Equation

According to second order discretisation form of the Gauss theorem, the momentum equation Eq.(2.2) can be rewritten in a semi-discrete form as (in 2D for the sake of simplicity):

$$\frac{\partial}{\partial\tau}(\rho Vol\vec{U})_C + \frac{\partial}{\partial t}(\rho Vol\vec{U})_C + \sum C_{nb}^{UU}\vec{U}_{nb} + C_d\vec{U}_C + \overrightarrow{SrcU} + \int_{Vol_C}(\overrightarrow{\nabla p^i} + \overrightarrow{\nabla p^e})dV + \int_{Vol_C}\rho\vec{g}dV + \int_{Vol_C}\overrightarrow{SrcRij}dV = \vec{0} \tag{2.10}$$

where $\sum C_{nb}^{UU}\vec{U}_{nb}$ corresponds to the generic implicit discretisation of the convective and diffusive terms of momentum equations in which the excluded contribution from the center point is explicitly defined in the term $C_d\vec{U}_C$. The source terms that are evaluated at the center point C are:

- $\overrightarrow{SrcU}$ is a source term containing all the explicit terms coming from the spatial discretisation and from the isotropic turbulence model.
- $\int_{Vol_C}(\overrightarrow{\nabla p^i} + \overrightarrow{\nabla p^e})dV$ is the integral of the pressure where terms treated implicitly and explicitly in the pressure equation are distinguished by indices i and e.
- $\overrightarrow{SrcRij}$ is a source term containing the additional contributions appearing when an anisotropic non-linear turbulence model is used.

The time derivative in Eq.(2.11) can be evaluated using three-level Euler second-order accurate approximations:

$$\frac{\partial Q}{\partial t} \approx \frac{\delta Q}{\delta t} = e_c Q_c + e_p Q_p + e_q Q_q \tag{2.11}$$

where Q is a generic term and subscript c refers to the current time t_c, p the previous time t_p, and q the time t_q anterior to p. If the time step Δt is constant, then $t_p = t_c - 2\Delta t$. Coefficient $\{e_c, e_p, e_q\}$ are odtained from Taylor series expansion from t_c and depend on a possibly prescribed variable time step law $\Delta t(t)$. The fictitious local time differencing is evaluated by:

$$\frac{\partial Q}{\partial \tau} = (Q_c - Q_o)/\Delta\tau \tag{2.12}$$

where Q_o is a previous estimation of Q_c in the framework of the non-linear process. Finally the Eq.(2.11) can be rewritten as:

$$Vol_C e_C \rho_C \vec{U}_C + Vol_P e_P \rho_P \vec{U}_P + Vol_Q e_Q \rho_Q \vec{U}_Q + Vol_C(\rho_C\vec{U}_C - \rho_0\vec{U}_0)/\Delta\tau + \sum C_{nb}^{UU}\vec{U}_{nb} + C_d\vec{U}_C + \overrightarrow{SrcU} + \int_{Vol_C}(\overrightarrow{\nabla p^i} + \overrightarrow{\nabla p^e})dV + \int_{Vol_C}\rho\vec{g}dV + \int_{Vol_C}\overrightarrow{SrcRij}dV = \vec{0} \tag{2.13}$$

In order to build a pressure equation, a new pseudo-velocity field $\vec{\hat{U}}_C$ can be introduced:

$$\vec{\hat{U}}_C - \frac{1}{Vol_C}\sum C_{nb}^{UU}\vec{U}_{nb} = \frac{\overrightarrow{SrcU}}{Vol_C} \tag{2.14}$$

Injecting the definition of pseudo-velocity into the discretised momentum equation leads to:

$$\left[C_d + Vol_C e_C \rho_C + \frac{Vol_C \rho_C}{\Delta\tau}\right]\vec{U}_C + Vol_C\vec{\hat{U}} + \int_{Vol_C}(\overrightarrow{\nabla p^i} + \overrightarrow{\nabla p^e})dV + Vol_P e_P \rho_P \vec{U}_P$$
$$+ Vol_Q e_Q \rho_Q \vec{U}_Q - Vol_C \rho_0 \vec{U}_0/\Delta\tau + \int_{Vol_C}\rho\vec{g}dV + \int_{Vol_C}\overrightarrow{SrcRij}dV = \vec{0} \tag{2.15}$$

The following notations are introduced in ISIS-CFD:

$$\int_{Vol_C}\overrightarrow{SrcRij}dV = Vol_C\overrightarrow{SrcUh} \tag{2.16}$$

and:

$$Sk_{imp} = SrcU + Vol_C\left[\frac{\partial p^e}{\partial x} + \rho g_x + SrcUh\right] + Vol_P e_P \rho_P U_P$$
$$+ Vol_Q e_Q \rho_Q U_Q + Vol_C \rho_0 U_0/\tau \tag{2.17}$$
$$Sk_{exp} = SrcV + Vol_C\left[\frac{\partial p^e}{\partial y} + \rho g_y + SrcVh\right] + Vol_P e_P \rho_P V_P$$
$$+ Vol_Q e_Q \rho_Q V_Q + Vol_C \rho_0 V_0/\tau \tag{2.18}$$

By introducing the discretisation coefficients for the integrated implicit pressure gradient, one gets:

$$\int_{Vol_C}\frac{\partial p^i}{\partial x}dV = Vol_C\sum C_{nb}^{UP}p_{nb} \tag{2.19}$$
$$\int_{Vol_C}\frac{\partial p^i}{\partial y}dV = Vol_C\sum C_{nb}^{VP}p_{nb} \tag{2.20}$$

where the summation is made here on all the points of the stencil (including the central point C). Using the notations introduced in the ISIS-CFD code, one gets:

$$\left[C_d + Vol_C e_C \rho_C + \frac{Vol_C \rho_C}{\Delta\tau}\right]U_C + Vol_C\hat{U} + Vol_C\sum C_{nb}^{UP}p_{nb} + Sk_imp - SrcU = 0 \tag{2.21}$$

$$\left[C_d + Vol_C e_C \rho_C + \frac{Vol_C \rho_C}{\Delta\tau}\right]V_C + Vol_C\hat{V} + Vol_C\sum C_{nb}^{VP}p_{nb} + Sk_exp - SrcV = 0 \tag{2.22}$$

In ISIS-CFD, $\overrightarrow{Suhh} = \frac{\overrightarrow{SrcU}}{Vol_C}$, which will be stored in the arrays $Suhh$ and $Svhh$. The Eqn.(2.22) and (2.23) can be finally rewritten as:

$$C_{Diag-Solv}U_C + Vol_C\hat{U} + Vol_C\sum C_{nb}^{UP}p_{nb} = -Sk_imp + Vol_CSuhh \tag{2.23}$$

$$C_{Diag-Solv}U_C + Vol_C\hat{V} + Vol_C\sum C_{nb}^{VP}p_{nb} = -Sk_imp + Vol_CSvhh \tag{2.24}$$

with $C_{Diag-Solv} = C_d + Vol_Ce_C\rho_C + \frac{Vol_C\rho_C}{\Delta\tau}$.

2.2.2 Discretised Pressure Equation

To build the pressure equation, the starting point is the mass conservation equation for multi-fluid or mono-fluid incompressible flows integrated on a control volume Vol_C.

$$\int_{\partial Vol_C}\vec{U}\cdot\vec{n}dS = 0 \tag{2.25}$$

The mass flux can be computed by using the discretised expression of the velocity at point C from Eqn.(2.23) and (2.24):

$$U_C = \frac{1}{C_{Diag-Solv}}[-Vol_C\hat{U} - Vol_C\sum C_{nb}^{UP}p_{nb} - Sk_{imp} + Vol_CSuhh] \tag{2.26}$$

$$V_C = \frac{1}{C_{Diag-Solv}}[-Vol_C\hat{V} - Vol_C\sum C_{nb}^{VP}p_{nb} - Sk_{exp} + Vol_CSvhh] \tag{2.27}$$

The source terms Sk_{imp} and Sk_{exp} can be re-arranged to make each elementary fluxes visible:

$$\begin{aligned}
-Sk_{imp} + Vol_CSuhh &= -SrcU - Vol_C\left[\frac{\partial p^e}{\partial x} + \rho g_x + SrcUh\right] + SrcU - Vol_Pe_P\rho_PU_P \\
&\quad - Vol_Qe_Q\rho_QU_Q + Vol_C\rho_0U_0/\Delta\tau \\
&= -Vol_C\left[\frac{\partial p^e}{\partial x} + \rho g_x + SrcUh\right] - Vol_Pe_P\rho_PU_P \\
&\quad - Vol_Qe_Q\rho_QU_Q + Vol_C\rho_0U_0/\Delta\tau
\end{aligned} \tag{2.28}$$

$$\begin{aligned}
-Sk_{exp} + Vol_CSvhh &= -SrcV - Vol_C\left[\frac{\partial p^e}{\partial y} + \rho g_y + SrcVh\right] + SrcV - Vol_Pe_P\rho_PV_P \\
&\quad - Vol_Qe_Q\rho_QV_Q + Vol_C\rho_0V_0/\Delta\tau \\
&= -Vol_C\left[\frac{\partial p^e}{\partial y} + \rho g_y + SrcVh\right] - Vol_Pe_P\rho_PV_P \\
&\quad - Vol_Qe_Q\rho_QV_Q + Vol_C\rho_0V_0/\Delta\tau
\end{aligned} \tag{2.29}$$

Finally the mass flux can be expressed as:

$$\vec{U}\cdot\vec{n} = \left[\frac{1}{C_{Diag-Solv}}\left[Vol_C\left(-\overrightarrow{\hat{U}} - \overrightarrow{\nabla p^i} - \overrightarrow{\nabla p^e} - \rho\vec{g} - \overrightarrow{SrcUh}\right)\right] - \frac{1}{C_{Diag-Solv}}\left[Vol_P e_P \rho_P \overrightarrow{U_P} + Vol_Q e_Q \rho_Q \overrightarrow{U_Q} - Vol_C \rho_0 \overrightarrow{U_0}/\Delta\tau\right]\right]\cdot\vec{n} \tag{2.30}$$

The pressure equation is then obtained reconstructing the fluxes at the faces and the assembling them:

$$\begin{aligned} &- Div\left(\frac{Vol_C}{C_{Diag-Solv}}\overrightarrow{\hat{U}}\right) - Div\left(\frac{Vol_C}{C_{Diag-Solv}}\overrightarrow{\nabla p^i}\right) - Div\left(\frac{Vol_C}{C_{Diag-Solv}}\rho\vec{g}\right) \\ &- Div\left(\frac{Vol_C}{C_{Diag-Solv}}\overrightarrow{\hat{SrcUh}}\right) - Div\left(\frac{Vol_C}{C_{Diag-Solv}}\overrightarrow{\nabla p^e}\right) - Div\left(\frac{Vol_P}{C_{Diag-Solv}}e_P\rho_P\vec{U}_P\right) \\ &- Div\left(\frac{Vol_Q}{C_{Diag-Solv}}e_Q\rho_Q\vec{U}_Q\right) + Div\left(\frac{Vol_C}{C_{Diag-Solv}}\rho_0\frac{\vec{U}_0}{\Delta\tau}\right) = 0 \end{aligned} \tag{2.31}$$

which leads to the follwoing discritized form of the pressure equation:

$$\sum C_{nb}^{P\,P} P_{nb} + \sum C_{nb}^{P\,\hat{U}} \hat{U}_{nb} + \sum C_{nb}^{P\,\hat{V}} \hat{V}_{nb} = -S_P \tag{2.32}$$

where:

$$\begin{aligned} Div\left[\frac{Vol_C}{C_{Diag-Solv}}\overrightarrow{\nabla p^i}\right] &= \sum C_{nb}^{P\,P} P_{nb} \\ Div\left[\frac{Vol_C}{C_{Diag-Solv}}\overrightarrow{\hat{U}}\right] &= \sum C_{nb}^{P\,\hat{U}} \hat{U}_{nb} + \sum C_{nb}^{P\,\hat{V}} \hat{V}_{nb} \\ Div\left[\frac{Vol_C}{C_{Diag-Solv}}\left(\overrightarrow{\nabla p^e} + \rho\vec{g} + \overrightarrow{SrcUh} - \rho_0\frac{\vec{U}_0}{\Delta\tau}\right)\right] & \\ + Div\left[\frac{Vol_P}{C_{Diag-Solv}}e_P\rho_P\vec{U}_P + \frac{Vol_Q}{C_{Diag-Solv}}e_Q\rho_Q\vec{U}_Q\right] &= S_P \end{aligned} \tag{2.33}$$

2.3 The Velocity-Pseudo-velocity Pressure coupled System

Now the linear coupled linear system can be built for the current control volume from the previous equations:

$$\begin{aligned}
C_{Diag-Solv}U_C + Vol_C\hat{U} + Vol_C\sum C_{nb}^{UP}p_{nb} &= -Sk_imp + Vol_CSuhh \\
C_{Diag-Solv}U_C + Vol_C\hat{V} + Vol_C\sum C_{nb}^{VP}p_{nb} &= -Sk_imp + Vol_CSvhh \\
\hat{U}_C - \frac{1}{Vol_C}\sum C_{nb}^{UU}U_{nb} &= \frac{SrcU_C}{Vol_C} \\
\hat{V}_C - \frac{1}{Vol_C}\sum C_{nb}^{VV}V_{nb} &= \frac{SrcV_C}{Vol_C} \\
\sum C_{nb}^{PP}P_{nb} + \sum C_{nb}^{P\hat{U}}\hat{U}_{nb} + \sum C_{nb}^{P\hat{V}}\hat{V}_{nb} &= -S_P
\end{aligned} \tag{2.34}$$

The data-structure that was used for this research is:

$$\begin{aligned}
\vec{X} = \Big(&\hat{U}_1, \hat{V}_1, U_1, V_1, P_1, \ldots, \hat{U}_i, \hat{V}_i, U_i, V_i, P_i, \ldots, \hat{U}_{ncell}, \hat{V}_{ncell}, U_{ncell}, V_{ncell}, P_{ncell}, \\
&\hat{U}_{bnd}, \hat{V}_{bnd}, U_{bnd}, V_{bnd}, P_{bnd}\Big)
\end{aligned} \tag{2.35}$$

