



**INTEGRATION OF A LARGE EDDY MODEL FOR
TURBULENT FLOW SIMULATIONS INTO THE HPC
FRAMEWORK PEANO**

BY

M.Ibrahim Mohamed

SUBMITTED TO THE INSTITUTE OF INFORMATICS
AND THE EXAMINATION BOARD
OF TECHNISCHE UNIVERSITÄT MÜNCHEN
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTATIONAL SCIENCE AND ENGINEERING

1st Examiner: Prof.Dr. Hans-Joachim Bungartz

2nd Examiner: Prof.Dr. Thomas Huckle

Supervisor: Dipl.-Tech.Math. Tobias Neckel

Munich on September 1, 2008

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

September 1, 2008

M. Ibrahim Mohamed



In the name of Allah, the most graceful, the most merciful

Acknowledgment

First, all the thanks and praises are to Allah who gave me all what I have, and guided me all the way to the right path.

Then I would like to pay all the gratitude to my family. I would like to thank my father for his continuous trust, support and motivation in my academic progress which started ever since I was a child, and also for funding my engineering studies, and afterward my stay in Germany to finish my master's degree. Then I would like to thank my mother for her care, always being there when I needed her, and for her emotional support. In addition to all the happy moments and the peaceful atmosphere that she provided our life with. I am really grateful for all the many great things they have given me, without their continuous encouragement and support, I would not be able to achieve what I have done so far. It's an honor to be their son.

Also, I would like to thank my sister for being patient with my parents's support and for encouraging them to keep up their support for me while being very happy and proud of my success. She is one of a kind sister that can't be thanked enough for all what she has put up with and done for me. I would like to thank my lovely wife for being here beside me, for her care and support, for her patience, and for helping me typing and proofreading the thesis.

I am very grateful to Prof.Dr. Mustafa Gouda for his care, academic support, and the opportunities he offered me during the five years of my undergraduate studies. I also feel gratitude to Dr.Osama El-Reedi, Prof.Dr. Hesham Moustafa, Prof.Dr. Medhat Metawee, Prof.Dr. Ahmed Sultan, and all my Professors who taught me at the Higher Technological Institute.

I would like to thank my supervisor Dipl.-Tech.Math. Tobias Neckel for providing

me with the opportunity to work on a very interesting topic for this master's thesis, and for always offering me the time when I needed his support. This thesis was enabled by his continuous support and brilliant guidance, I really admire his personality and his style of supervision that always lift up my spirit.

I also would like to thank Prof.Dr. Bungartz for accepting to be the first examiner and Prof.Dr. Huckle for accepting to be the second examiner, indeed its a real honor. Furthermore, I would like to thank them, thank Dr. Micheal Bader, and all the Computational Science and Engineering (CSE) international master people for the chance that they gave me to participate, for the great efforts they spent to organize such a prestigious master's program, and for finding solutions for my problems during the last two years.

I would like to thank Prof.Dr. Micheal Manhart for his valuable advices and explanations that helped me during my work, and enabled me to better understand the turbulence phenomena. I would like to thank all the Professors who taught me at Technische Universität München.

I am very grateful to Mohamed Bamakhrama for the time and scientific support that he offered me during my two years of the CSE. I would like to thank Coşkun İzgür, Fayssal El-Moufatich, and Abdelali Zahi for being there when I needed their advices and help. Also, many thanks to Ahmed Omran and Marwan Radi for proofreading the thesis.

Last, but not least, a big "thank you" to all of my friends for being there all the time.

Abstract

Turbulence is one of the most interesting phenomenons for science and engineering applications, and therefore a promising Computational Fluid Dynamics (CFD) tool has to have the capability of tackling problems involving turbulence. Starting from this fact, this thesis work was accomplished to integrate and test a turbulent solver into the High Performance Computing (HPC) Peano framework.

The Large Eddy Simulation (LES) paradigm was chosen as a base for the turbulent solver. This decision was taken because LES lies between the Direct Numerical Simulation (DNS) and statistical simulations in the sense of the information offered and the cost by/of the simulation. In addition, LES is a modern paradigm for a modern object-oriented solver. After the literature survey was accomplished to understand the fundamentals of LES, the simple Smagorinsky eddy viscosity model was selected with a coarse grid as a filter.

