INTERNSHIP REPORT

Is Bunyamin SURYO

RENUMBERED ITERATIVE ALGORITHM for NAVIER-STOKES SIMULATION

May - July 2009
Internship of MASTER 1 of MATHMODS Program
University of Nice - Sophia Antipolis, France

Advised by Prof. Alain Dervieux
The TROPICS Project team, INRIA Sophia Antipolis, France

Chapter 1

Introduction

1.1 About INRIA Sophia Antipolis

This training was hold in the Institut National de Recherche en Informatique et Automatique (INRIA) with TROPICS project team during May 4th until June 29th 2009 in Sophia Antipolis. The French National Institute for Research in Computer Science and Control (INRIA) conducts research in the field of Information and Communication Science and Technology (ICST), and more generally in the fields of computer science and modelling. INRIAs Sophia Antipolis - Mediterrane research centre was set up in 1983 at the heart of one of the most important scientific parks in Europe. Today it has premises in Sophia Antipolis, Marseille and Montpellier, bringing together almost 500 staff - including nearly 400 scientists divided into 30 research teams, over half of which have established partnerships with public science and technology institutions.

The scientific research of INRIA are grouped around 5 major themes :

1. Applied Mathematics, Computation and Simulation
2. Algorithmics, Programming, Software and Architecture
3. Networks, Systems and Services, Distributed Computing
4. Perception, Cognition, Interaction
5. Computational Sciences for Biology, Medicine and the Environment

One of project team in Applied Mathematics, Computation and Simulation theme is Program Transformations for Scientific Computing (TROPICS) that coordinated by Laurent HASCOET. The TROPICS project-team is at the junction of two research domains Automatic Differentiation (AD) and Computational Fluid Dynamics (CFD) application of AD where this training conducted. CFD application of AD applies to two real-life problems, optimal shape design and mesh adaption. The training in TROPICS project team was advised by Prof. Alain DERVIEUX with the help of Dr. Stephen WORNOM and Ms. Anca BELME. The activities of training consist of introduction of Linux and Latex system, computation in parallel cluster NEF server with AERO-08 code, and literature study of papers and books. The focus of training is the effects of mesh numbering and the linear solver in Navier-Stokes simulation.

1.2 The Training Aims

In numerical simulation, Central Processing Unit (CPU) time consuming is one of the important aspects that designed to improved or reduced. The more CPU time consuming, the more power needed, it means that the computation cost is not efficient. In order to reduce CPU cost, the researchers try to improve the algorithm's efficiency for computing. For instance, if we want to solve a large sparse matrix with iterative method, the CPU cost is depend upon the behaviour of the sparse matrix. Because the sparse matrix is generated by a particular numerical approximation, the developing algorithm of its numerical approximation is important aspect for improving CPU cost. In this training, the algorithm of mesh numbering will be improved in order to increase the convergence rate of the Navier-Stokes computation in the case of the flow around a circular cylinder.

The study of flow around a bluff body such us a cylinder have been developed long time ago. Nonetheless it is still interesting for researchers to investigate the flow behaviours around a cylinder because there are many engineering problems related with it. For instance the investigation of the seawave load on the offshore structure of oil and gas exploration platform. Seawave, and also wind, will generate a drag load on the riser of platform stucture. By investigation of the flow around the riser, generally the riser has a cylinder geometry, we can calculate the strength of riser structure in order to restrain the drag load of seawave and wind. Consider to this purpose, researchers simulate and study the flow around a circular cylinder. For example in INRIA Sophia Antipolis, The TROPICS project-team, with AERO-08 code, is working on offshore platform hydrodynamics and also on aircraft aerodynamics.

Chapter 2

Navier-Stokes Simulations

2.1 Navier-Stokes Equations

The behaviour of fluid flow is described by well-established partial differential equations (PDE) the Navier-Stokes equations, which are, essentially, particular forms of Newton's laws of motion, supplemented by an equation describing the conservation of mass. They are one of the most useful sets of equations because they describe the physics of a large number of phenomena of academic and economic interest. They may be used to model weather, ocean currents, car and aircraft aerodynamics. As such, these equations in both full and simplified forms, are used in the design of aircraft and cars, the study of blood flow, the design of power stations, the analysis of the effects of pollution, etc.

The Navier-Stokes equations commonly written as

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}$$

where $\mathbf{v}$ is the flow velocity, ρ is the fluid density, p is the pressure, $\mathbf{T}$ is the (deviatoric) stress tensor, and $\mathbf{f}$ represents body forces (per unit volume) acting on the fluid and ∇ is the del operator. This is a statement of the conservation of momentum in a fluid and it is an application of Newton's second law to a continuum.

The Navier-Stokes equations are also of great interest in a purely mathematical sense. Somewhat surprisingly, given their wide range of practical uses, mathematicians have not yet proven that in three dimensions solutions always exist (existence), or that if they do exist they do not contain any infinities, singularities or discontinuities (smoothness). These questions are called the Navier-Stokes existence and smoothness problems. Except for very simple conditions, these equations need to be solved numerically with the aid of computers (often super-computers).

The Navier-Stokes equations are differential equations which, unlike algebraic equations, do not explicitly establish a relation among the variables of interest (e.g. velocity and pressure). Rather, they establish relations among the rates of change. For example, the Navier-Stokes equations for simple case of an ideal fluid (inviscid and incompressible) can state that acceleration (the rate of change of velocity) is proportional to the gradient (a type of multivariate derivative) of pressure.