2.4 Global Structure of the Linear Coupled System

The system of momentum and mass conservation equations can be written as:

$$\begin{aligned}
(E+A)\vec{U} + G_x\vec{P} &= \overrightarrow{f_U} \\
(E+A)\vec{U} + G_x\vec{P} &= \overrightarrow{f_V} \\
D_x\vec{U} + D_y\vec{V} &= 0
\end{aligned} \tag{2.36}$$

where $\vec{U}$, $\vec{V}$ and $\vec{P}$ are grouping all the unknowns for the whole computational domain. E is the diagonal matrix and A is such that $diag(A) = 0$. $E + A$ contains all the dicretisation coefficients corresponding to the implicit unsteady, pseudo-steady, convective and diffusive terms. G_x (resp. G_y) is the matrix corresponding to the discretization of $\frac{d}{dx}$ (resp. $\frac{d}{dy}$) and D_x (D_y) are the matrices corresponding to the discretization of Div. One can recall here that $D_x = G_x$ and $D_y = G_y$ if the discretization formula used to build the divergence and gradient operators are identical, which should be the case since the continuous operators are identical. Now one can introduces the pseudo-velocity fields by:

$$\begin{aligned}
\vec{\hat{U}} &= -A\vec{U} + \overrightarrow{f_U} \\
\vec{\hat{V}} &= -A\vec{V} + \overrightarrow{f_V}
\end{aligned} \tag{2.37}$$

Using this definition, we can build the usual pressure equation:

$$\begin{aligned}
\vec{U} + E^{-1}\vec{\hat{U}} + E^{-1}G_x\vec{P} &= \vec{0} \\
\vec{V} + E^{-1}\vec{\hat{V}} + E^{-1}G_y\vec{P} &= \vec{0} \\
\vec{\hat{U}} &= -A\vec{U} + \overrightarrow{f_U} \\
\vec{\hat{V}} &= -A\vec{V} + \overrightarrow{f_V} \\
G_xE^{-1}\vec{\hat{U}} + G_yE^{-1}\vec{\hat{V}} + G_xE^{-1}G_x\vec{P} + G_yE^{-1}G_y\vec{P} &= \vec{0}
\end{aligned} \tag{2.38}$$

which leads to the new coupled formulation:

$$M\vec{X} = \vec{f} \tag{2.39}$$

with $\vec{X} = [\vec{\hat{U}}, \vec{\hat{V}}, \vec{U}, \vec{V}, \vec{P}]$. M is given by:

$$M = \begin{pmatrix}
E^{-1} & 0 & Id & 0 & E^{-1}G_x \\
0 & E^{-1} & 0 & Id & E^{-1}G_y \\
Id & 0 & A & 0 & 0 \\
0 & Id & 0 & A & 0 \\
G_xE^{-1} & G_yE^{-1} & 0 & 0 & G_xE^{-1}G_x + G_yE^{-1}G_y
\end{pmatrix} \tag{2.40}$$

Chapter 3

Programming with PETSc

3.1 MPICH and PETSc Installation

3.1.1 MPICH Installation

All PETSc programs use the MPI(Message Passing Interface) standard for message-passing communication. MPICH provides compiler wrappers such as mpif90, mpicc and mpicxx. The latest version of MPI, "MPICH2" can be downloaded and installed while configuring PETSc by including `--download-mpich=1` option. But because of the limitation of lab's computer, the previous version was installed (mpich-1.2.7p1). This version can be downloaded from the site http://www.mcs.anl.gov/research/projects/mpi/mpich1/download.html.

One can create a folder in `/work/common/`, say "PETSc" to install MPICH and PETSc there. The compressed mpich installation file can be extracted in this folder and then cofigured and installed by using the following commands:

```
tar zxof mpich.tar.gz
cd mpich-1.2.7p1
./configure --with-device=ch_p4 --prefix=/work/common/PETSc/mpich-1.2.7p1/ch_p4\
--with-common-prefix=/work/common/PETSc/mpich-1.2.7p1
make
make install
```

After installing MPICH, it can be tested, whether the mpich is installed correctly or not, by using the commands:

```
cd example/test/pt2pt/
make testing
```

3.1.2 PETSc Installation

The latest version of PETSc "Petsc-3.0.0-p8" can be downloaded from the site www.mcs.anl.gov /petsc. The compressed installation file can be extracted at the same folder "/work/common/PETSc".

```
tar zxof petsc-3.0.0-p8.tar.gz
cd petsc-3.0.0-p8/
```

Before the installation one should specify `PETSC_DIR` and `PETSC_ARCH` variable in `~/.bashrc` as the path of petsc directory and the architecture respectively.

```
PATH=/work/common/PETSc/mpich-1.2.7p1/bin:${PATH}
export PETSC_DIR=/work/common/PETSc/petsc-3.0.0-p8
export PETSC_ARCH=linux-gnu-c-debug
```

The PETSc can be configured and installed by the commands

```
./config/configure.py --with-mpi-dir=/work/common/PETSc/mpich-1.2.7p1 \
--download-f-blas-lapack=1 --download-hypre=1 --download-ml=1 --with-shared=0
make all
```

For this research, the external packages, Hypre and Trillion/ML are included with the installation. To check the correct installation of PETSc, one can test the examples by using command:

```
make test
```

3.2 Writing PETSc Programs

3.2.1 Include Files

PETSc has fortran interface which can work with fortran 77 and 90 compiler. The PETSc program uses CPP preprocessing, for which it requires to use the PETSc include files in the directory petsc/include/finclude. It allows the use of `#include` statements that define PETSc objects and variables. The include files can be used by the statement such as:

```
#include "finclude/includefiles.h"
```

Some of the include files that are used for the research are as follows:

```
petsc.h  - base PETSc routines
petscvec.h  - vectors
```

```
petscmat.h  - matrices
petscpc.h  - preconditioners
petscksp.h  - Krylov subspace methods
petscsys.h  - system routines
```

3.2.2 PETSc Objects

The PETSc has its own objects or data type to be used in the program. Some of them are same as fortran. The fortran data type also can also be used in the program. But it is important that one should not mix the variable with different data types. Some of the objects used in the research are given below:

- *PetscInt* - PETSc type that represents integer. Similar to *integer* in fortran.
- *PetscReal* - PETSc type that represents a real number. Similar to *real* in fortran.
- *PetscScalar* - PETSc type that represents either a double precision real number or a double precision complex number. Similar to *double precision* in fortran.
- *Vec* - The is one of the simplest PETSc objects which is used to denote vectors. Vectors are used to store discrete PDE solutions, right-hand sides for linear systems, etc.
- *Mat* - Abstract PETSc matrix object which is used in various matrix computation.
- *KSP* - Abstract PETSc object that manages all Krylov methods.
- *PC* - Abstract PETSc object that manages all preconditioners.

3.2.3 PETSc Initialization and Finalization

In fortran the PETSc is initialize by the command,

```
call PetscInitialize(PETSC_NULL_CHARACTER,ierr)
```

`PetscInitialize()` automatically calls `MPI_Init()` if MPI has not been not previously initialized. In certain circumstances in which MPI needs to be initialized directly (or is initialized by some other library),the user can first call `MPI_Init()` (or have the other library do it), and then call `PetscInitialize()`. By default, `PetscInitialize()` sets the PETSc world communicator, given by `PETSC_COMM_WORLD`, to `MPI_COMM_WORLD`.

All PETSc programs should call `PetscFinalize()` as their final (or nearly final) statement, as given in Fortran formats:

```
call PetscFinalize(ierr)
```

This routine handles options to be called at the conclusion of the program, and calls `MPI_Finalize()` if `PetscInitialize()` began MPI. If MPI was initiated externally from PETSc (by either the user or another software package), the user is responsible for calling `MPI_Finalize()`.

3.2.4 Error Checking

The Fortran version of PETSc routine has as its final argument an integer error variable. The error code is set to be nonzero if an error has been detected; otherwise, it is zero. For example, the call of `KSPSolve()` in Fortran is given below, where `ierr` denotes the error variable:

```
call KSPSolve(ksp,b,x,ierr)
```

The most common reason for crashing PETSc Fortran code is forgetting the final `ierr` argument.

3.2.5 Passing Null Pointers

In PETSc fortran functions, if one wants to pass a 0 (null) argument, users must pass `PETSC_NULL_XXX` to indicate a null argument (where `XXX` is `INTEGER`, `DOUBLE`, `CHARACTER`, or `SCALAR` depending on the type of argument required); otherwise passing 0 from Fortran will crash the code. For example, to pass a null argument for the 5th element of `MatCreateSeqAIJ()` can be written as:

```
call MatCreateSeqAIJ(PETSC_COMM_WORLD,N,N,5,PETSC_NULL_INTEGER,A,ierr)
```

3.2.6 Vector Operations

In PETSc, the vector objects are defined as `Vec`. There are many commands that can be used for various vector operations. Here, some of them, which are used in the research, will be explained. There are two types of vector in PETSc: sequential and parallel (MPI based). This two types of vector can be created by the following commands respectively:

```
call VecCreateSeq(PETSC_COMM_WORLD,PetscInt n,Vec v,PetscErrorCode ierr)
call VecCreateMPI(PETSC_COMM_WORLD,PetscInt n,PetscInt N,Vec v,PetscErrorCode
ierr)
```

where `PETSC_COMM_WORLD` is used to communicate with the MPI. The local size of the vector (for current processor) is defined by `'n'`, and global size is defined by `'N'` (requires for

parallel vector). For parallel vector, one can either specify the local size `'n'` and let the PETSc to determine about the global size by putting `PETSC_DETERMINE` or specify the global size and let the PETSc to decide about the local size by putting `PETSC_DECIDE`. For example:

```
call VecCreateMPI(PETSC_COMM_WORLD,n,PETSC_DETERMINE,v,ierr)
```

or

```
call VecCreateMPI(PETSC_COMM_WORLD,PETSC_DECIDE,N,v,ierr)
```

In the 2nd case, the PETSc will choose a reasonable partition trying to put nearly an equal number of elements on each processor.

One can create a vector with the same format of an existing vector by using the command:

```
call VecDuplicate(Vec old,Vec new,ierr)
```

There are couple of functions to assign values in a vector. To assign one by one value in a vector, one can use `VecSet()`; but it is not a very efficient way. By using `VecSetValues()` one can assign many values (in array format) in a vector at a time. The command can be written as:

```
  call VecSetValues(Vec x,PetscInt ni,PetscInt ix[],PetscScalar y[],InsertMode
$ iora,ierr)
```

where `x` is the vector in which the values will be inserted, `ni` the number of elements to be added. `ix` is the array of global indices of the vector where the values will be added and `y` is the array of values. The `iora` is a flag which has two options, either `INSERT_VALUES` or `ADD_VALUES`, where `ADD_VALUES` adds values to any existing entries, and `INSERT_VALUES` replaces existing entries with new values. As the vector entries are specified in global location, each processor can contribute any vector entries, regardless of which processor 'owns' them; any nonlocal contributions will be transferred to the appropriate processor during the assembly process. One thing must be noted that `VecSetValues()` uses 0-based row and column numbers in fortran as well as C.

Once all of the values have been inserted with `VecSetValues()`, one must call

```
call VecAssemblyBegin(Vec x,ierr)
```

followed by

```
call VecAssemblyEnd(Vec x,ierr)
```

to perform any needed message passing of nonlocal components. In order to allow the overlap of communication and calculation, the users code can perform any series of other actions between these two calls while the messages are in transition.

Function Name	Operation
VecAXPY(Vec y,PetscScalar a,Vec x,ierr)	$y = y + a * x$
VecAYPX(Vec y,PetscScalar a,Vec x,ierr)	$y = y + a * y$
VecWAXPY(Vec w,PetscScalar a,Vec x,Vec y,ierr)	$w = a * x + y$
VecAXPBY(Vec y,PetscScalar a,PetscScalar b,Vec x,ierr)	$y = a * x + b * y$
VecScale(Vec x, PetscScalar a,ierr)	$x = a * x$
VecMAXPY(Vec y,int n, PetscScalar *a, Vec x[])	$y = y + \sum_i a_i * x[i]$
VecNorm(Vec x,NormType type, double *r,ierr)	$r =\| x \|_{type}$
VecMax(Vec x, int *idx, double *r,ierr)	$r = \max x_i$
VecMin(Vec x, int *idx, double *r,ierr)	$r = \min x_i$
VecAbs(Vec x,ierr)	$x_i =\| x_i \|$
VecReciprocal(Vec x,ierr)	$xi = 1/x_i$

TABLE 3.1: PETSc Vector Operation

To visualize a vector, one can use the command:

```
call VecView(Vec vec,PetscViewer viewer,ierr)
```

where `viewer` is an visualization option which includes `PETSC_VIEWER_STDOUT_SELF` and `PETSC_VIEWER_STDOUT_WORLD`. The `PETSC_VIEWER_STDOUT_SELF` is a default standard output option. The `PETSC_VIEWER_STDOUT_WORLD` is a synchronized standard output option where only the first processor opens the file. All other processors send their data to the first processor to print.

When a vector is no longer needed, it should be destroyed with the command

```
call VecDestroy(Vec x,ierr)
```

3.2.7 Matrix Operations

The matrix is the most important part of a system to be solved. There are different types of matrix format appropriate for different problems. PETSc currently supports dense storage and compressed sparse row storage (both sequential and parallel) formats. The use of PETSc matrices involves the following actions: create a particular type of matrix, insert values into it, process the matrix, use the matrix for various computations, and finally destroy the matrix. The matrix object is denoted by `Mat`.

Before understanding the use of matrix in PETSc, one must know how the ISIS-CFD sents the system to the solver to solve. In each iteration, ISIS-CFD creates the system of matrix in compressed sparse row format. It actually creates three set of arrays which are:

- The array `a` contains the values of the matrix
- The array `IndCon_CC` contains the local column indices of the values in `"a"`

• The array `IpntCF_CC` contains the location pointing into `IndCon_CC`, where to begin a new row.
These information can be passed to the PETSc to build the Matrix and by using this matrix and the `src` term one can solve the system with PETSc.