Before the implementation of the Smagorinsky eddy viscosity model, many tasks were performed to add the necessary functionalities within the framework for the regular and the adaptive grids implementations. These functionalities include averaging, restart mechanisms, and checkpointing. Afterward, the model was implemented on the regular grid, and the need for Van Driest damping and the extension of the implementation to the adaptive grid arose during the early numerical simulations stage. All the solver functionalities were tested on both unit and integration levels. Finally, many numerical simulations were performed, the results were analyzed, and more advanced model was recommended for the future extension of the project.

Kurzfassung

Turbulenz ist eine der interessantesten Phänomene für wissenschaftliche und technische Anwendungen, demzufolge muss ein vielversprechendes Werkzeug für Numerische Strömungsmechanik (Engl. Computational Fluid Dynamics - CFD) die Fähigkeit besitzen, Probleme mit Turbulenzen anzugehen. Ausgehend von dieser Tatsache wurde diese Arbeit verfasst, um einen Turbulenzlöser in das Hochleistungsrechenframework (High Performance Computing - HPC) 'Peano' einzubinden und zu testen.

Das Beispiel der Simulation großer Wirbelstrukturen (Large Eddy Simulation - LES) wurde als Basis für den Turbulenzlöser ausgewählt. Diese Entscheidung wurde getroffen, weil die LES zwischen der direkten numerischen Simulation (DNS) und statistischer Simulationen liegt, im Sinne der angebotenen Informationen und des Rechenaufwandes. Darüber hinaus ist die LES eine moderne Basis für einen modernen objektorientierten Löser. Nachdem die Literaturuntersuchung durchgeführt wurde, um die Grundlagen der LES zu verstehen, wurde das einfache Smagorinsky-Modell zur Wirbelviskosität ausgewählt, mit einem groben Gitter als Filter.

Vor der Implementierung des Smagorinsky-Modells der Wirbelviskosität wurden viele Aufgaben ausgeführt um die nötigen Funktionalitäten innerhalb des Frameworks für die gewöhnliche und adaptive Gitterimplementierung hinzuzufügen. Diese Funktionalitäten beinhalten die Mittelwertbildung, Neustartmechanismen und Kontrollpunktsetzung. Danach wurde das Modell auf dem gewöhnlichen Gitter implementiert, und der Bedarf nach Van Driest Dämpfung und der Erweiterung der Implementierung auf den adaptiven Gitter ergab sich während der frühen Phase der numerischen Simulation.

Sämtliche Löserfunktionalitäten wurden sowohl auf Unit- als auch auf Integration-Test-Ebene getestet. Anschließend wurden zahlreiche numerische Simulationen durchgeführt, die Ergebnisse analysiert und ein fortgeschritteneres Model für den weiteren Ausbau des Projektes empfohlen.

Nomenclature

List of symbols

A^+	Van Driest constant
b	duct width
C	cross stress tensor
C_{smag}	Smagorinsky constant
f	any filed variable
\bar{f}	filtered field variable
f'	subgrid contribution
h	kernel function
k	wave number
k_c	cut-off wave number
L	Leonard stress tensor
L_x	duct length
p	pressure normalized by the density
p_w	wall pressure
Q	any space averaged statistic
R	Reynolds sub-grid stress tensor that represents
\bar{S}	strain rate of the filtered quantities
u	velocity field
u_1	velocity component in x direction
u_2	velocity component in y direction
u_3	velocity component in z direction
u_{exact}	exact velocity filed

u_τ	friction velocity
X	Cartesian coordinate system
y^+	distance from the wall normalized by the viscous lengthscale
ν	kinematic viscosity
Δ	filter width
δ	half of the duct height
ρ	density
δ_ν	viscous lengthscale
Re_τ	friction Reynolds' number
ν_{smag}	Smagorinsky subgrid viscosity
τ	subgrid stress tensor
τ_w	wall shear stress
γ	constant that scales the Gaussian distribution

Abbreviations

CFD	Computational Fluid Dynamics
HPC	High Performance Computing
LES	The Large Eddy Simulation
DNS	Direct Numerical Simulation
SLE	System of linear equations
PDE	partial differential equations
RMS	Root Mean square

Contents

Acknowledgment	iv
Abstract	vi
Kurzfassung	vii
Nomenclature	ix
1 Introduction	1
1.1 Fluid dynamics	1
1.2 Computational science	3
1.3 Computational fluid dynamics	4
1.3.1 Levels of space and time resolution	5
1.3.2 Features and advantages of CFD	5
1.4 Turbulence	7
1.4.1 Turbulence structures	8
1.4.2 Different paradigms used to tackle turbulence	8
1.5 Structure of the thesis	10
2 Basics of LES	11
2.1 Scale separation or filtering	12
2.1.1 Explicit filtering “Convolution integral”	12
2.1.2 Implicit filtering “Discretization Effect”	16
2.1.3 The effective filter	16