2.2 Turbulent Flow

2.2.1 What is Turbulence

Many of flows in engineering problem are turbulent thus the turbulent flow regime is not just of theoretical interest. At values of the high Reynolds number (Re_D), $Re_D \succeq 20000$ for external flow, a complicated series of events take place which eventually leads to a radical change of the flow character. In the final state of the behaviour is random and chaotic. The motions becomes intrinsically unsteady even with constant imposed boundary conditions. The velocity and all other flow properties vary in a random and chaotic way. This regime is called turbulent flow. Figure 2.1 represents a structure of turbulent flow, while figure 2.2 represents a fluctuating velocity in turbulent flow. At this condition, turbulent fluctuations cause a much greater net momentum transfer than viscous forces throughout most of the flow. Thus, accurate modelling of the Reynolds stresses is vital.

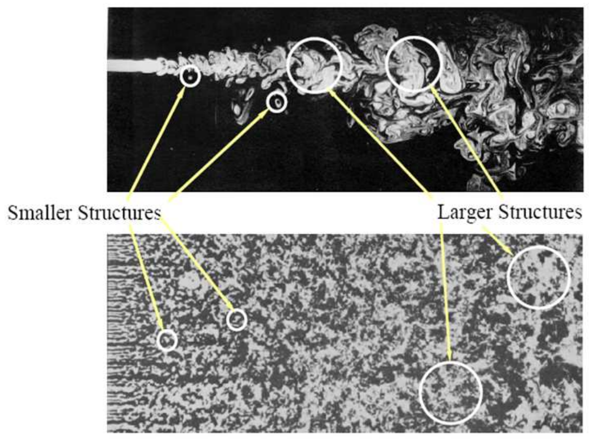

Figure 2.1: Flow structure of turbulent flow

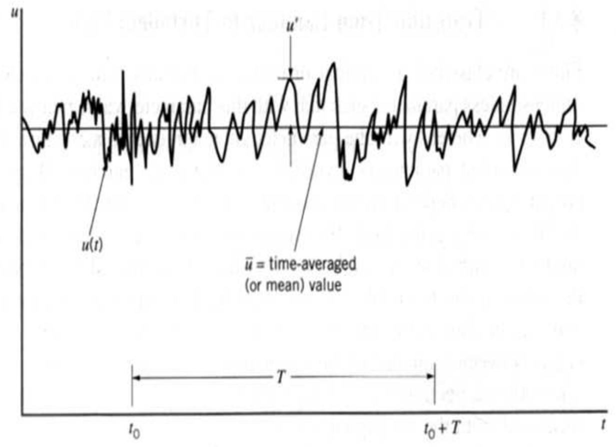

Figure 2.2: Fluctuating velocity in turbulent flow

2.2.2 Turbulent Modelling

Turbulence modeling is the area of physical modeling where a different mathematical model than from the full time dependent Navier-Stokes Equations is used to predict in much easier way the effects of turbulence. There are various mathematical models used in flow modelling to approximate the Reynolds stresses (and other turbulent fluxes) in order to close the mean-flow equations. The classes of turbulence models are classified as

1. Algebraic models
2. Eddy viscosity transport models : one and two equation models
3. Non-linear eddy viscosity models and algebraic stress models
4. Reynolds stress transport models
5. Detached eddy simulations and other hybrid models
6. Large eddy simulations
7. Direct numerical simulations

Here is just an overview some of those models.

Direct Numerical Simulation (DNS)

- Theoretically all turbulent flows can be simulated by numerically solving the full Navier-Stokes equations.
- Resolves the whole spectrum of scales.
- No modeling is required.

- The cost is too prohibitive.
- Not practical for industrial flows

Large Eddy Simulation (LES)

- Solves the spatially averaged Navier-Stokes equations.
- Large eddies are directly resolved, but eddies smaller than the mesh sizes are modeled.
- Less expensive than DNS, but the amount of computational resources and efforts are still too large for most practical applications.

Reynolds-Averaged Navier-Stokes (RANS) Equations Models

- Solve ensemble-averaged Navier-Stokes equations.
- All turbulence scales are modeled in RANS.
- The most widely used approach for calculating industrial flows.

There is not yet a single turbulence model that can reliably predict all turbulent flows found in industrial applications with sufficient accuracy. The general comparison of DNS, LES, and RANS is showed in figure 2.3.

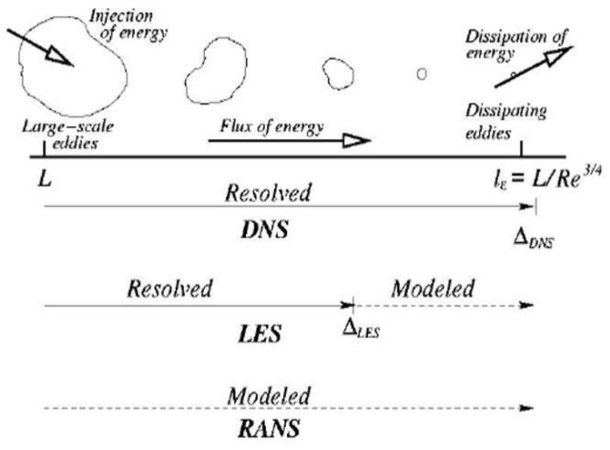

Figure 2.3: Comparison of DNS, LES, and RANS

Because for most engineering purposes it is unnecessary to resolve the details of the turbulent fluctuations, only the effects of the turbulence on the mean flow are usually sought, we can use LES as our turbulence models in modelling flow around a cylinder where time-dependent flow equations are solved for the mean flow and the largest eddies and where the effects of the smaller eddies are modelled. It was argued earlier that the largest eddies interact strongly with the mean flow and contain most of the energy thus LES approach results in a good model of the main effect of turbulence.