The simplest routine to create a matrix is `MatCreate()` which can be followed by `MatSetSizes()` to set the size and `MatSetType()` to set the type of the matrix (sequential or parallel). One can also add `MatMPIAIJSetPreallocation()` (for parallel matrix) to preallocate the nonzero terms of the matrix (for sequential matrix `MatSeqAIJSetPreallocation()`). The commands can be written as follows:

```
call MatCreate(PETSC_COMM_WORLD,Mat A,ierr)
call MatSetSizes(Mat A,PetscInt m,PetscInt n,PetscInt M,PetscInt N,ierr)
call MatSetType(Mat A,MatType matype,ierr)
call MatMPIAIJSetPreallocation(Mat A,PetscInt d_nz,PetscInt d_nnz[],PetscInt
o_nz,PetscInt o_nnz[],ierr)
```

where `A` is a matrix. In `MatSetSizes()`, `m` and `n` are the number of local rows and columns and `M` and `N` are the number of global rows and columns of the matrix respectively. One can either set the number of local rows and columns and let the PETSc determine the number of global rows and columns by using `PETSC_DETERMINE` instead of `M` and `N`; or set the number of global rows and columns and let the PETSc decide about the local ones by using `PETSC_DECIDE` instead of `m` and `n`. The both approaches can be used for parallel matrix but for sequential matrix the first one is preferable. The `MatSetType()` is used to set the type of the matrix. The `matype` can be set as `MATMPIAIJ` (for parallel matrix) or as `MATSEQAIJ` (for sequential matrix). In `MatMPIAIJSetPreallocation()` routine, `d_nz` is the number of nonzeros per row in DIAGONAL portion of local submatrix, `d_nnz` is the array containing the number of nonzeros in the various rows of the DIAGONAL portion of the local submatrix (possibly different for each row) or `PETSC_NULL`, `o_nz` is the number of nonzeros per row in the OFF-DIAGONAL portion of local submatrix and `o_nnz` is the array containing the number of nonzeros in the various rows of the OFF-DIAGONAL portion of the local submatrix (possibly different for each row) or `PETSC_NULL`.

One can also use `MatCreateMPIAIJ()` which can do the same thing as the four routines described above. It can be called as:

```
call MatCreateMPIAIJ(PETSC_COMM_WORLD,PetscInt m,PetscInt n,PetscInt M,PetscInt
N,PetscInt d_nz,const PetscInt d_nnz[],PetscInt o_nz,const PetscInt o_nnz[],Mat
A,ierr)
```

To assign values into a matrix, one can use `MatSetValues()` which inserts or adds a block of values into a matrix. The command can be written as:

```
call MatSetValues(Mat mat,PetscInt m,PetscInt idxm[],PetscInt n,PetscInt
idxn[],PetscScalar values[],InsertMode addv,ierr)
```

This routine inserts or adds a logically dense subblock of dimension $m \times n$ into the matrix. The integer indices `idxm` and `idxn`, respectively, indicate the global row and column numbers to be inserted. `MatSetValues()` uses the standard C convention, where the row and column matrix indices begin with zero regardless of the storage format employed. The array `values` is logically two-dimensional, containing the values that are to be inserted. After the matrix elements have been inserted or added into the matrix, they must be processed (also called assembled) before they can be used. The routines for matrix processing are:

```
call MatAssemblyBegin(Mat A,MAT FINAL ASSEMBLY,ierr)
call MatAssemblyEnd(Mat A,MAT FINAL ASSEMBLY,ierr)
```

There are some other routines which speacially deals with the CSR matrix format. For example, `MatMPIAIJSetPreallocationCSR()` can be used for the preallocation of the nonzeros as well as to assign values of the matrix by using the CSR informations. The command can be written as:

```
call MatMPIAIJSetPreallocationCSR(Mat A,PetscInt irow[],PetscInt jcol[],
PetscScalar v[],ierr)
```

where `jcol` is the column indices (starts with zero) for each local row (same as IndCon_CC), `irow` the indices (starts with zero) into jcol for the start of each local row (same as IpntCF_CC) and `v` is the array of values of the matrix (same as `a`). Before calling this routine, one should call `MatCeate()`, `MatSetSize()` and `MatSetType()`; and this routine doesn't require the assemble routine, `MatAssemblyBegin()` and `MatAssemblyEnd()`.

Another routine for CSR format is `MatCreateMPIAIJWithArrays()` (for sequential matrix `MatCreateSeqAIJWithArrays()`), which is one the best routines for matrix because by only one routine one can create MPI matrix, set sizes, and assign values by using the arrays from CSR format. It also doesn't need assymbly routines (same as the previous one). So only one line is sufficient to complete the matrix initialization which is very efficient as it requires less step and less computational time. That's why, it is extensively used for the research. This routine can be called as:

```
MatCreateMPIAIJWithArrays(PETSC_COMM_WORLD,PetscInt m,PetscInt n,PetscInt M,
PetscInt N,PetscInt irow[],PetscInt jcol[],PetscScalar v[],Mat A,ierr)
```

where the notations are same as described for the previous routines.

`MatCreateMPIAIJWithSplitArrays()` is another routine which also deals with the CSR format. But this routine requires the arrays (a, IpntCF_CC and IndCon_CC) to be splited

Function Name	Operation
MatAXPY(Mat Y,PetscScalar a,Mat X,MatStructure,ierr)	$Y = Y + a * X$
MatMult(Mat A,Vec x,Vec y,ierr)	$y = A * x$
MatMultAdd(Mat A,Vec x, Vec y,Vec z,ierr)	$z = y + A * x$
MatMultTranspose(Mat A,Vec x, Vec y,ierr)	$y = A^T * x$
MatMultTransposeAdd(Mat A,Vec x, Vec y,Vec z,ierr)	$z = y + A^T * x$
MatScale(Mat A,PetscScalar a,ierr)	$A = a * A$
MatTranspose(Mat A,MatReuse,Mat B,ierr)	$B = A^T$
MatGetDiagonal(Mat A,Vec x,ierr)	$x = diag(A)$
MatNorm(Mat A,NormType type,double r,ierr)	$r =\| A \|_{type}$

TABLE 3.2: PETSc Matrix Operation

into diagonal and off-diagonal parts which makes the code bigger and time consuming. Also there are certain packages in PETSc (e.g. Hypre, etc.) that doesn't work with this routine and cannot be used for single processor.

One can print a matrix (sequential or parallel) to the screen with the command:

```
call MatView(Mat mat,PETSC_VIEWER_STDOUT_WORLD,ierr)
```

When a matrix is no longer needed, it should be destroyed by:

```
call MatDestroy(A,ierr)
```

3.2.8 KSP solver and Preconditioner

The object KSP is the heart of PETSc, because it provides uniform and efficient access to all of the packages linear system solvers, including parallel and sequential, direct and iterative. KSP is intended for solving nonsingular systems of the form

$$Ax = b \tag{3.1}$$

where A denotes the matrix representation of a linear operator, b is the right-hand-side vector, and x is the solution vector.

To solve a linear system with KSP, one must first create a solver context with the command

```
call KSPCreate(PETSC_COMM_WORLD,KSP ksp,ierr)
```

Here comm is the MPI communicator, and ksp is the newly formed solver context. Before actually solving a linear system with KSP, the user must call the following routine to set the matrices associated with the linear system:

```
call KSPSetOperators(KSP ksp,Mat Amat,Mat Pmat,MatStructure flag,ierr)
```

The argument Amat, representing the matrix that defines the linear system, is a symbolic place holder for any kind of matrix. In particular, KSP does support matrix-free methods. Typically the preconditioning matrix (i.e., the matrix from which the preconditioner is to be constructed), Pmat, is the same as the matrix that defines the linear system, Amat; however, occasionally these matrices differ. The argument flag can be used to eliminate unnecessary work when repeatedly solving linear systems of the same size with the same preconditioning method; when solving just one linear system, this flag is ignored. The user can set flag as follows:

• `SAME_NONZERO_PATTERN` - the preconditioning matrix has the same nonzero structure during successive linear solves,
• `DIFFERENT_NONZERO_PATTERN` - the preconditioning matrix does not have the same nonzero structure during successive linear solves,
• `SAME_PRECONDITIONER` - the preconditioner matrix is identical to that of the previous linear solve.

By default the PETSc uses GMRES method as KSP and ILU(0) as preconditioner (for single processor). If the computation runs in multi-processors the default preconditioner will be BLOCK JACOBI method (with one block per processor, each of them will be solved with ILU(0)). However one can change the default methods of the KSP as well as the preconditioner. To set any preconditioner within the code, the PETSc provides a routine which extracts the PC context:

```
call KSPGetPC(KSP ksp,PC pc,ierr)
```

where `ksp` is the Krylov Subspace context (which is created earlier by using `KSPCreate()`) and `pc` is the preconditioner context which will be extracted by this routine. Now to specify the particular preconditioning method, the user can either select it from the option database using the input of the form `-pc_type <methodname>` or set the method in the code by writing the command:

```
call PCSetType(PC pc,PCType type,ierr)
```

The PC types supported by PETSc is given in the table. For some external packages like HYPRE, the type can be set by using the routine `PCHYPRESetType()`, where the default type for this package is `boomeramg`, the Algebric Multigrid Method.

The `KSPSetTolerances()` routine can be used to set the relative, absolute, divergence tolerance and maximum iteration used by the default KSP convergence testers.

```
call KSPSetTolerances(KSP ksp,PetscReal rtol,PetscReal abstol,PetscReal dtol,
PetscInt maxits,ierr)
```

where `ksp` is the Krylov subspace context, `rtol` is the relative convergence tolerance (relative decrease in the residual norm), `abstol` is the absolute convergence tolerance (absolute size of the residual norm), `dtol` is the divergence tolerance (amount residual can increase before `KSPDefaultConverged()` concludes that the method is diverging) and `maxits` is the maximum number of iterations to use. The `PETSC_DEFAULT_DOUBLE_PRECISION` can be used to retain the default values of any tolerance.

While solving the system, sometimes it is needed to print and observe the residual norm. PETSc has options by which one can print true residual norm as well as preconditioned norm. The option for printing both residual is `-ksp_monitor_true_residual`. If one wants to print only preconditioned residual, can use `-ksp_monitor`. The command to specify these can be written as:

```
call PetscOptionsSetValue('-ksp_monitor_true_residual','monitor.dat',ierr)
```

where 'monitor.dat' is the file where the residuals will be printed.

`KSPSetType()` is the routine which is used to set the Krylov Subspace context. The PETSc soupported KSP type are listed in the Table. To set the default options for other KSP options (except those whice are already defined before) one can call `KSPSetFromOptions()`. To solve the linear system, `KSPSolve()` can be called finally. All these three routine can be written as:

```
call KSPSetType(KSP ksp,KSPType type,ierr)
call KSPSetFromOptions(KSP ksp,ierr)
call KSPSolve(KSP ksp,Vec b,Vec x,ierr)
```

where the vector `'b'` is the right hand side of the system and vector `'x'` is the solution vector which can be created as empty vector or assigning values as initial value for the system. If the vector is kept empty, the PETSc will solve the system with zero initial value.

To print the KSP information and data structure used in solving the system, `KSPView()` can be used. Once the KSP context is no longer needed, it should be destroyed with the routine `KSPDestroy()`.

```
call KSPView(ksp,PETSC_VIEWER_STDOUT_WORLD,ierr)
call KSPDestroy(ksp,ierr)
```

PRECONDITIONER					
Algorithm	**PCType**	**Matrix Type***	**External Package**	**Parallel**	**Complex**
Jacobi	PCJACOBI	aij, baij, sbaij, dence, mpiaij	–	×	×
Point Block Jacobi	PCPBJACOBI	baij, bs=2,3,4,5	–	×	×
SOR	PCSOR	seqdence, seqaij, seqbaij	–		×
Point Block SOR		seqbaij, bs=2,3,4,5	–		×
Block Jacobi	PCBJACOBI	aij, baij, sbaij, mpiaij	–	×	×
Additive Schwarz	PCASM	aij, baij, sbaij, mpiaij	–	×	×
ILU(k)	PCILU/PCICC	seqaij, seqbaij	–		×
ICC(k)		seqaij, seqbaij	–		×
ILU dt		seqaij	Sparsekit		
ILU(0)/ICC(0)		aij, mpiaij	BlockSolve95	×	
ILU(k)	PCHYPRE	aij, mpiaij	Euclid/HyPre	×	
ILU dt		aij, mpiaij	Pilut/HyPre	×	
Matrix-free	PCSHELL			×	×
Multigrid/ infrastructure	PCMG			×	×
Multigrid/ geometric structured grid	DMMG			×	×
Multigrid Algebric	PCHYPRE	aij, mpiaij	BoomerAMG/ Hypre	×	
	PCML	aij, mpiaij	ML/Trilinos	×	
Approximate Inverses	PCHYPRE	aij, mpiaij	Parasails/ Hypre	×	
	PCSPAI	aij, mpiaij	SPAI	×	
Balancing Neumann-Neumann	PCNN	is		×	×
DIRECT SOLVER					
LU	PCLU	seqaij, seqbaij			×
		seqaij	MATLAB		×
		aij, mpiaij	Spooles	×	×
		aij, mpiaij	PastuiX	×	×
		aij, mpiaij	SuperLU, Sequential /Parallel	×	×
		aij, mpiaij	MUMPS	×	×
		seqaij	ESSL		
		seqaij	UMFPACK		
		dence	PLAPACK	×	×
Cholesky	PCCHOLESKY	seqaij, seqbaij			×
		sbaij	Spooles	×	×
		sbaij	PastuiX	×	×
		sbaij	MUMPS	×	×
		seqsbaij	DSCPACK	×	
		dense	PLAPACK	×	×
		matlab	MATLAB		
		aij, mpiaij		×	
QR		matlab	MATLAB		
XXt and XYt		aij, mpiaij		×	

Table 3.3: Preconditioner in PETSc

*Matrix Type

aij - Sparse matrix.

bij - Block sparse matrix.

sbaij - Symmetric block sparse matrix.

seqaij - Sequential sparse matrix.

mpiaij - Parallel sparse matrix.

seqbaij - Sequential block sparse matrix.

seqsbaij - Sequential symmetric block sparse matrix.

dense - Dense matrix.

seqdense - Sequential dense matrix.

is - A matrix type to be used for using the Neumann-Neumann type preconditioners.

Krylov Subspace Method	**KSP Type in PETSc**
Richardson	KSPRICHARDSON
Chebychev	KSPCHEVBYCHEV
Conjugate Gradients	KSPCG
GMRES	KSPGMRES
Bi-CG-Stab	KSPBCGS
Transpose-free Quasi Minimal-Residual	KSPTFQMR
Conjugate Residuals	KSPCR
Conjugate Gradient Squared	KSPCGS
Bi-Conjugate Gradient	KSPBICG
Minimum Residual Method	KSPMINRES
Flexible GMRES	KSPFGMRES
Least Squares Method	KSPLSQR
SYMMLQ	KSPSYMMLQ
LGMRES	KSPLGMRES
Conjugate gradient on the normal equations	KSPCGNE

Table 3.4: Krylov Sybspace Methods in PETSc

3.3 Compile and Run PETSc

3.3.1 Makefile

The directory `$PETSC_DIR/conf` contains virtually all makefile commands and customizations to enable portability across different architectures. Most makefile commands for maintaining the PETSc system are defined in the file `$PETSC_DIR/conf`. These commands, which process all appropriate files within the directory of execution, include:

• lib - Updates the PETSc libraries based on the source code in the directory.