2.2	Filtered Navier-Stokes	17
2.2.1	Filtered Navier-Stokes	17
2.2.2	Leonard's decomposition	17
2.2.3	The closure problem	18
2.3	Smagorinsky model	18
2.3.1	Interaction between grid and sub-grid scales	19
2.3.2	The model structure	20
2.4	Validation of LES data	21
2.4.1	Validation techniques	21
2.4.2	Benchmark	24
3	Basics of Peano	29
3.1	The framework Peano	29
3.2	Features	29
3.2.1	Performance related features	30
3.2.2	Structure related features	31
3.3	CFD component	31
4	Implementation Aspects	33
4.1	Enlarging the CFD component	34
4.2	Statistical averaging	35
4.2.1	The averaging on the regular grid	38
4.2.2	The averaging on the adaptive grid	40
4.3	Restart Mechanism	41
4.4	LES Implementation Details	43
4.4.1	The filtering	43
4.4.2	The Smagorinsky model	44
4.4.3	Van Driest damping	46
5	Numerical Experiments	49
5.1	The scenario	49
5.2	The experiments	51

5.2.1	Experiments on the regular grid	51
5.2.2	Experiments on the adaptive grid	52
5.2.3	Conclusions	56
6	Conclusion and Outlook	63
	Bibliography	65

List of Tables

5.1	The different stimulations that were conducted at the adaptive grid for a maximum mesh size of $2/27$ and minimum mesh size of $2/81$ in y direction, and the way that they were coded.	53
-----	---	----

List of Figures

1.1	Main activities on a simulation pipe line	4
1.2	Schematic drawing which shows that when the mesh is not fine enough, some information might be lost.	6
1.3	Schematic drawing which shows that when the mesh is refined enough, the small scales information is captured.	6
1.4	schematic drawing time evolution of the axial velocity component in a turbulent flow	7
1.5	Schematic drawing of the resolved and modeled energy spectrum in DNS, k is the wave number, and $E(k)$ is the energy spectrum.	9
1.6	Schematic drawing of the resolved and modeled energy spectrum in RANS.	9
1.7	Schematic drawing of the resolved and modeled energy spectrum in LES.	10
2.1	The sharp cut-off filter in spectral space. Source [2].	15
2.2	The top-hat filter in spectral space. Source [1].	15
2.3	The Gaussian filter in spectral space. Source [1].	16
2.4	The forward and backward energy cascades.	19
2.5	The channel geometry. Source: [3].	25
3.1	Peano Framework Features.	30
4.1	The extension of the fluid component to accommodate the turbulence subcomponent	34
4.2	General idea of space averaging.	36

4.3	The data that lies on one of the two adjacent periodic boundaries is excluded from averaging.	37
4.4	The data that lies on one of the two adjacent periodic boundaries is excluded.	37
4.5	The class diagram for the created Average data types and the wrappers for the standard sets.	40
4.6	Schematic drawing for the idea of the restart capability.	42
4.7	Schematic drawing for the idea of the checkpointing.	42
4.8	The original implementation of the diffusion operator computation.	44
4.9	The extensions of the implementation to add the subgrid contribution to the diffusion operator computation.	46
5.1	The flow velocity filed at Re_τ of 3950 and mesh size of $(\frac{1}{16})$ on the regular grid after 3000 time steps	52
5.2	The three dimensional velocity field in a surface representation of the run at Re_τ of 395 , and C_{smag} of 0.1 after 3000 time steps	53
5.3	An arrow representation of the velocity field at a slice taken at the center of the channel, and normal to the spanwise direction. The run was at Re_τ of 395 , and C_{smag} of 0.1 after 3000 time steps	54
5.4	A slice taken at the center of the channel, and normal to the spanwise direction. The run was at Re_τ of 395 , and C_{smag} of 0.1 after 3000 time steps	54
5.5	The Peano's LES mean velocity normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]	55
5.6	The Peano's LES streamwise velocity variance normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]	55
5.7	The Peano's LES wall-normal velocity variance normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]	56
5.8	The Peano's LES spanwise velocity variance normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]	56