2.3 Computational Fluid Dynamics (CFD)

2.3.1 Steps of a CFD Calculation

Computational fluid dynamics (CFD) is the use of computers and numerical techniques to analyze the system or solve the problems involving fluid flow. CFD has been successfully applied in a huge number of areas, including many of interest in engineering such us aerodynamics of aircraft and automobiles. The main stages in a CFD simulation are

1. Pre-processing
2. Solving
3. Post-processing

Pre-processing step consist of problem formulation (governing equations, boundary conditions) and construction of a computational mesh. For more detail, the activities in the pre-processing step involve

1. Definition of the geometry of the region of the interest : the computational domain.
2. Grid generation-the sub-division of the domain into a number of smaller, non-overlapping sub-domain : a grid of cells or elements.
3. Selection of the physical and chemical phenomena that need to be modelled.
4. Definition of the fluid properties.
5. Specification of appropriate boundary condition at the cells or elements.

Solving step involves discretisation of the governing equations and numerical solution of the governing equations. There are some distinct streams of numerical solution of the governing equation, they are finite difference method, finite element method, finite volume method, and spectral method. In outline the numerical methods that form the basis of solver perform the following steps

1. Approximation of the unknown flow variables by means of simple functions

2. Discretisation by subtitution of the approximations into the governing flow equations and subsequent mathematical manipulations.

3. Solution of the algebric equations. The main differences between the separate streams are associated with the way in which the flow variables are approximated and with the descretisation processes.

Post-processing step includes visualisation and analysis of results. The raw output of the solver is a huge set of numbers corresponding to the values of each field variable (u,v,w,p,etc.) at each point of the mesh. This must be reduced to some meaningful subset and or manipulated further to obtain the desired predictive quantities. For example, a subset of surface pressures and cell-face areas is required to compute a drag coefficient.

2.3.2 Discrete Problem

There are many physical phenomenas and engineering problems that can be represented as Partial Differential Equations (PDEs) in order to analyze the phenomenas and solve the problems. For example we use Navier-Stokes equation in order to analyze the flow behaviour around a cylinder with vary of high Reynolds number. But it is quite difficult to solve those PDEs by analitical scheme because of the complexity and assumptions or boundary condition of the problem. In order to overcome this difficulty, numerical approximation is the solution. In numerical approximation, we discretize the domain of the problem in such way, for example :

1. Finite Difference method (FDM)
2. Finite Volume method (FVM)
3. Finite Element method (FEM)
4. Finite Boundary method (FBM)
5. Spectral method (SPM)
6. others

Those discretizations will generate a linear system problem that is represented in either a square sparse matrix, commonly from FDM; FVM; FEM, or a square dense matrix, commonly from FBM; SPM. And then we solve this linear system by a such method, for instance iterative method. Furthermore, we call all those processes as discrete problem.

Hence, discrete problem works for solving the problems in a set of computer program or code. This set of computer program consist of two main parts, Approximate theory and Algorithmic theory. In Approximate theory, the computer program contains a set of assembly routines of numerical approximation such us FVM, FEM, FDM. While in Algorithmic theory, it contains a set of routines of solution such us iterative algorithm. Both of those computer algorithm, Approximate theory and Algorithmic theory, can be developed in a such languange programming, for instance Fortan, C++, etc. Even we try to type all the computer programs in the best way as we can, there are still many limitations in computing discrete problem. Those limitations are contributed by following aspects

1. Real number can not be directly putted in computer because of the binary system number for computer.
2. Limitation of memory storage for computing.
3. Finite number of allowable floating point during computation.

Thus, the result of discrete problem just approximates the exact or continous solution of the real problem.

2.4 Iterative Method for Solving Linear System Problem

2.4.1 Solving Linear System

In general the linear system equation can be formulated as :

$$A.u = f$$

where A is a square matrix of the linear system. There are two schemes in solving the linear system problem, Direct method and Iterative method.

The Direct method will yield a solution of linear system problem in finite number of steps. For example, if we have n *by* n matrix A the number of steps in solving linear system problem by Direct method is n steps. Some of the Direct methods are

1. Gauss Elimination method
2. Kramer Formula method
3. Frontal method
4. others

For really large systems such us computation in weather forecast Direct methods become too expensive in CPU time and storage requirements, and therefore an Iterative approach is needed.

In Iterative methods, they will yield a solution in (theoretically) an infinite number of steps. Thus in Iterative methods we apply a convergence criteria in iterations in order for stopping when we set a solution sufficiently close to the one we seek. There are some methods either simple iterative methods or complex iterative methods, for instance they are

1. Jacobi method
2. Gauss-Siedel method
3. Conjugate Gradient method
4. Generalized Minimal Residual Method (GMRES)
5. others

2.4.2 GMRES and Incomplete Lower Upper (ILU) Preconditioning

Most of the existing practical iterative techniques for solving large linear systems utilize a projection process. The idea of projection techniques is to extract an approximate solution to $A.u = f$ problem from a subspace of $\Re^n$. Let A be an n times n real matrix and $\mathcal{K}$ and $\mathcal{L}$ be two m-dimensional subspaces of $\Re^n$. A projection technique onto the subspace $\mathcal{K}$ and orthogonal to $\mathcal{L}$ is a process which finds an approximate solution ũ to $A.u = f$ by imposing the conditions that ũ belong to $\mathcal{K}$ and that the new residual vector be orthogonal to $\mathcal{L}$.

$$\text{Find } \tilde{\text{u}} \in \mathcal{K}, \text{ such that } f - A\tilde{\text{u}} \perp \mathcal{L}$$