• libfast - Updates the libraries faster. Since libfast recompiles all source files in the directory

at once, rather than individually, this command saves time when many files must be compiled.
• clean - Removes garbage files.

So the most important line in the makefile is the line starting with include:

```
include $PETSC_DIR/conf/base
```

This line includes other makefiles that provide the needed definitions and rules for the particular base PETSc installation (specified by $PETSC_DIR) and architecture (specified by $PETSC_ARCH). The makefiles that is used in this research is given below:

```
RM = /bin/rm

MYSRCS = $(wildcard *.F)
MYOBJS = $(subst .F,.o,$(MYSRCS))

include $PETSC_DIR/conf/base

test_system:  $(MYOBJS) chkopts
-$FLINKER -o test_system $(MYOBJS) $PETSC_KSP_LIB
$RM *.o

include $PETSC_DIR/conf/test
```

3.3.2 Running a PETSc Program

To run a PETSc program in multiprocessor one can write the command as:

```
mpirun -np 2 machinefile machines ./test_system
```

One can also add options at the end of the command. To see the available options one can use `-help` with the previous command.

```
mpirun -np 2 machinefile machines ./test_system -help
```

Chapter 4

Results and Discussion

4.1 Test Case

Generally the linear solver ISIS-CFD uses PCGSTAB (same as BiCGStab) as krylov subspace with ILU(1) preconditioner. But in this research, two types of krylov subspace, BiCGStab and GMRES were used, for both ISIS-CFD and PETSc. In ISIS-CFD solver, only ILU(0) and ILU(1) can be utilized as preconditioner for the strongly coupled system. The PETSc has a number of preconditioners (including external package) most of which can be applied for coupled system. The krylov subspace and preconditioner used in the test are tabulated below:

	KSP	**Preconditioner**
ISIS-CFD	BiCGStab GMRES	ILU(0) ILU(1)
PETSc	BiCGStab GMRES	ILU(0) ILU(1) ILU(2) Block Jacobi Multigrid

Although PETSc has Additive Schwardz method to provide ILU preconditioner, there are some external package like HyPre which also has ILU preconditioners. In this research, the both approaches are used to see the efficiency.

In this research the geometry of the KVLCC2 tanker was used. The coupled system that was used for the test is described below:

• The mesh is a coarse grid mesh with 56800 cells which makes the coupled system with 397600 variables.
• The computation was run in single processor.

• The residual reduction was up to $1e^{-3}$.

• The maximum iteration was fixed to 500.

• There are two systems which were used for the test. One is a linear system after 1st non-linear iteration (lets call it "1st case") and the other is a linear system after 30th non-linear iteration(lets call it "2nd case")

• For the 1st case, zero (0) initial value was assigned and for the 2nd case, results from the 30th non-linear iteration was used as initial value.

In the research, the residual reduction is considered up to $1e-3$ because at the starting of the computation the difference in values (specially pressure) between the boundary and the free stream is large. So, some very first non-linear iterations will not give good results and thats why the residual reduction (of linear system) up to machine accuracy may lead to a difficulty to reach the accurate result at the end of the non-linear iterations. Moreover, it will be a waste of time to go to machine accuracy for every non-linear iteration.

4.2 Tests with BiCGStab

4.2.1 1st Case

The system is tested with BiCGStab krylov subspace and different preconditioners to compare the convergence of ISIS-CFD solver and PETSc. The Fig.(4.1) presents the convergence test of ISIS-CFD and PETSc with ILU(0) preconditioner. In case of PETSc, two types of ILU(0) were used. One is from PETSc itself (Additive Schward method) and other one is from HyPre package(Euclid).From the Fig.(4.1), one can see that all the cases are similarly fluctuating where none of them shows any distinct characteristics.

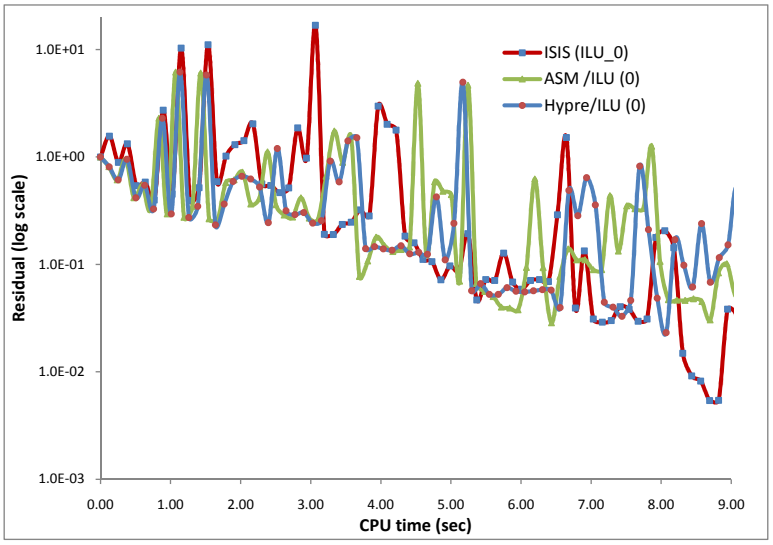

FIGURE 4.1: Convergence of ISIS and PETSc with BiCGStab-ILU(0)

The Fig.(4.2) shows the convergence with ILU(1) preconditioner. In this plot, both ASM and Hypre method are showing better result than the ISIS-CFD. The ASM-ILU(1) seems the fastest among the three although it shows some big fluctuations.

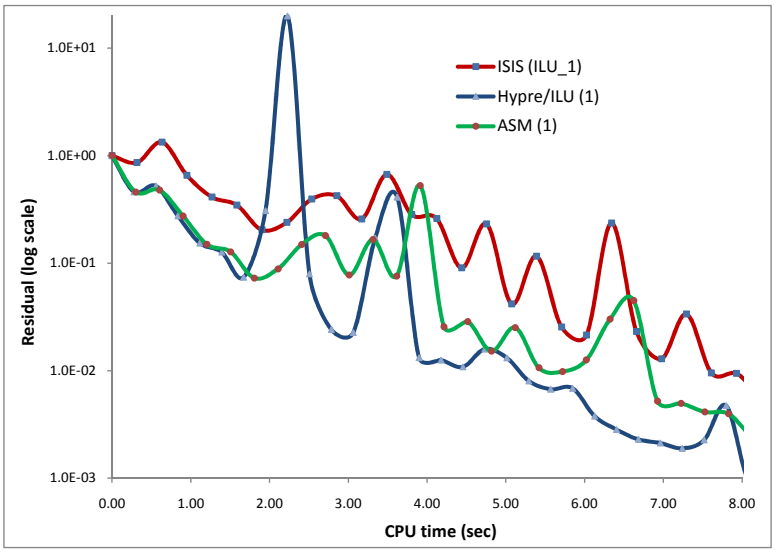

FIGURE 4.2: Convergence of ISIS and PETSc with BiCGStab-ILU(1)

In ISIS-CFD solver, ILU(2) cannot be used as a preconditioner for the coupled system because of the limitation of the code. Thats why, in Fig.(4.3) the convergence of ILU(2) preconditioners of the PETSc were compared with the ILU(1) of the ISIS-CFD. The ASM/ILU(2) and HyPre/ILU(2) are almost same in convergence. But there is a big difference between ISIS-CFD and them, although ISIS-CFD is compared with ILU(1).

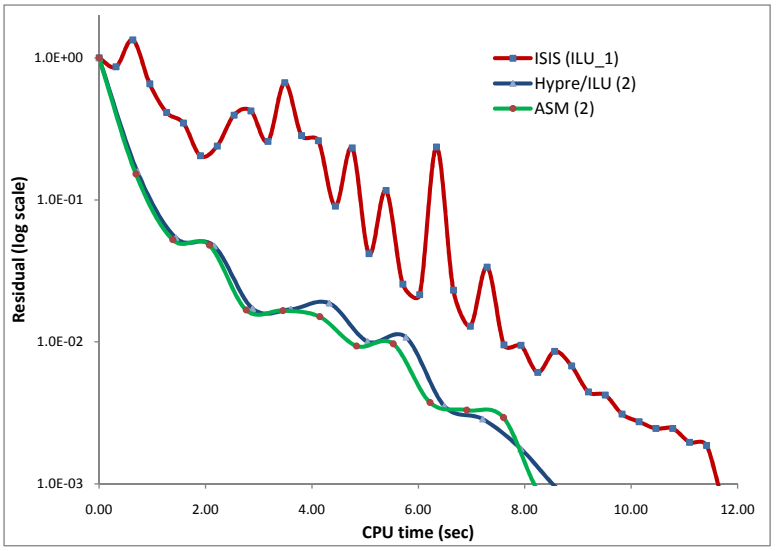

FIGURE 4.3: Convergence of ISIS and PETSc with BiCGStab-ILU(2)

The Block Jacobi preconditioner (PETSc) gives almost similar result to ILU(0) (ISIS-CFD), both of which contain big fluctuations as shown in Fig.(4.4). On the other hand ILU(1) (ISIS-CFD) has less fluctuation compared to the previous two, although the convergence rate is almost same as the previous two cases.

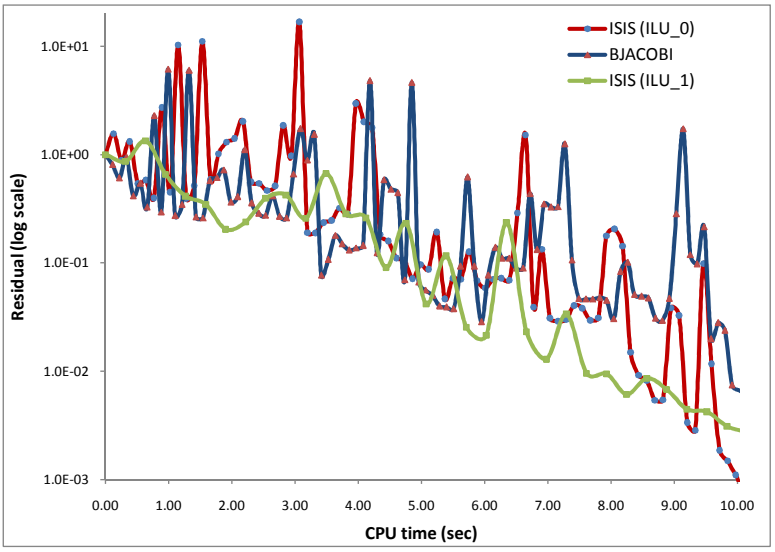

FIGURE 4.4: Convergence of ISIS and PETSc with BiCGStab-Block Jacobi

The Fig.(4.5) presents the convergence of the Multigrid preconditioner from different package of the PETSc and ISIS-CFD with ILU(1) preconditioner. The preconditioner MG is from PETSc itself, AMG is the Algebric Multigrid provided by Hypre package and other multigrid method is from Trillion package. From the Fig.(4.5) it can be seen that the multigrid (PETSc)

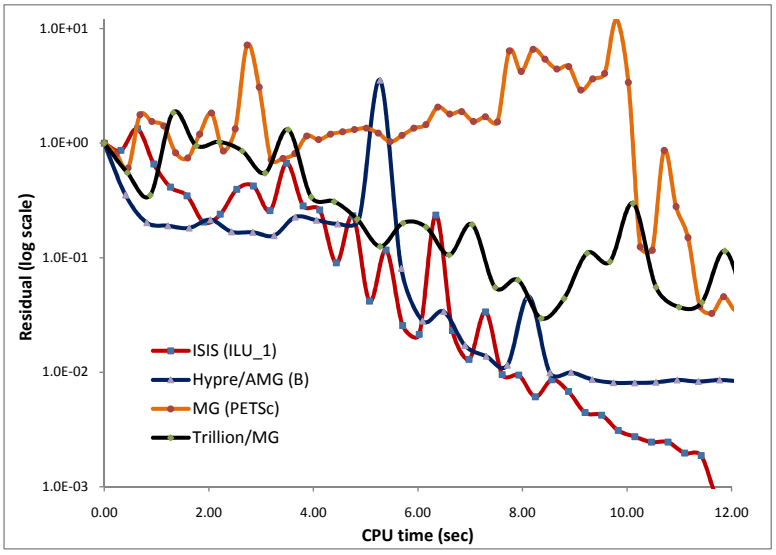

FIGURE 4.5: Convergence of ISIS and PETSc with BiCGStab-Multigrid

is not giving good results with compared to ILU(1) (ISIS-CFD). The reason may be, the

coupled system is a badly-conditioned because of the pressure equation which is difficult to solve and multigrid is quite aggressive method to solve a system.

4.2.2 2nd Case

The system of the 2nd case is more matured as it is taken after the 30th iteration of the non-linear iteration. And in this case the initial value assigned is taken from the result of previous non-linear iteration. In the Fig.(4.6) the comparison in convergence is presented with ILU(0) preconditioner. There is a sharp reduction in residual in the first step for both the solvers

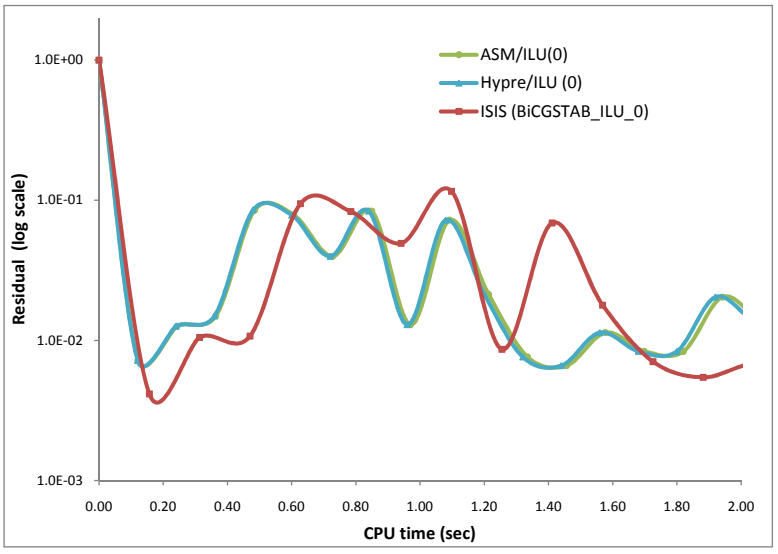

FIGURE 4.6: Convergence of ISIS and PETSc with BiCGStab-ILU(0)

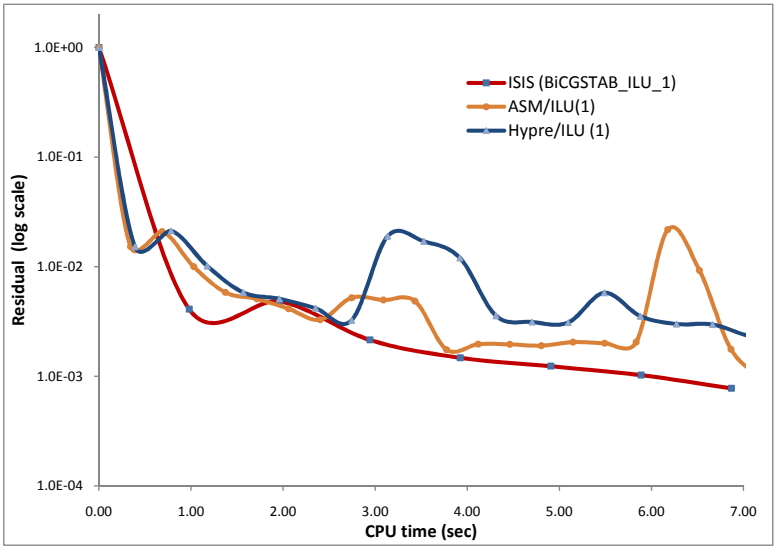

FIGURE 4.7: Convergence of ISIS and PETSc with BiCGStab-ILU(1)

(ISIS-CFD & PETSc) although they are fluctuating at the end. If one considers the residual

reduction up to $1e-2$ surely PETSc shows (both ASM & HyPre) a quick reduction ($0.12sec$) within one step whereas ISIS-CFD finishes the first step with $0.16sec$.