5.9	The Peano's LES uv normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]	57
5.10	The Peano's LES mean velocity normalized by u_τ vs the reference data of Tokyo[9]	57
5.11	The Peano's LES streamwise velocity RMS normalized by u_τ vs the reference data of Tokyo[9]	58
5.12	The Peano's LES wall-normal velocity RMS normalized by u_τ vs the reference data of Tokyo[9]	58
5.13	The Peano's LES spanwise velocity RMS normalized by u_τ vs the reference data of Tokyo[9]	59
5.14	The Peano's LES pressure RMS normalized by u_τ vs the reference of Tokyo[9]	59
5.15	The mean velocity vs. the reference data of Mansour, Moser and Kim[8]	60
5.16	The streamwise velocity variance vs. the reference data of Mansour, Moser and Kim[8]	60
5.17	The wall-normal velocity variance vs. the reference data of Mansour, Moser and Kim[8]	61
5.18	The spanwise velocity variance vs. the reference data of Mansour, Moser and Kim[8]	61
5.19	The uv vs the reference data of Mansour, Moser and Kim[8]	62

Chapter 1

Introduction

LES is one recent and promising paradigm to tackle turbulent flows. Before proceeding to understand such an advanced topic of LES, and the work done in this thesis, this chapter is meant to give a brief introduction to the fundamentals and basics of fluid dynamics, computational science, CFD, and turbulent flows.

1.1 Fluid dynamics

Fluid dynamics is the science of studying the fluid flow, both its kinematics¹ and kinetics². The fluid flow is observed in nature during our everyday life (e.g. the flow of rivers, the ocean waves, the effect of wind on tree leaves, etc.) and sometimes has catastrophic impacts (e.g. tornadoes, Tsunami, etc.). Furthermore, it plays more than an important role in science and engineering (e.g. plasma physics, aerospace engineering, automotive engineering, power technologies and energetic systems, chemical engineering, etc.). Thus, the study of fluid flow has been always a great concern and an interesting topic for passionate scientists.

Fluid flow is governed by the conservation laws of momentum, mass, and energy. Incompressible flows on a continuum scale³ are thought to be best governed by the

¹kinematics is the part of dynamics where the motion is the only interest, regardless the forces that causes it.

²kinetics is the part of dynamics where the motion and the forces causing it are both of interest.

³In the continuum approach the fluid media is thought to be continuous

fundamental and famous Navier-Stokes equations,

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i u_j) = -\frac{\partial p}{\partial x_i} + \nu \frac{\partial}{\partial x_j} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right), \quad i = 1, 2, 3 \quad (1.1)$$

and the continuity equation,

$$\frac{\partial u_i}{\partial x_i} = 0 \quad (1.2)$$

where $X = (x_1, x_2, x_3)$ is the Cartesian coordinate system, $u = (u_1, u_2, u_3)$ is the velocity field, p is the pressure normalized by the density, and ν is the kinematic viscosity. Eq. (1.1) is simply the momentum balance where the product of a finite fluid element mass and its acceleration is equal to the sum of all the forces that exerted on it. Eq. (1.2) is nothing but the conservation of mass, where what comes in should of course comes out.

The fluid flows can be classified into many categories, listed below the fundamental classifications.

1. Based on dimensions:

- Two dimensional.
- Three dimensional.

2. Based on compressibility

- Compressible.
- Incompressible.

3. Based on time dependency

- Steady.
- Unsteady.

4. Based on irregularity

- Laminar.
- Transient.

- Turbulent.
5. Based on viscous effect
- Viscous.
 - Inviscid.

Navier-Stokes is time dependent, three dimensional, compressible, and non-linear, thus an analytical solution of it in its most generic form is impossible, but under many assumptions and simplifications done to a problem an analytical solution might be possible (e.g. laminar incompressible flows in pipes, potential flows, etc.). For more details [6], and [7] are recommended.

1.2 Computational science

Computational science is the art of optimally utilizing the computer resources and employing the proper numerical techniques to produce accurate and time efficient numerical simulations.

Whether it's a problem of studying fundamental phenomena or designing and optimizing a highly complicated system, nowadays modeling and simulations play a key role in answering open questions or finding solutions. There are many types of models and subsequently many different types of simulations. The scope here is about the mathematical models and numerical simulations.

Numerical simulations After the physical phenomena or an engineered system are forced to obey a group of governing equations, these equations have to be solved to enable the study of the system and its aspects of interest. In many cases an analytical solution is impossible unless many assumptions are posed. These assumptions are sometimes impractical, and the more generic form of the problem is of higher interest. Here comes the role of the numerical treatment and numerical simulations. Fig. 1.1 shows the main activities during modeling and simulation of a problem.