GMRES is a projection method based on taking $\mathcal{K} = \mathcal{K}_m$ and $\mathcal{L} = A\mathcal{K}_m$, in which $\mathcal{K}_m$ is the m-th Krylov subspace. The order-m Krylov subspace $\mathcal{K}_m(A, r_0$ generated by an nxn matrix A and a vector r_0 of dimension n is the linear subspace spanned by r_0 under the first m powers of A (starting from $A^0 = I$), that is

$$\mathcal{K}_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}$$

In general, the convergence of the Iterative methods depends on spectral properties of the coefficient matrix. Hence one way attempt to transform the linear system into one that is equivalent in the sense that it has the same solution, but that has more favorable spectral properties. Thus in order to improve the rate of convergence, we can apply preconditioning for some Iterative methods such us GMRES method for solving a sparse matrix of linear system. Preconditioning is simply a means of transforming the original linear system into one which has the same solution, but which is likely to be easier to solve with an iterative solver. In preconditioning, we use a preconditioner to transform the original linear system. Roughly speaking, a preconditioner is any form of implicit or explicit modification of an original linear system which makes it easier to solve by a given iterative method. For example, scaling all rows of a linear system to make the diagonal elements equal to one is an explicit form of preconditioning. If we have a matrix M that approximates the coefficient matrix A in some way, the transformed system is

$$M^{-1}Au = M^{-1}f$$

has the same solution as the original system $A.u = f$, but the spectral properties of its coefficient matrix $M^{-1}A$ may be more favorable. We call matrix M as preconditioner matrix. We have to choose preconditioner matrix M in order to solve the linear system in easier way than to solve one with A and in practice, the preconditioning operation M^{-1} should be inexpensive to apply to an arbitrary vector.

One of the simplest ways of defining a preconditioner is to perform an Incomplete Lower Upper (ILU) factorization of the original matrix A. This entails a decomposition of the form $A = LU - R$ where L and U have the same nonzero structure as the lower and upper parts of A respectively, and R is the residual or error of the factorization. This incomplete factorization known as ILU(0) is rather easy and inexpensive to comupte. Consider a general sparse matrix A whose elements are $a_{i,j}, i, j = 1, ..., n$. A general ILU factorization process computes a sparse lower triangular matrix L and a sparse upper triangular matrix U so the residual matrix $R = LU - A$ satisfies certain constraints, such as having zero entries in some locations. A general algorithm for building ILU factorizations can be derived by performing Gaussian elimination and dropping some elements in predetermined nondiagonal positions. The Algorithm for computing ILU(0) for a $n\ by\ n$ matrix A is given by

```
$for\ r := 1\ step 1\ until\ n - 1\ do$
 $d := 1/a_{rr}$
  $for\ i := (r + 1)\ step 1\ until\ n\ do$
```

```
          $if\ (i,r) \in\ S\ then$
          $e := da_{i,r}$
            $a_{i,r} := e$
            $for\ j := (r+1)\ step1\ until\ n\ do$
              $if\ ((i,j) \in\ S)\ and\ ((r,j) \in\ S)\ then$
                $a_{i,j} := a_{i,j} - ea_{r,j}$
              $end\ if$
            $end\ (j - loop)$
          $end\ if$
        $end\ (i - loop)$
      $end\ (r - loop)$
```

Here S represents the set of elements of matrix A. The same algorithm could be applied to full matrix A.

Lemma: ILU(0) is a direct solver for tridiagonal matrix

There are some types of ILU preconditioners. ILU factorization technique with no fill-in, denoted by ILU(0), consist of taking the zero pattern of P to be precisely the zero pattern of A, where any zero pattern set of P, such that

$$P \subset \{(i,j)|i \neq j; 1 \preceq i,j \preceq n\}$$

The ILU(0) preconditioner preserves the structure of the original matrix in the result. The use of ILU(0) factorization as a preconditioner is quite frequent when solving linear systems of CFD computations. This is because of its efficiency and moderate memory requirements. Another types of ILU is ILU with threshold, generic name ILUT. Unlike the ILU(0) preconditioner, the ILUT preconditioner preserves some resulting fill-in in the preconditioner matrix structure. The distinctive feature of the ILUT preconditioner is that it saves the resulting entry of the preconditioner if the entry satisfies two conditions simultaneously: the value of the entry is greater than the product of the given tolerance and matrix row norm, and the entry is in the given bandwidth of the resulting preconditioner matrix.

2.5 Meshing Effects

In discretizing the domain of problem (PDEs), we will create a mesh in its domain in such way that we will get a proper matrix A, it means matrix A will be easy to be treated in order to get the solution. The parameters that effect to A matrix condition are mesh scaling and vertice numbering. This matrix condition is the condition number that means the ratio between the maximum eigenvalue and the minimum eigenvalue of A matrix. The formulation of the condition number is

$$\kappa(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}$$

where $\sigma_{max}(A)$ and$\sigma_{min}(A)$ are the maximum and minimum eigenvalue of matrix A, respectively. In numerical analysis, the condition number associated with a problem is a measure of that problem's amenability to digital computation, that is, how numerically well-conditioned the problem is. A problem with a low condition number is said to be well-conditioned, while a problem with a high condition number is said to be ill-conditioned.

The other important matrix property is the bandwidth of matrix. The bandwidth of a sparse matrix is the maximum distance, in diagonals, between two nonzero elements of the matrix. In another way we can say that the bandwidth of a matrix $A = a_{i,j}$ is defined as the maximum absolute difference between i and j for which $a_{i,j} \neq 0$. The problem of reducing the bandwidth of a matrix consists of finding a permutation of the rows and columns that keeps the nonzero elements in a band that is as close as possible to the main diagonal of the matrix. The shorter bandwidth of the matrix, the easier matrix to be treated and otherwise. In order to avoid the large bandwidth of matrix by vertice numbering, we have to choose a proper way when we are going to number the vertice.