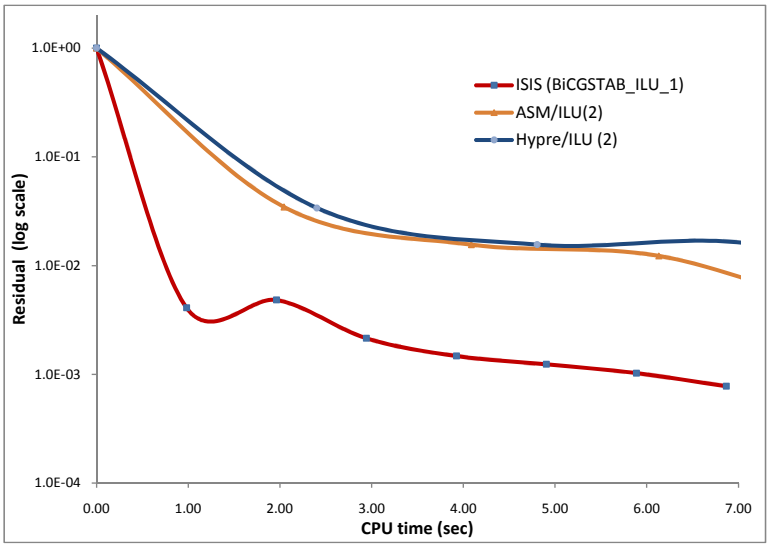

FIGURE 4.8: Convergence of ISIS and PETSc with BiCGStab-ILU(2)

The Fig. (4.7) shows the convergence for ILU(1) preconditioner. Although the overall convergence rate is almost similar for both ISIS-CFD and PETSc, there is a steeper slope at the start for ASM and Hypre compared to ISIS-CFD. So considering the residual up to around $1e-2$, the PETSc solver has faster convergence within the 1st iteration compared to ISIS-CFD.

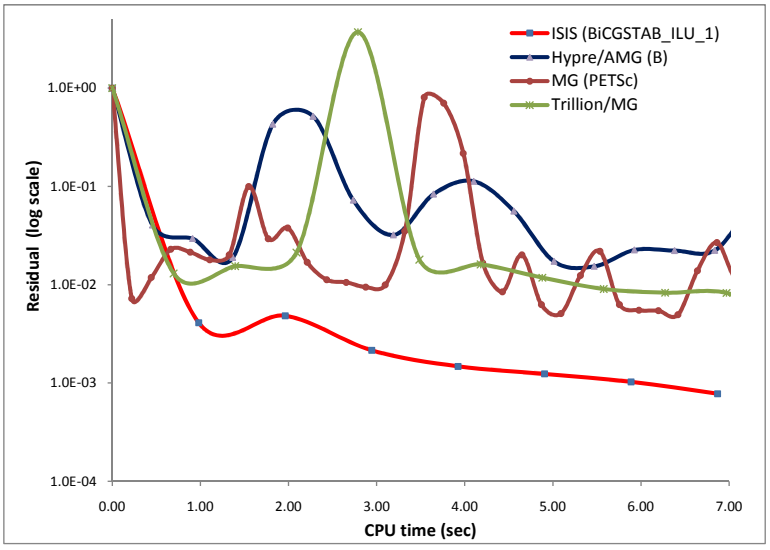

FIGURE 4.9: Convergence of ISIS and PETSc with BiCGStab-Multigrid

The ILU(2) preconditioner has slower rate of convergence with BiCGStab. In the Fig.(4.8), the ASM and Hypre package with ILU(2) is compared with ISIS-CFD where ISIS-CFD is

faster compared to PETSc.

The convergence of Multigrid preconditioner is shown in Fig.(4.9). Among the packages, the MG from PETSc itself and the Tillion/MG have comparable results with ILU(1) of ISIS-CFD. By considering the residual up to $1e-2$, the MG (PETSc) is the faster in the first step ($0.22sec$) compared to ISIS-CFD ($0.99sec$). Also the Trilloin/MG has better convergence within residual $1e-2$ ($0.69sec$).

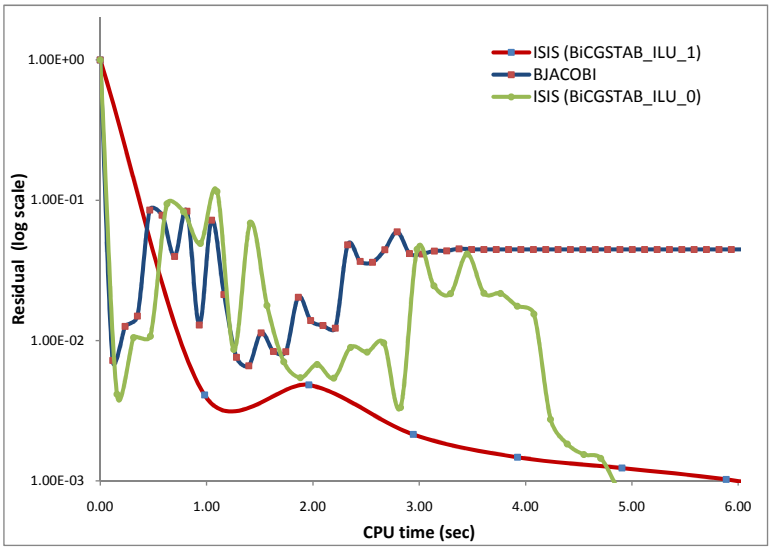

FIGURE 4.10: Convergence of ISIS and PETSc with BiCGStab-Block Jacobi

The Fig.(4.10) shows the convergence of Block Jacobi (PETSc) preconditioner along with ILU(0) and ILU(1) preconditioner from ISIS-CFD. The Block Jacobi and ILU(0) (ISIS-CFD) have almost similar slope in first iteration which is steeper than the ILU(1). Also Block Jacobi is seen to be faster considering the residual limiting up to $1e-2$ in a non-linear iteration.

4.3 Tests with GMRES

4.3.1 1st Case

The GMRES krylov subspace method is also used for the 1st test and 2nd case with the several preconditioners from both ISIS-CFD and PETSc. Here the results from the 1st case are discussed. In the Fig.(4.11), the convergence of the ILU(0) preconditioner are presented for both ISIS-CFD and PETSc with GMRES. Both ASM and Hypre ILU(0) are very similar in convergence rate. They have steeper slope up to around $1e-1$ compared to ISIS-CFD, although later ISIS-CFD became faster than the previous two method.

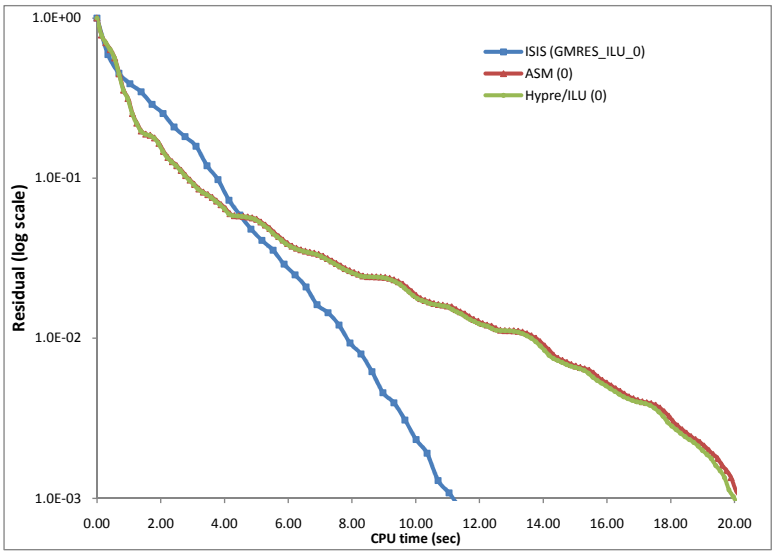

FIGURE 4.11: Convergence of ISIS and PETSc with GMRES-ILU(0)

The ILU(1) preconditioner from PETSc shows an interesting result in Fig.(4.12). The PETSc packages show faster convergence with almost 40% less time compared to ISIS-CFD ILU(1) where Hypre package is the fastest one.

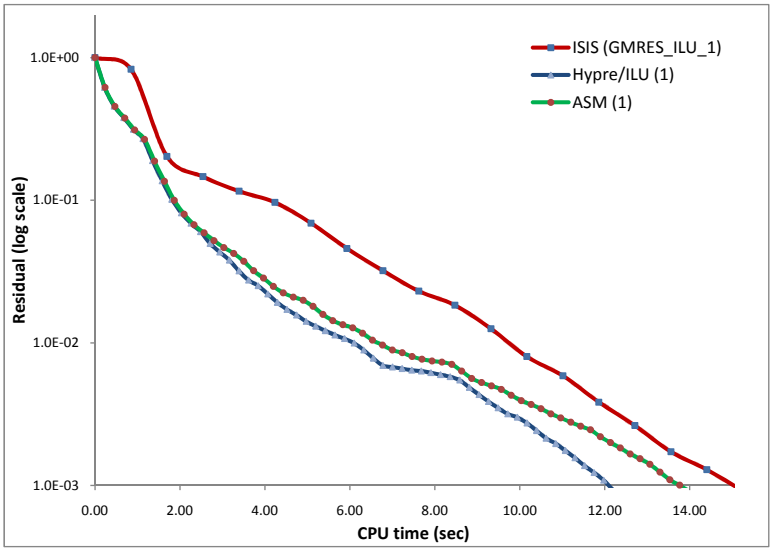

FIGURE 4.12: Convergence of ISIS and PETSc with GMRES-ILU(1)

The Fig.(4.13) shows the convergence of the ILU(2) preconditioner (PETSc) which compared with the ILU(1) of ISIS-CFD (because ILU(2) cannot be run in ISIS-CFD). Here also the PETSc packages are almost 60% faster than the ISIS-CFD.

The convergence rate of Multigrid preconditioner (from PETSc) is almost same as the ISIS-CFD ILU(1) except the Hypre/AMG and MG method. The MG from PETSc is showing very slow convergence. But Hypre/AMG Multigrid method is quite faster than ISIS-CFD

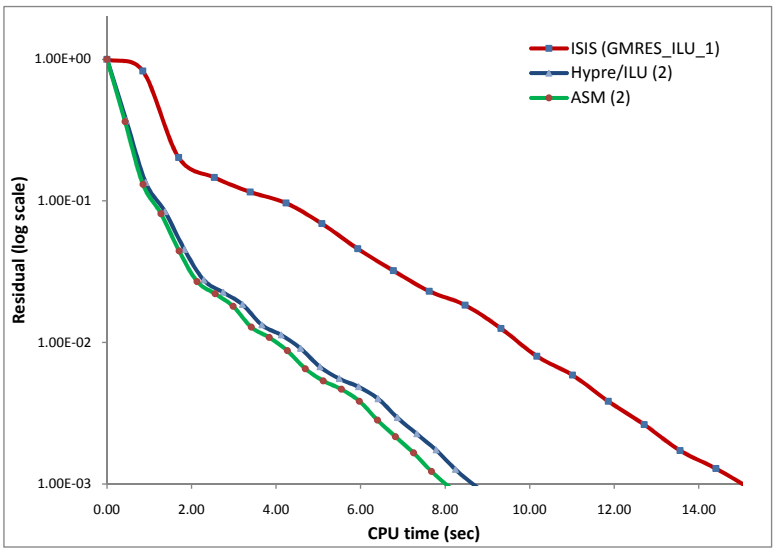

FIGURE 4.13: Convergence of ISIS and PETSc with GMRES-ILU(2)

within the residual reduction up to $1e-2$ and it requires 40% less time to reach this level compared to ISIS-CFD.

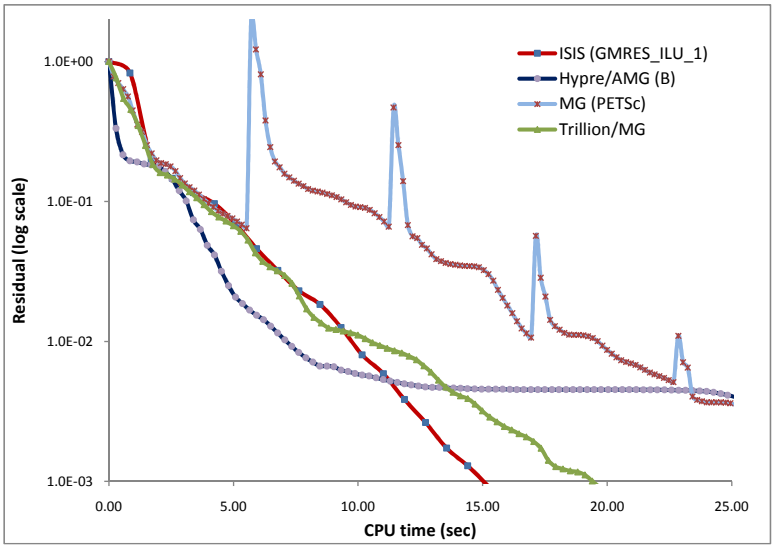

FIGURE 4.14: Convergence of ISIS and PETSc with GMRES-Multigrid

In the Fig.(4.15), the Block Jacobi preconditioner (PETSc) is presented with the ILU(0) and ILU(1) of ISIS-CFD. The Block Jacobi is a little bit faster up to $1e-1$ (approx.) compared to others but ISIS-CFD ILU(0) is the fastest at the end among the others.

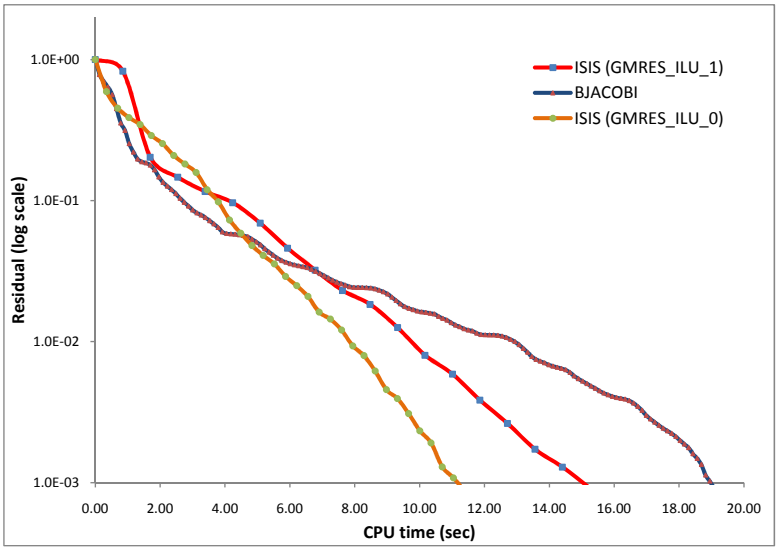

FIGURE 4.15: Convergence of ISIS and PETSc with GMRES-Block Jacobi

4.3.2 2nd Case

The 2nd case is also tested with the GMRES Krylov subspace method with the different preconditioners. The Fig.(4.16) presents the convergence of the ILU(0) preconditioner of both ISIS-CFD and PETSc. The results of the ASM and Hypre are same as they cannot be seen seperated. Also both have steeper slope for the very 1st iteration ($0.10sec$) compared to ISIS-CFD ($0.45sec$) to reach the residual reduction up to $1e-2$.

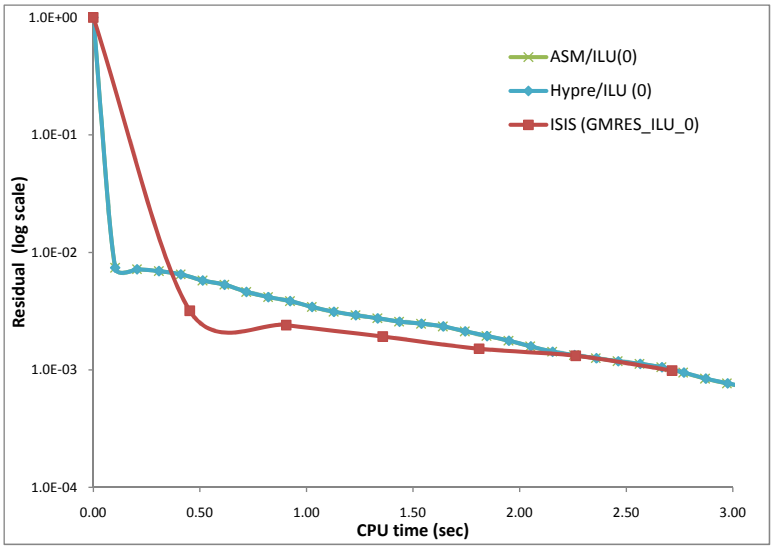

FIGURE 4.16: Convergence of ISIS and PETSc with GMRES-ILU(0)

The GMRES-ILU(1) method showed very good convergence for the 1st case. In the second case also better slope compared to ISIS-CFD up to $1e-2$ as shown in the Fig.(4.17). The

residual reduction reaches up to the $1e-2$ with almost 50% faster rate than ISIS-CFD's first iteration.

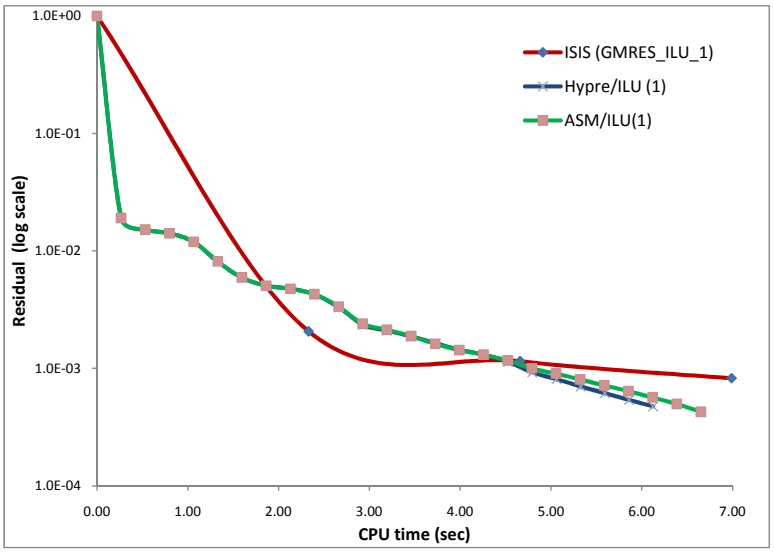

FIGURE 4.17: Convergence of ISIS and PETSc with GMRES-ILU(1)

The ILU(2) preconditioner (PETSc) has different result for this case than the 1st case. In the 1st case ILU(2) of PETSc was faster than the ILU(1) of ISIS-CFD. But in the Fig.(4.18), it can be seen that the ILU(2) of the PETSc packages have very slow convergence comapared to ILU(1) of ISIS-CFD.

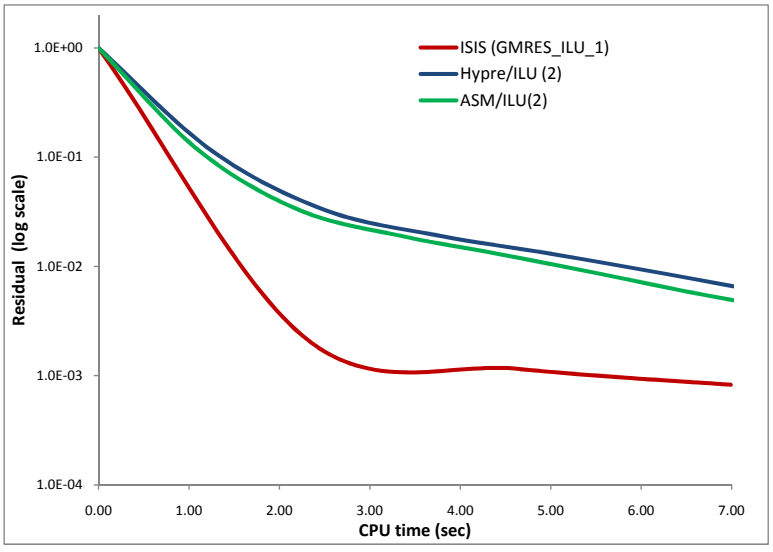

FIGURE 4.18: Convergence of ISIS and PETSc with GMRES-ILU(2)

The Fig.(4.19) shows the convergence of the Multigrid preconditioner from PETSc compared with ILU(1) of ISIS-CFD. As the residual reduction is considered up to $1e-2$ for non-linear iteration,among the PETSc packages, MG (from PETSc itself) and Trillion/MG shows

promising result compared to ISIS-CFD. The ILU(1) of ISIS-CFD takes $2.33sec$ to reduce the residual below $1e-2$ where MG takes $0.16sec$ and Trillion/MG takes $0.44sec$ which indicates faster convergence compared to ISIS-CFD.

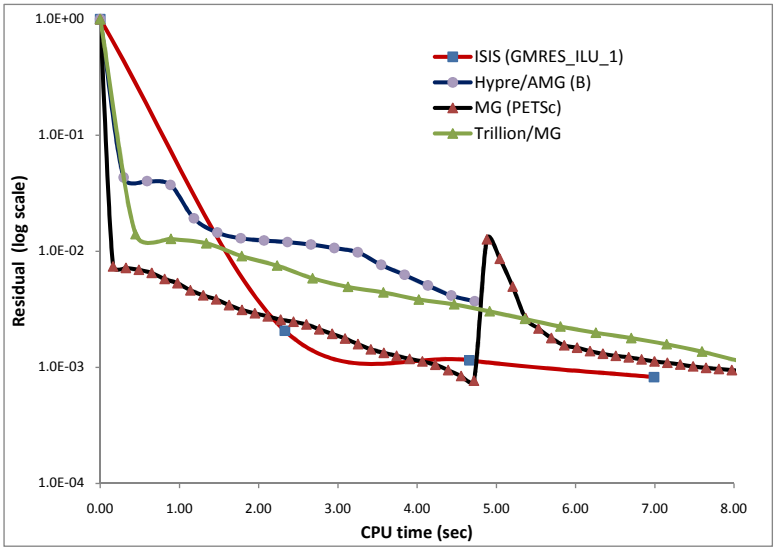

FIGURE 4.19: Convergence of ISIS and PETSc with GMRES-Multigrid

The Block Jacobi shows better convergence rate for the 2nd case than the 1st one. In the Fig.(4.20), the Block Jacobi (PETSc) is compared with ILU(0) and ILU(1) of ISIS-CFD where it shows sharp reduction of residual within 1st iteration which goes below $1e-2$ within $0.09sec$ where ILU(0) takes $0.45sec$ and ILU(1) takes $2.33sec$. So Block Jacobi seems good for more matured system like the 2nd case where the system is taken after 30th non-linear iteration.

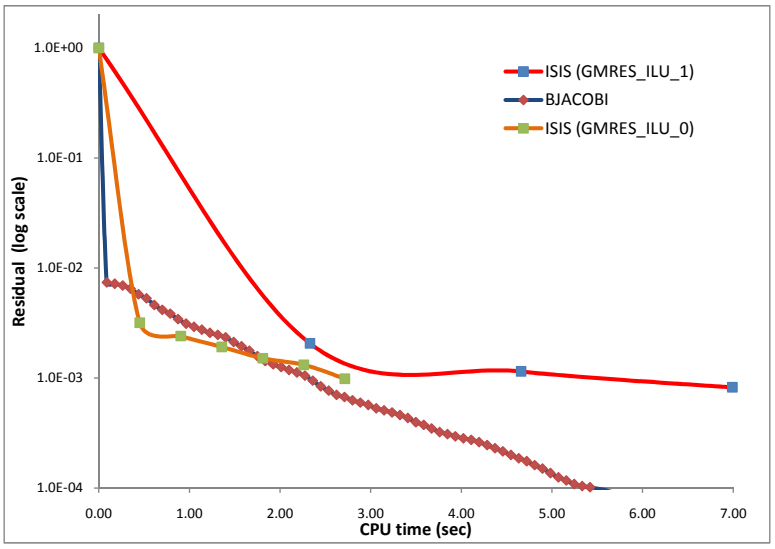

FIGURE 4.20: Convergence of ISIS and PETSc with GMRES-Block Jacobi

4.4 Comparison

From the previous sections one has seen that some of the PETSc preconditioners have shown better results compared to ISIS-CFD. Here some of those good results from BiCGStab and GMRES will be compared together. As the 2nd case is more reliable and mature system, only this case will be considered here to compare between the KSPs.

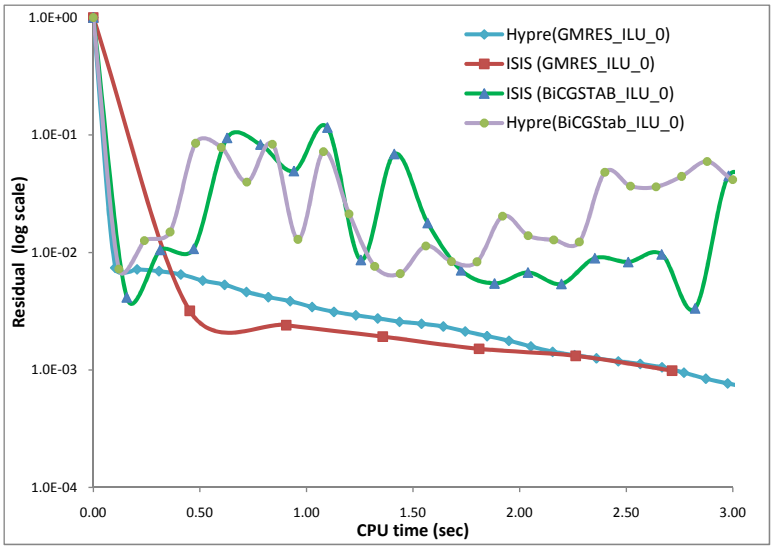

FIGURE 4.21: Convergence of ILU(0) preconditioner with GMRES and BiCGStab

The convergence ILU(0) preconditioner with GMRES and BiCGStab is compared in Fig.(4.21). The Hypre/ILU(0) is used from the PETSc as it has similar convergence rate (most cases) as ASM. Also in some cases it is better than ASM. The Hypre/ILU(0) with both BiCGStab

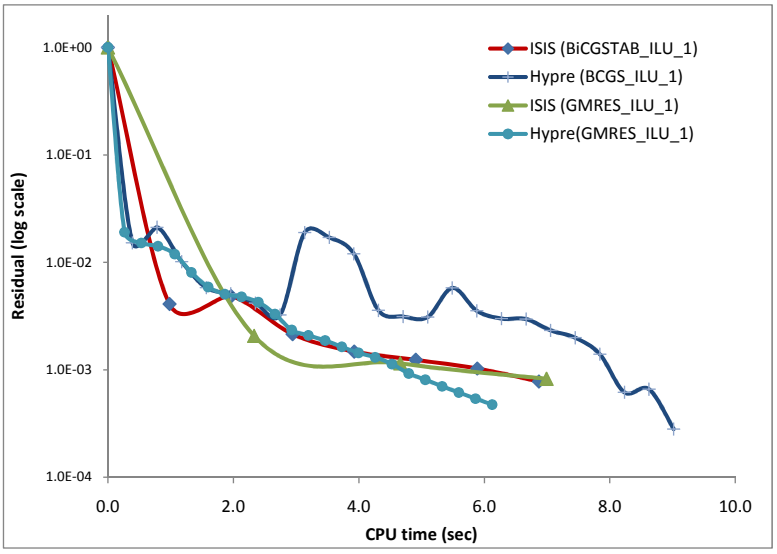

FIGURE 4.22: Convergence of ILU(1) preconditioner with GMRES and BiCGStab

and GMRES is showing same slope of residual reduction for 1st iteration $(1e-2)$ which is

similar to ISIS-BiCGStab where ISIS-GMRES is a bit slower than others. The plot from BiCGStab krylov space have fluctuations for both ISIS and PETSc cases where the GMRES cases have smoother reduction of residuals.