There are two typical vertice numberings, Lexicographic and Orthogonal Lexigraphic. In Lexicographic vertice numbering, we use formulation as

$$k = i + (j-1)i_{max}$$

$$u_k = u_{i,j}$$

where $u_k \equiv a_{i,j}$

In Orthogonal Lexicographic, we use formulation as

$$k = j + (i-1)j_{max}$$

where $u_k \equiv a_{i,j}$

If we apply those formulations in vertice numbering on domain of figure 2.4, we will get a matrix of figure 2.5 for Lexicographic and a matrix of figure 2.6 for Orthogonal Lexicographic. For those matrices, the application of ILU results a complete solution of the linear system for Lexicographic and does not solve the linear system for Orthogonal Lexicographic (and in fact it gives a poor approximation of the solution).

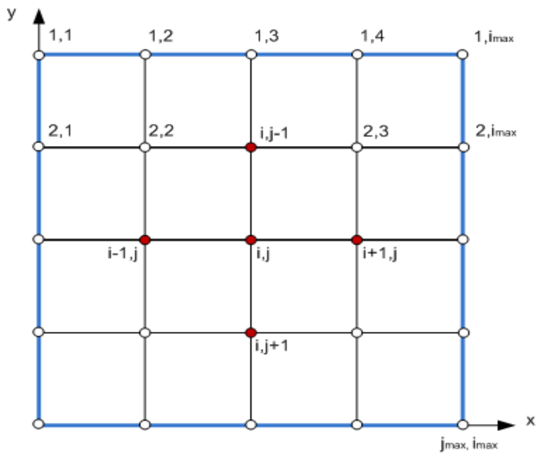

Figure 2.4: Domain of lexicographic and orthogonal lexicographic

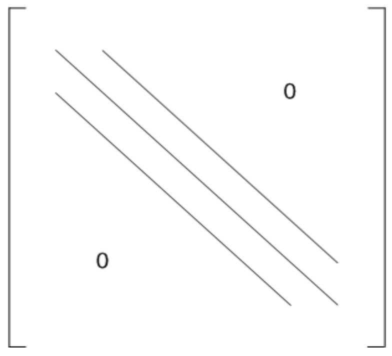

Figure 2.5: Generated matrix A by lexicographic

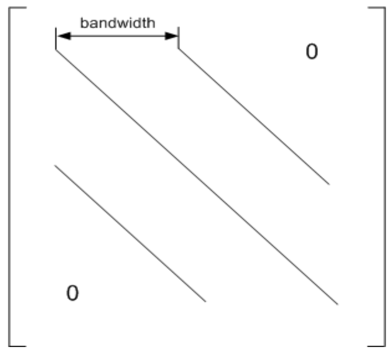

Figure 2.6: Generated matrix A by orthogonal lexicographic

As the conclusion of two typical vertice numbering (lexicographic and orthogonal lexicographic):
If $|a_{i-1,j}| >> |a_{i,j-1}|$, then ILU is more efficient by choosing lexicographic. Otherwise, if $|a_{i-1,j}| << |a_{i,j-1}|$, then ILU is more efficient by choosing orthogonal lexicographic.

In the case of external flow modelling such us modelling for flow around cylinder, the mesh scaling will be very important. In order to know the flow behaviours in the region or sub-domain close to the body (surface) of cylinder such us boundary layer, we have to mesh its domain with finer mesh than the others domain (far from surface cylinder). Of course, this treatment will effect to the condition of generated matrix A. Thus we must optimize this treatment in order to get a proper flow modelling.

Chapter 3

Experiment Description

3.1 Numerical Code AERO-08

We will carry out a flow simulation around a cylinder with a numerical code in multi parallel computing. In this simulation, we will solve numerically unsteady three dimensional (3D) Navier-Stokes equation in high Reynolds number (turbulent Reynolds number). As we explained before, there are three main stages in a CFD simulation : Pre-processing, Solving, and Post-processing. In this work, we will use a numerical code AERO-08 for Pre-processing and Solving step. While for Post-processing, we will use Paraview-3.2.1 to visualize and analyze the results. In Pre-processing and Solving step, we perfomed computation with 8 processors in parallel cluster NEF server of INRIA Sophia Antipolis.

AERO-08 code was developed for compressible flows about aircraft and space shuttles. Then this code is extended for the technology of offshore research which in many ways is not only for compressible flow but also for incompressible flow. AERO-08 code consists of hundred of subroutine programs that are used simultaneously each others. For Pre-processing and Solving step, we will use `Nsc3Dm-HLLE.f` as the main routine program and `flu.data` as the main file for setting of simulation parameters such us numerical scheme methods, number of iterations, magnitude of Reynolds number and etc. We will use GMRES method, written in `GmresASR-uniq.f`, to solve the matrix of linear system of the problem. Furthermore, we use an interface file, called `aero3dgm.x`, in order to get readable solution file for Paraview-3.2.1.

3.2 Compiling and Running AERO-08 Code

In order to compile and run AERO-08 code, firstly, we must connect to parallel cluster NEF server of INRIA Sophia Antipolis. Furthermore, we go to the

directory of the source file in NEF server by the path
`/home/isuryo/AERO-08-Suryo/aero-08/src-F95` and then take a look and check the main routine program `Nsc3Dm-HLLE.f`. For compiling and running the experiments, we follow the path
`/home/isuryo/Cylinder-Riser-NDP-16K-mesh2-43K/8-proc`
in parallel cluster NEF server to get the directory of the case file. In this directory, we take a look three mains file

1. `flu.data`
2. `Makefile-AERO-08-F95-sophia`
3. `submit-8-nef-OpenMpi.pbs`

`flu.data` is the main file for setting of simulation parameters. Some important simulation parameters should be designed in `flu.data` are

1. `ivis` (defines the flow equations, such us Euler, Navier-Stokes, etc.)
2. `les` (defines the turbulence model for eddies, such us LES, k-epsilon, etc.)
3. `rey, xl` (defines the Reynolds number and the reference length for the Reynolds number)
4. `ktmax` (defines maximum number of time-steps)
5. `nexp` (decides whether to use an explicit time or an implicit time-integration strategy)
6. `cflmin, cflstart, cflstep, cflmax` (these parameters define the Courant-FriedrichsLewy (CFL) strategy)
7. `iflux` (specifies the flux algorithm)

After setting of simulation parameters, we compile file
`Makefile-AERO-08-F95-sophia` in order to ensure that all the routines are working properly. Then, if the compiling is successful, we submit the job by file `submit-8-nef-OpenMpi.pbs` to parallel cluster NEF server. We use 8 processors for running the job because we decompose our domain to 8 subdomain. It means that each subdomain is computed in one processor and the final result of each processor will be associated in only one processor in order to get the global solution. Finally, we will get some output files: `nef-32.out`, `solf.xxxxxx.data`, and `startf.xxxxxx.data`. In order to visualize the output files, we use interface file `aero3dgm.x` to convert `solf.xxxxxx.data` and `startf.xxxxxx.data` to be `solf.xxxxxx.vtu`. The last file is readble file for Paraview-3.2.1 which is the post-processing or visualisation software.

3.3 Renumerotation Method

In order to accelerate the convergence rate of GMRES method in solving the matrix of the linear system, we will try to apply a scheme of vertices renumbering for all vertices close to cylinder surface. In applying this scheme, we will modify the subroutine code `SubMesh.f` which is responsible to read the mesh data in subdomain. Inside `SubMesh.f`, we will call subroutine `RenSom.f` that responsible to renumerotate of the list of vertices in order separate interface vertices from those that are purely interior to the submesh. In this experiment, we will modify subroutine `Rensom.f` by introducing and calling a new subroutine `RenumILU.f`. With `RenumILU.f`, the order of vertice numbering will be started from vertice in cylinder surface as the boundary and the next vertice is the closest vertice in orthogonal direction of the boundary. It will proceed until the last vertice in such layer that defined in the code `RenumILU.f`. After that, the order of vertice numbering will be started from vertice in boundary again. This process is showed in figure 3.1 and 3.2.

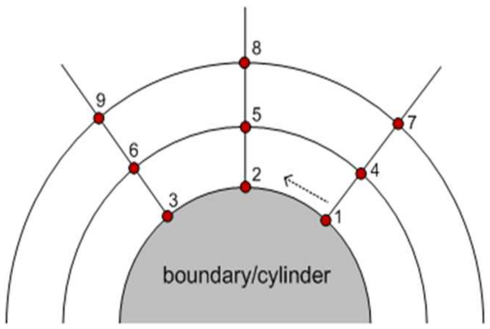

Figure 3.1: Order of vertices numbering before apply `RenumILU.f`

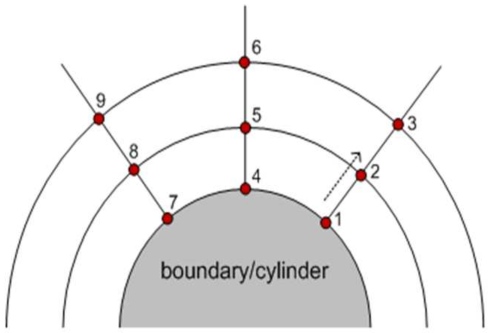

Figure 3.2: Order of vertices numbering after apply `RenumILU.f`

We apply this method only for a finite number of layer close to the cylinder surface or boundary because this region is very important for analyzing the boundary layer development such that we need precise mesh scheme.

Chapter 4

Results and Discussions

4.1 Domain Decomposition

The domain of the computation is showed by figure 4.1 and 4.2. The computation domain consist of 43282 vertices and 228792 tetrahedrals. We can see that the meshes close to the surface of cylinder are much finer than other regions because we need more precise mesh in the region which is close to boundary in order to get sufficient results.

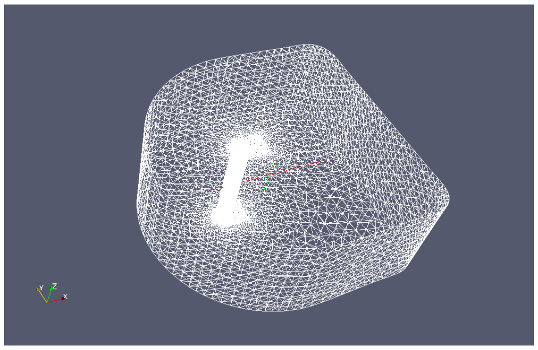

Figure 4.1: Global domain in 3 dimensional view

In this computation, we use parallel computing with 8 processors in parallel cluster NEF server. Such that we decompose the domain of computation onto 8 subdomain, it means that each domain is computed by one processor. In order to get global solution, the results of each processor will be compile in the one designed processor. Figure 4.3 represents the domain decomposition for computation. The number of vertices and tetrahedrals, respectively, for each subdomain are