The Hypre/ILU(1) has similar slope for both KSPs as shown in (4.22). And it has better convergence within first few iterations with both KSPs comapared to ISIS-CFD.

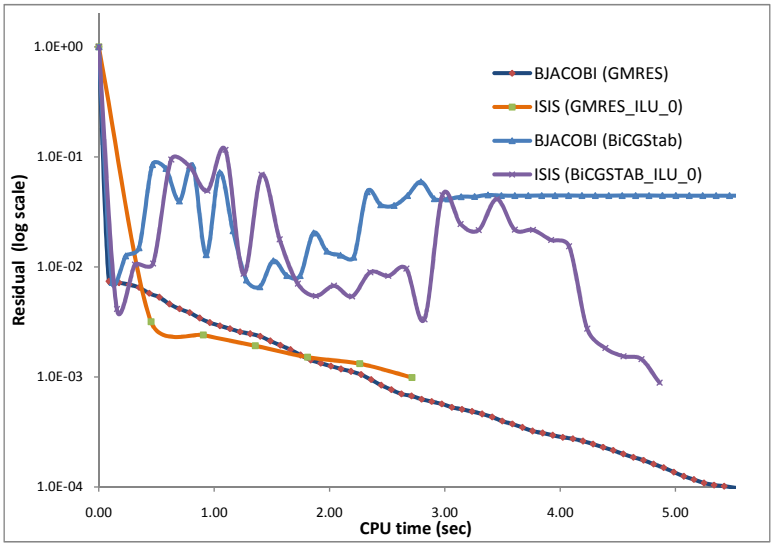

FIGURE 4.23: Convergence of BJACOBI preconditioner with GMRES and BiCGStab

The Block Jacobi has faster convergence slope with both GMRES and BiCGStab at the starting (Fig(4.23)) which is similar to ISIS-BiCGStab/ILU(0) where the ISIS-GMRES/ILU(0) has slower convergence. The BJACOBI with GMRES has smoother plot where plot from BiCGStab (ISIS and PETSc) are not consistant.

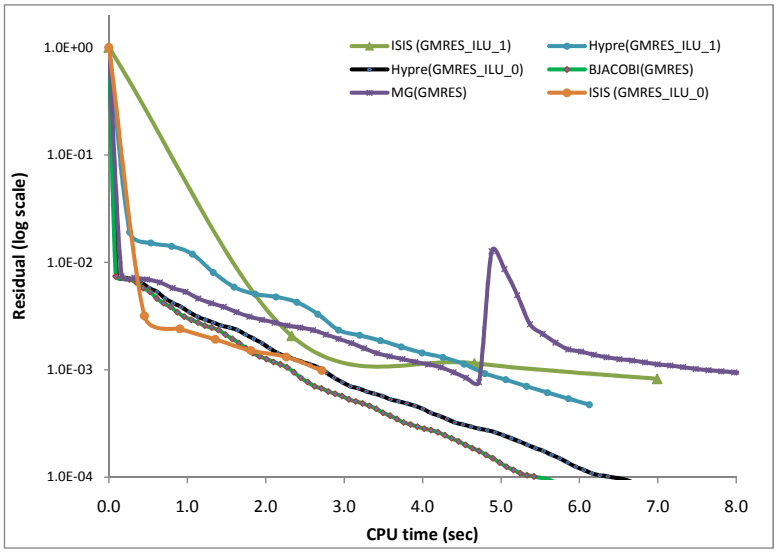

FIGURE 4.24: Convergence of some of the best preconditioners with GMRES (case 2)

Some of the preconditioners which have shown better results so far, have been presented all together in Fig.(4.24). Here only GMRES Krylov subspace is considered as it has smoother convergence. Among the preconditioners presented in the figure, the Hypre/ILU(0), BJACOBI and MG has almost similar slope up to $1e-2$ which are the fastesr than others. The Hypre/ILU(1) had similar slope as the ISIS-ILU(0) but it could not reduce the residual up to $1e-2$.

4.5 Memory Usage

In PETSc, one can see the memory usage information by using runtime option database key "`-memory_info`" which prints the current and maximum memory information at the end of run. The `PetscMemoryGetCurrentUsage()` or `PetscMemorySetGetMaximumUsage()` and `PetscMemoryGetMaximumUsage()` can also be used in the code to see the memory info.

The Table(4.1) shows the memory usage of PETSc and ISIS for different KSP and preconditioner. From the table one can conclude following points:

• For any case (case 1 or 2) the memory usage is almost same for the same KSP and same preconditioner. So, memory usage does not depend on the cases or systems as long as the structure is similar and number of variable is same.
• The memory usage of ISIS-CFD is farely constant for different KSP as long as preconditioner is same.

Table 4.1: Memory Usage in PETSc

Solver	**1st Case**		**2nd Case**	
	BiCGStab	**GMRES**	**BiCGStab**	**GMRES**
ISIS/ILU(0)	134 Mb	140 Mb	134 Mb	140 Mb
PETSc/ASM ILU(0)	306 Mb	395 Mb	309 Mb	398 Mb
PETSc/Hypre ILU(0)	311 Mb	400 Mb	314 Mb	404 Mb
BJACOBI	244 Mb	333 Mb	247 Mb	336 Mb
ISIS/ILU(1)	225 Mb	231 Mb	197 Mb	231 Mb
PETSc/ASM ILU(1)	398 Mb	487 Mb	401 Mb	490 Mb
PETSc/Hypre ILU(1)	437 Mb	526 Mb	440 Mb	529 Mb
PETSc/ASM ILU(2)	753 Mb	842 Mb	756 Mb	784 Mb
PETSc/Hypre ILU(2)	904 Mb	993 Mb	907 Mb	936 Mb
PETSc/Hypre AMG	273 Mb	363 Mb	276 Mb	337 Mb
PETSc/MG	260 Mb	349 Mb	263 Mb	352 Mb
PETSc/Trillion(MG)	519 Mb	608 Mb	522 Mb	611 Mb

• In case of PETSc, BiCGStab has less memory usage than the GMRES for any specific preconditioner.
• For any specific preconditioner the ISIS-CFD has less memory consumption than PETSc.

• The memory usage increases with the ILU level for ILU preconditioner (ILU 0, 1 and 2) of both PETSc and ISIS-CFD. The Hypre/ILU(2) has the highest memory consumption among all the preconditioners used here.

• In case of PETSc ILU preconditioners, the ASM has lower memory consumption than the Hypre package.

• Among the PETSc Multigrid preconditioners, the MG and and the Hypre BoomerAMG have reasonable memory usage specially with BiCGStab krylov subspace. Trillon/MG has the highest memory consumption among the multigrid preconditioner.

Chapter 5

Conclusion

The strongly coupled system is a huge, unsymmetric and badly conditioned system. It is very difficult to solve specially at the starting of the non-linear iteration because of initial and boundary conditions. Thats why, the 1st case which is taken after the 1st non-linear iteration, creates more fluctuations while solving than the 2nd case (taken after 30th non-linear iteration). So the 2nd case is more mature and the result is more reliable than the 1st one.

Among the KSP used in this research, the results from the GMRES are less fluctuating and smoother than BiCGStab. This is because GMRES method approaches by approximating the solution by the vector in a Krylov subspace with minimal residual. So the GMRES is more preferable than the BiCGStab for stable convergence although the memory usage of GMRES is more than BiCGStab (but memory is cheap!).

The residual was not reduced to machine accuracy but $1e-3$. The results are compared by considering the residual reduction up to $1e-2$. The reason is, as written in the first paragraph, because of the ill-conditioned system and its initial and boundary conditions, the solutions are not as reliable as to reduce the residual to machine accuracy (specially for first few iterations). Because it may lead the solutions far from convergence with non-linear iterations.

Among the ILU preconditioners of PETSc, the ILU(1) shows a better convergence than ISIS-CFD which is fairly consistant for cases and KSP methods. Most cases of ASM/ILU(1) and Hypre/ILU(1) have similar results whereas some cases Hypre/ILU(1) is better, although Hypre demands more memory than ASM. ILU(1) of PETSc can be preferably used with GMRES krylov subspace as it has less fluctuations and smooth reduction of residual than BiCGStab. In some cases, ILU(0) and BJACOBI has good results specially for 2nd case

with GMRES krylov subspace although non-linear iteration is needed to be checked for these methods.

Among the multigrid preconditioners, MG of PETSc and Trillion/MG have interesting results with GMRES. They have shown better convergence than ISIS for 2nd case (Fig.4.19). Although in 1st case, MG has slower convergence and Trillion has similar to ISIS/ILU(1) (Fig.4.14), the 2nd case is more reliable than the 1st one.

Chapter 6

Further Recommendation

To carry out the research further, the following points can be considered for future work:

- In this research only one grid (coarse mesh) was tested thoughout. One can go further by analyzing different grids (coarse, medium, fine, very fine, etc.) of the same geometry to see the change in convergence of methods for changing grids.
- The main PETSc code is written as functions so that it can be integrated readily in ISIS. By using this subroutine the PETSc can be used throughout the non-linear iterations and convergence results can be compared with ISIS-CFD solver.
- Multi-block analysis can be carried out and parallel computation can be compared between ISIS and PETSc. To facilitate parallel computation, the code is written closer to parallel code as much as possible so that one can use it with minor change.