- subdomain 1 : nsmax,ntmax = 5827, 28938

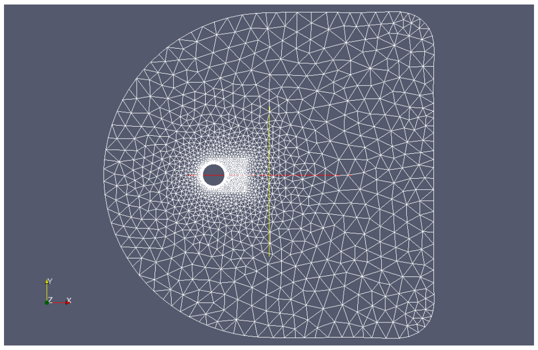

Figure 4.2: Global domain in 2 dimensional view

- subdomain 2 : nsmax,ntmax = 5581, 28091
- subdomain 3 : nsmax,ntmax = 6127, 29449
- subdomain 4 : nsmax,ntmax = 5713, 27781
- subdomain 5 : nsmax,ntmax = 5744, 27767
- subdomain 6 : nsmax,ntmax = 5853, 28957
- subdomain 7 : nsmax,ntmax = 5931, 28889
- subdomain 8 : nsmax,ntmax = 5802, 28920

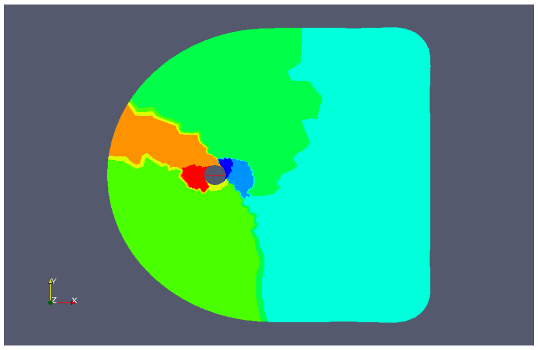

Figure 4.3: Domain decomposition

4.2 Effects of Renumerotation

Inside the subroutine `SubMesh.f` we call subroutine `RenSom.f`. These subroutines responsible to read the mesh data inside the domain. In order to accelerate the convergence of the iterative linear solver, in this case we use

GMRES solver, we modify the order of vertice numbering by applying subroutine `RenumILU.f` inside the subroutine `RenSom.f`. Furthermore we test the modified code in one time step with Reynolds number 100 and only one GMRES iteration. The figure 4.5 and 4.6 show the computation results unmodified code and modified code, respectively.

Comparison between figure 4.5 and 4.6 tell us that the zero velocity or blue line around the surface in figure 4.5 is denser than the zero velocity or blue line in figure 4.6. It means that the developing of non-zero velocity in direction of restrain from the surface in figure 4.6 is faster than in figure 4.5. By the theoretical manner, the fluid velocity of viscous fluid around a surface is equal to zero only on the surface. Thus, with only one GMRES iteration, applying `RenumILU.f` in the subroutine `RenSom.f` will give more sufficient result as figure 4.6 that means the convergance rate is accelerated in modified code. More explicitly, the figure 4.7 and figure 4.8 tell us about the result comparison between unmodified code and modified code. While figure 4.9 tells the velocity profile for unmodified and modified code.

Basically, applying `RenumILU.f` in the subroutine `RenSom.f` will change the spectral properties of generated matrix of linear system. Either Unmodified code and modified code perform a different order of vertices numbering in a such way such that the generated matrix of unmodified code will be much sparser or less denser than generated matrix of modified code. With less denser of spectral properties, the generated matrix of unmodified code is more difficult to solve than the generated matrix of modified code. It means that the convergance rate of unmodified code is longer than modified code. Thus, with only one GMRES iteration, the modified code generates more sufficient result than the unmodified code.

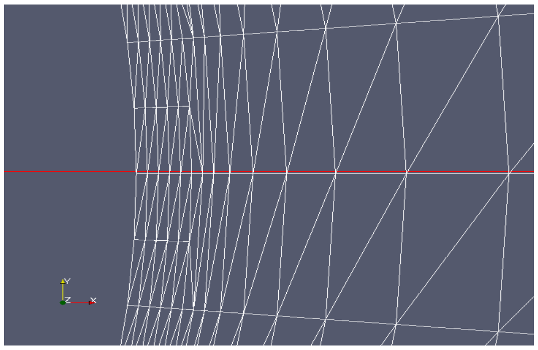

Figure 4.4: Meshing layers close to surface as the object for applying `RenumILU.f`

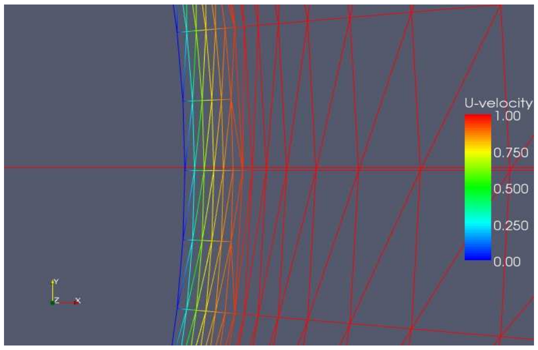

Figure 4.5: Developing of velocity value in Re=100, GMRES iteration=1, before applying `RenumILU.f`