Appendix A

The PETSc Program

```
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                          MAIN ROUTINE
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      program main_file
      include "precision.h"
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                  Variable declarations
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      integer,dimension (:), allocatable:: IpntCF_CC,IndCon_CC
      double precision,dimension (:), allocatable::a,Src,
     $       Sol,i_value
      integer :: matopt,ivalue,pcopt,nvariable,ndim_mat,maxits,restype
!
      common /pvmmb/me,nproc
      Common /umesg/ imesg
!
      character*5   iluk
      character*10 pctype
      character*10 ksptype
!  Timing variables
      Integer, Parameter :: iprec_single=selected_real_kind(4)
      Integer, Parameter :: iprec_double=selected_real_kind(8)
!
      Real(iprec_single) :: time
      Integer :: itime_start, itime_end, itime_rate, time_max
!
```

```
      me=0
      nproc=1
      imesg=6
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Choose the defining matrix, preconditioner and solver type
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!!!   Select Matrix creation options
!     [1] MatCreate() and MatMPIAIJSetPreallocationCSR()
!     [2] MatCreateMPIAIJWithArrays()
!
      matopt=2
!
!!!   Select Initial value options (assigned through 'Sol' vector)
!     [1] Zero Initial value
!     [2] Assigned initial value
      ivalue=1

!***   Preconditioner options
!     [0] PETSc Preconditioners; PCBJACOBI,PCMG
!     [1] Additive Schwarz Method (PCASM)
!     [2] HYPRE/ILU(K)
!     [3] HYPRE/Multigrid
!     [4] ML/Multigrid
!
      pcopt=2
!
!     if 0 please enter type
      pctype="bjacobi"  !  bjacobi,mg
!
!     If 1,2 please enter the ILU level
      iluk='1'

!***   Enter KSP type
!
      ksptype="gmres" !richardson,gmres,bcgs,cg...
!
!
!  Set the Tolerance (relative/preconditioned residual tol.)
      tol = 1.e-3
```

```
!
!  Set maximum iteration number
      maxits = 500
!
!  To write the ksp residual norm in residual.dat file
!     [1] True residual
!     [2] Preconditioned residual
!
      restype=1
!
! >>> Start timing 1
      Call SYSTEM_CLOCK(COUNT=itime_start, COUNT_RATE=itime_rate,
     $                  COUNT_MAX=time_max)
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                   Extract system information
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!
      open(10,file='Strongly_Coupled_System.bin',status='unknown',
     $       form='unformatted')
      read(10) nvariable
c      Import the connectivity
      allocate(IpntCF_CC(nvariable+1))
      call read_bin_int(nvariable+1,IpntCF_CC)
      ndim_mat=IpntCF_CC(nvariable+1)-1
      allocate(IndCon_CC(ndim_mat),a(ndim_mat))
      call read_bin_int(ndim_mat,IndCon_CC)
c      Import the matrix
      call read_bin_float(ndim_mat,a)
c      Import the right hand side
      allocate(Src(nvariable),Sol(nvariable))
      call read_bin_float(nvariable,Src)
c      Import the solution
      call read_bin_float(nvariable,Sol)
      close(10)
      allocate(i_value(nvariable))
c      Initialization of the system
      do i=1,nvariable
         i_value(i)=Sol(i)
      enddo
```

```
!---------------------------------------------------------------------
!                        Call PETSc routine
!---------------------------------------------------------------------
      call test_system(matopt,ivalue,pcopt,iluk,nvariable,ndim_mat,
     $     IpntCF_CC,IndCon_CC,a,Src,Sol,i_value,tol,pctype,ksptype,
     $     maxits,restype)
!*********************************************************************
!
! <<< Stop timing 1
      Call SYSTEM_CLOCK(itime_end)
! The elapsed time in seconds in PETSc
      time=REAL(itime_end - itime_start)/REAL(itime_rate)
      Print *, 'Elapsed time in PETSc: ',time,'sec'
      end
```

```
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                       PETSc SUBROUTINE
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      subroutine test_system(matopt,ivalue,pcopt,iluk,nvariable,
     $          ndim_mat,IpntCF_CC,IndCon_CC,a,Src,Sol,i_value,tol,
     $          pctype,ksptype,maxits,restype)
!
      Include "precision.h"

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                    Include files
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!
!   petsc.h  - base PETSc routines  petscvec.h - vectors
!   petscmat.h - matrices           petscksp.h - Krylov subspace methods
!   petscpc.h  - preconditioners

#include "finclude/petsc.h"
#include "finclude/petscvec.h"
#include "finclude/petscmat.h"
#include "finclude/petscpc.h"
#include "finclude/petscsys.h"
#include "finclude/petscksp.h"

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                   Variable declarations
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      PC       pc
      KSP      ksp
      Mat      D
      Vec      u,rhs,x,b
      PetscInt N,k,i,j,globalIndRow,globalIndCol,its,maxits,dummy,
     $        Istart,Iend,ii,jj,kk,ll,ojj,okk,kkA,kkB,kkN,pntBegin,
     $        pntEnd,yy,nrow,ncolumn,lg,matopt,pcopt,nz,nvariable
     $        ivalue,count_zero,ndim_mat,Irow,Ipnt,ivar,nn,neig,
     $        restype
      PetscErrorCode ierr
      PetscMPIInt    rank
      PetscScalar    neg_one,Resmax,Res,msr
```

```
      PetscReal      errRHSmax,mem,r,c
      PetscTruth     iflag
      KSPType kspt
      PCType pct
      PetscLogDouble memory

      common /pvmmb/me,nproc
      Common/parallele/mybloc
      common /com/mytid,itids(1000)
      COMMON/STMPI /bloc
      character*4 bloc

      Common /umesg/ imesg
      CHARACTER*150 fname
      character*5   iluk
      character*10 pctype
      character*10 ksptype

      integer,dimension (:), allocatable::nfcom,nblcom,ncell_local,
     $     Local_to_Global_Mapping,LoToGlo,
     $     column,ocolumn,opointer,LoToGloR,col,row
      integer,dimension (:,:), allocatable:: Ind_send,Ind_Receive
      double precision,dimension (:), allocatable::
     $       p,v,ov,aa,ModSrc

      integer,dimension(nvariable+1):: IpntCF_CC
      integer,dimension(ndim_mat):: IndCon_CC
      double precision,dimension(ndim_mat):: a
      double precision,dimension(nvariable):: Src,Sol,i_value
!
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                    Initialize PETSc
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      call PetscInitialize(PETSC_NULL_CHARACTER,ierr)
      call PetscMemorySetGetMaximumUsage(ierr) ! used to get memory info
      call initmb1
*      mybloc=me
!
!     Open files and get the values or send indices and receive indices
```

```
      if(nproc .ne. 1)then
        if(me .eq. 1) then
           write(*,*) 'This is a uniprocessor example only'
        endif
        SETERRQ(1,' ',ierr)
      endif

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!               Establish local to global index mapping
!- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      N=nvariable
      write(imesg,*)'Number of variables: ',N
      allocate(Local_to_Global_Mapping(ndim_mat+1))
      index0=0
      Local_to_Global_Mapping=0
      do i=1,ndim_mat+1
         Local_to_Global_Mapping(i)=index0+i-1
      end do
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                     Create matrix
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!
!   LoToGlo - the global column indices of the array a
!   LoToGloR - the global indices pointing at LoToGlo to start a new row
!
      allocate(LoToGlo(ndim_mat))
      allocate(LoToGloR(N+1))
      LoToGlo=Local_to_Global_Mapping(IndCon_CC)
      LoToGloR=Local_to_Global_Mapping(IpntCF_CC)
!-----------------------------------------------------------------------
      select case (matopt)
!
!     [1] MatCreate() and MatMPIAIJSetPreallocationCSR()
      case (1)
      write(imesg,*)'[1] MatCreate() and MatMPIAIJSetPreallocationCSR()'
      call MatCreate(PETSC_COMM_WORLD,D,ierr)
      call MatSetSizes(D,N,N,PETSC_DETERMINE,PETSC_DETERMINE,ierr)
      call MatSetType(D,MATMPIAIJ,ierr)
      call MatMPIAIJSetPreallocationCSR(D,LoToGloR,LoToGlo,a,ierr)
```

```
!
!     [2] MatCreateMPIAIJWithArrays()
      case (2)
      write(imesg,*) '[2] MatCreateMPIAIJWithArrays()'
      call MatCreateMPIAIJWithArrays(PETSC_COMM_WORLD,N,N,
     $     PETSC_DETERMINE,PETSC_DETERMINE,LoToGloR,LoToGlo,a,D,ierr)
!
      end select
!
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*                 Create and Set values to vectors
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

      call VecCreateMPI(PETSC_COMM_WORLD,N,PETSC_DETERMINE,rhs,ierr)
      call VecDuplicate(rhs,u,ierr) ! u - the approximated solution
      call VecDuplicate(rhs,b,ierr) ! b - the computed RHS from the matrix
      call VecDuplicate(rhs,x,ierr)! x - the exact solution

      call VecSetValues(x,N,Local_to_Global_Mapping,
     $     Sol,INSERT_VALUES,ierr) ! Set the exact solution vector

      call VecAssemblyBegin(x,ierr)
      call VecAssemblyEnd(x,ierr)

!     Set values for the right hand side vector
      call VecSetValues(rhs,N,Local_to_Global_Mapping,
     $     Src,INSERT_VALUES,ierr)
      call VecAssemblyBegin(rhs,ierr)
      call VecAssemblyEnd(rhs,ierr)

      write(imesg,*) 'The vector and matrix values',
     $               ' are set and assembled.'

*     Check if the matrix has been defined correctly
      neg_one=-1.0
      call MatMult(D,x,b,ierr)
      call VecAXPY(b,neg_one,rhs,ierr)
      call VecAbs(b,ierr)
      call VecMax(b,i,errRHSmax,ierr)
```

```
      write(imesg,*) 'rhs-b =',errRHSmax
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*           Create the linear solver and set various options
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

*     Create linear solver context
      call KSPCreate(PETSC_COMM_WORLD,ksp,ierr)

*     Set operators. Here the matrix that defines the linear system
!     also serves as the preconditioning matrix.  Here are matrix D.
      call KSPSetOperators(ksp,D,D,DIFFERENT_NONZERO_PATTERN,ierr)

*     Returns a pointer to the preconditioner context
      call KSPGetPC(ksp,pc,ierr)
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                     Select the Preconditioner
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      select case (pcopt)

      case (0)
* 0   Preconditioner of PETSc which can be used for parallel computing
*     without external package: Jacobi, SOR, Block Jacobi, Additive Schwarz
!
      call PetscOptionsSetValue('-pc_type',pctype,ierr)
      case (1)
* 1  Preconditioner: Additive Schwarz Method
!      By default: subdomain=1, overlab=1, type=restrict, level=0
!      ilu - if want to use icc, set the matrix is symmetric.
!      Use in place is to destroy the matrix after use to save memory
      call PCSetType(pc,PCASM,ierr)
      call PCASMSetUseInPlace(pc,ierr)
      call PetscOptionsSetValue('-sub_pc_factor_levels',iluk,ierr)
      call PetscOptionsSetValue('-sub_pc_factor_shift_positive_definite'
     $     ,PETSC_NULL_CHARACTER,ierr) ! to avoid zero pivot

      case (2,3)
* 2,3   Preconditioner: HYPRE
      call PCSetType(pc,PCHYPRE,ierr)
```

```
      if (pcopt.eq.2) then
* 2   Euclid for ILU(k)
      call PCHYPRESetType(pc,'euclid',ierr)
      call PetscOptionsSetValue('-pc_hypre_euclid_levels',iluk,ierr)
      call PetscOptionsSetValue('-pc_hypre_euclid_bj','TRUE',ierr)

      else
* 3   BoomerAMG for Multigrid
      call PCHYPRESetType(pc,'boomeramg',ierr)
      call PetscOptionsSetValue('-pc_hypre_boomeramg_max_levels',
     $     '10',ierr)
      call PetscOptionsSetValue('-pc_hypre_boomeramg_relax_type_all',
     $     'backward-SOR/Jacobi',ierr)
      endif

      case (4)
* 2    Preconditioner: ML
      call PCSetType(pc,PCML,ierr)
      call PetscOptionsSetValue('-pc_ml_maxNlevels','5',ierr)
      call PetscOptionsSetValue('-mg_coarse_pc_factor_zeropivot',
     $                          '1e-25',ierr)

      end select
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!              Set the relative,absolute,divergence, tolerances,
!                   maximum iteration and KSP solver type
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      call KSPSetTolerances(ksp,tol,PETSC_DEFAULT_DOUBLE_PRECISION,     &
     &     PETSC_DEFAULT_DOUBLE_PRECISION,maxits,ierr)
!
!     To enable the ksp monitoring and write in a file
!
      if(restype .eq. 1) then
      call PetscOptionsSetValue('-ksp_monitor_true_residual',
     $     'residual.dat',ierr)
      elseif(restype .eq. 2) then
      call PetscOptionsSetValue('-ksp_monitor',
     $     'residual.dat',ierr)
      endif
```

```
!
!  Set KSP solver type
      call PetscOptionsSetValue('-ksp_type',ksptype,ierr)
      call KSPSetFromOptions(ksp,ierr)
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                   Solve the linear system
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      if(ivalue .eq. 1) then
          iflag = PETSC_FALSE
      elseif(ivalue .eq. 2) then
          iflag = PETSC_TRUE
      call VecSetValues(u,N,Local_to_Global_Mapping,
     $     i_value,INSERT_VALUES,ierr) ! Set the initial vector

      call VecAssemblyBegin(u,ierr)
      call VecAssemblyEnd(u,ierr)
      endif
      call KSPSetInitialGuessNonzero(ksp,iflag,ierr)
      call KSPSolve(ksp,rhs,u,ierr)
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*     View the information of solver, preconditioner and matrix
      call KSPView(ksp,PETSC_VIEWER_STDOUT_WORLD,ierr)

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!                         check the error
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

*     Transfer the values from vector u, N elements, to array p.
!     Local_to_Global_Mapping is the global location to get the values.
      call VecGetValues(u,N,Local_to_Global_Mapping,Sol,ierr)
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Max_Sol and Min_Sol
         Solmin =  1e30
         Solmax = -1e30
         Do iv=1,nvariable
            Solmin=Min(Solmin,Sol(iv))
            Solmax=Max(Solmax,Sol(iv))
         EndDo
         Write(imesg,*) 'Min(Sol)=',Solmin,'    Max(Sol)=',Solmax
```

```
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

*     To get the iterations number used for computing
      call KSPGetIterationNumber(ksp,its,ierr)
      write(imesg,*) 'Total Iterations =', its

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*               Clean up and exit the programFree work space.
* All PETSc objects should be destroyed when they are no longer needed.
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      call KSPDestroy(ksp,ierr)
      call VecDestroy(x,ierr)
      call VecDestroy(b,ierr)
      call VecDestroy(u,ierr)
      call VecDestroy(rhs,ierr)
      call MatDestroy(D,ierr)
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*                        End the program.
*        Always call PetscFinalize() before exiting a program
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      call PetscLogPrintSummary(MPI_COMM_SELF,
     $                          'test_system.log',ierr)
! Memory info
      call PetscMemoryGetMaximumUsage(memory,ierr)
      memory=memory*1.0d-6
      write(imesg,*) 'Maximum Memory usage = ',memory,'Mb'
! Petsc Finalization
      call PetscFinalize(ierr)
      write(imesg,*) 'Normal end'
      end
```

Appendix B

An Output File

```
Number of variables:       397600
[1] MatCreateMPIAIJWithArrays()
The vector value is set and assembled.
rhs-b =   9314.84349615265
 0 KSP Residual norm 9.997386552591e+04
 1 KSP Residual norm 1.906239411055e+03
 2 KSP Residual norm 1.515021400569e+03
 3 KSP Residual norm 1.411146600606e+03
 4 KSP Residual norm 1.191624009204e+03
 5 KSP Residual norm 8.052818722618e+02
 6 KSP Residual norm 5.888752359267e+02
 7 KSP Residual norm 5.059841042376e+02
 8 KSP Residual norm 4.769151017274e+02
 9 KSP Residual norm 4.253128546915e+02
10 KSP Residual norm 3.298173385642e+02
11 KSP Residual norm 2.343863701271e+02
12 KSP Residual norm 2.090430289623e+02
13 KSP Residual norm 1.869535145600e+02
14 KSP Residual norm 1.634909140818e+02
15 KSP Residual norm 1.436915208402e+02
16 KSP Residual norm 1.305067901716e+02
17 KSP Residual norm 1.129509724929e+02
18 KSP Residual norm 9.238550828669e+01
19 KSP Residual norm 8.081840559938e+01
20 KSP Residual norm 7.007679651287e+01
21 KSP Residual norm 6.140609196231e+01
```

```
 22 KSP Residual norm 5.383704137012e+01
 23 KSP Residual norm 4.737426575836e+01

KSP Object:
  type: gmres
    GMRES: restart=30, using Classical (unmodified) Gram-Schmidt
                  Orthogonalization with no iterative refinement
    GMRES: happy breakdown tolerance 1e-30
  maximum iterations=500
  tolerances:  relative=0.001, absolute=1e-50, divergence=10000
  left preconditioning
PC Object:
  type: hypre
    HYPRE Euclid preconditioning
    HYPRE Euclid: number of levels 1
    HYPRE Euclid: Using block Jacobi ILU instead of parallel ILU
  linear system matrix = precond matrix:
  Matrix Object:
    type=mpiaij, rows=397600, cols=397600
    total: nonzeros=3824000, allocated nonzeros=3824000
      not using I-node (on process 0) routines
 Maximum error =    9194.30827928375
 Minimum error =   0.000000000000000E+000
 Iterations =          23
 Min(Sol)=  -9194.30827928375          Max(Sol)=   7434.84136860208
 Petsc Resmax =   1.36929558190539
 Elapsed time in PETSc :   6.1251 sec
 Normal end
```

Bibliography

[1] S. Balay, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. Smith, and H. Zhang, *PETSc Users Manual*,Revision 3.1, December 2008. URL `http://www.mcs.anl.gov/petsc/petsc-as/documentation/index.html`.

[2] G.B. Deng, J. Piquet, P.Queutey, and M. Visonneau *A fully coupled Solution of the Navier-Stokes equation*, International Journal of Numerical Method in Fluid, Vol 19, No. 7, Page 605-640, 1994.

[3] G. B. Deng, J. Piquet, X. Vasseur, M. Visonneau, *A new fully coupled method for computing turbulent flows*, Computers & Fluids, Volume 30, Issue 4, May 2001, Pages 445-472.

[6] J. Panyasantisuk, *Integration of PETSc Linear Solver Package into ISIS-CFD Flow Solver*, Master Thesis for M.Sc. in Computational Mechanics, June 2009.

[6] M. Visonneau, *A strongly-coupled velocity-pressure formulation for ISIS-CFD*, Internal report, Dec 2008.

[6] Division Modélisation Numérique, Laboratoire de Mcanique des Fluides, CNRS-UMR 6598, *FINE^{TM}/MARINE Theoretical manual*, Version 2.1, Feb 2009.