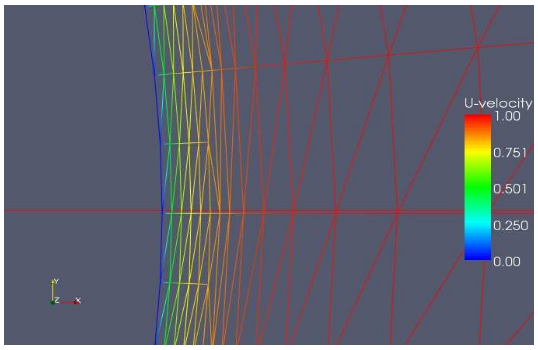

Figure 4.6: Developing of velocity value in Re=100, GMRES iteration=1, after applying `RenumILU.f`

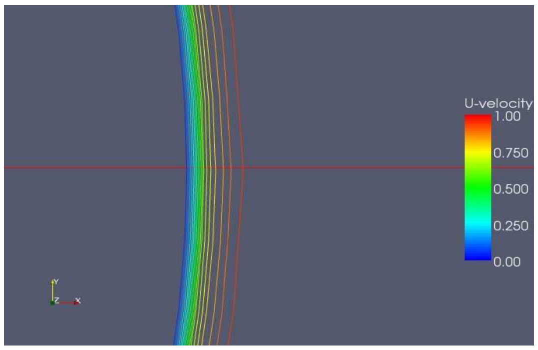

Figure 4.7: Isovelocity value in Re=100, GMRES iteration=1, before applying `RenumILU.f`

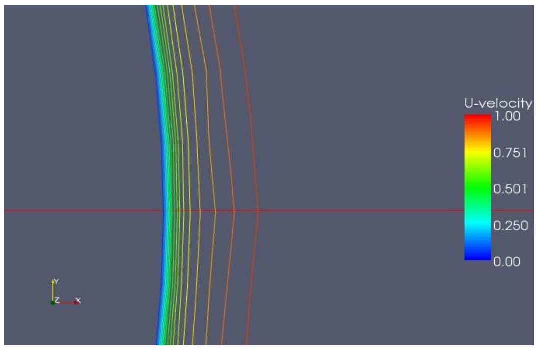

Figure 4.8: Isovelocity in Re=100, GMRES iteration=1, after applying `RenumILU.f`

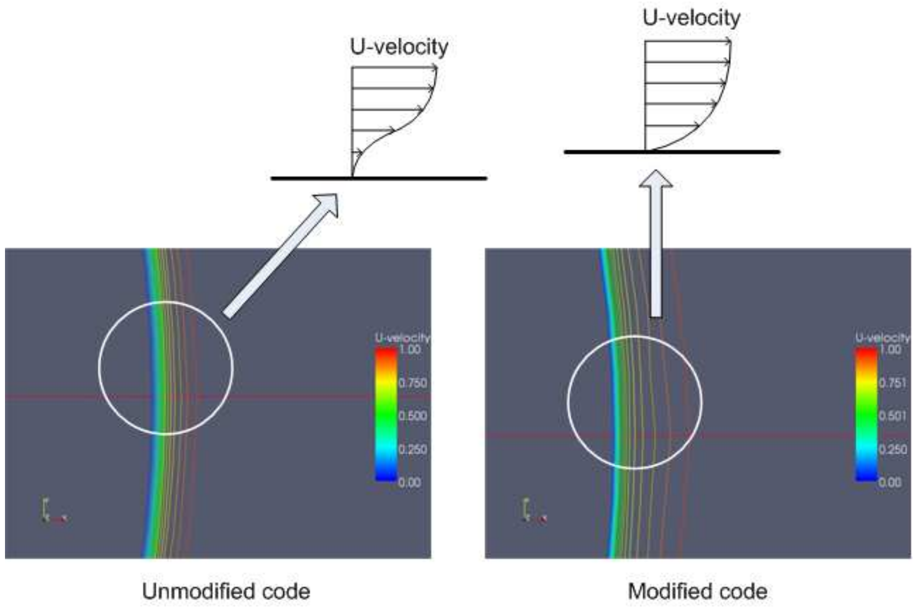

Figure 4.9: Velocity profile on the surface/boundary

Chapter 5

Conclusions

Navier-Stokes equation is one of the most interesting in fluid dynamic study. There are some numerical approximations used for solving this equation such us finite volume method and finite element method. Each of used numerical approximation for solving Navier-Stokes equation leads to generate a linear system which is formulated by $A.u = f$, where A is a square matrix. In order to get the solution of Navier-Stokes equation, we have to solve the linear system with either direct methods or iterative methods. In using the iterative methods, the behaviours or properties of matrix A strongly contribute to the convergence rate of the used iterative method. The larger matrix A coefficients are gathered close to diagonal, the easier the linear system to be solved.

The properties of matrix A are contributed by how the numerical approximation developed, for instance how to construct and partition off the mesh and how to number the vertices. In this training, we modified the vertice numbering system of AERO-08 code in order to accelerate the convergence rate of linear solver, GMRES, by introducing and calling a new subroutine `RenumILU.f` inside the subroutine `RenSom.f`. By the test case of the flow around a cylinder, with Re=100 and one step of GMRES iteration, it is proved that the modified code can accelerate the convergence rate of linear solver. It means that the modified code generated a matrix A on better properties than a matrix A of unmodified code such that the linear system is easy to be solved. Finally, by acceleration of the convergence rate in solving the linear system, it can save both computation power and computation time.

Bibliography

1. Y. Saad. *Iterative Methods for Sparse Linear System.* PWS Publishing Company, Boston, 1996.

2. INRIA. AERO-08 User Manual. Release: v2.0 F95, Sophia Antipolis, 2008.

3. Fluent User Services Center. Introductory FLUENT Notes. FLUENT v6.2, 2005.

4. Kitware, Inc. Paraview 3.2.1, New York, 2009.