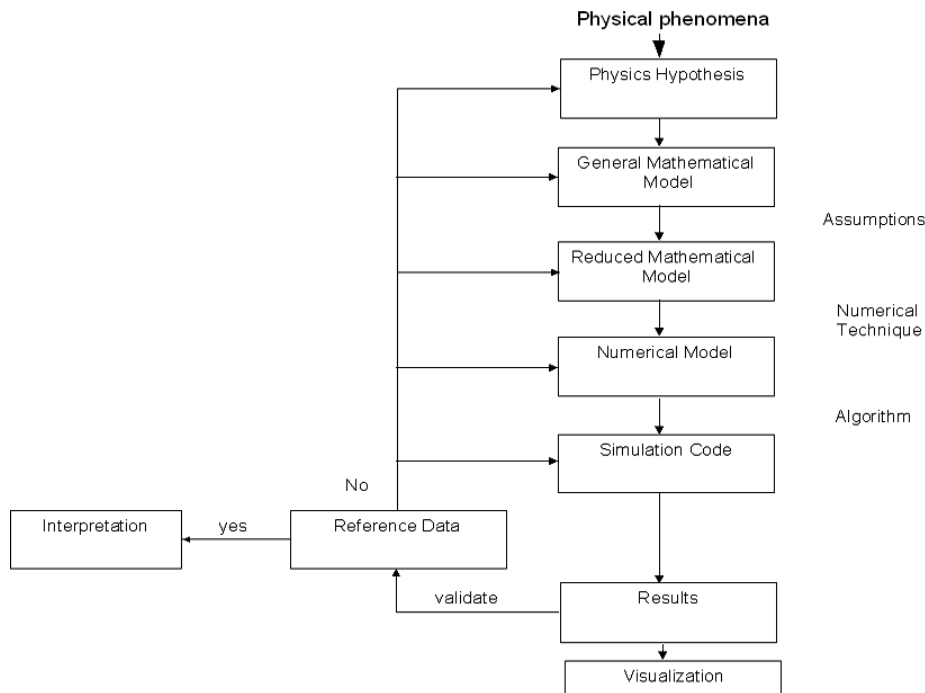


Figure 1.1: Main activities on a simulation pipe line

The simulation pipe line starts with setting the physics hypotheses and deriving the mathematical model out of it. The mathematical model complexity can be reduced by posing the proper assumptions. Afterward the reduced mathematical model can be treated numerically to convert it to a numerical model. Once the numerical model is constructed the computer power can be utilized by means of implementing the proper numerical algorithms resulting in a simulation code. At the end of the pipe line come the numerical simulations, the visualization of the results, the validation cycle, and the analysis of the results to interpret it.

1.3 Computational fluid dynamics

Computational fluid dynamics is the interdisciplinary field of computational science and fluid dynamics. CFD was developed to enable numerical experiments for fluid flow scenarios. Like in many science and engineering problems, where dynamic systems are of interest, Fluid dynamics modeling results in a system of partial differential

equations (e.g. the continuity and Navier-Stokes on the continuum level).

Elsewhere the modeling is involved, the main idea behind the numerical experiments for fluid flow scenarios is as the following; the PDE that are valid continuously over the flow domain are of course valid at each discrete point on it. Thus, these equations can be represented in a finite fashion on a discrete domain. The usage of a discretization technique (e.g. finite element, or finite differences) gives rise to a system of linear equations (SLE) instead of the PDE system. This system has no unique solution, except when the boundary conditions are imposed⁴. Many efficient numerical schemes and algorithms already exist for solving SLE both sequentially or in parallel. The numerical accuracy and the capability of capturing the system dynamics are significantly influenced by the mesh size.

1.3.1 Levels of space and time resolution

In order for a numerical simulation to have valid and meaningful results, it should capture all the scales that might have a dynamic effect and might influence the validity of the results. This simply means, the temporal and spatial meshes should have sizes that are smaller than these scales. Otherwise, information might fall down from the net (See Fig. 1.2, and 1.3).

1.3.2 Features and advantages of CFD

Here are listed some features and advantages of CFD comparing to analytical and experimental solutions:

- It helps when analytical solutions and experiments are impossible or extremely difficult.
- It allows easy and time efficient setup of an experiment.
- It produces accurate and highly dense information when compare to physical experiments

⁴For more details about discretization technique or Computational fluid dynamics in general [2] is advised

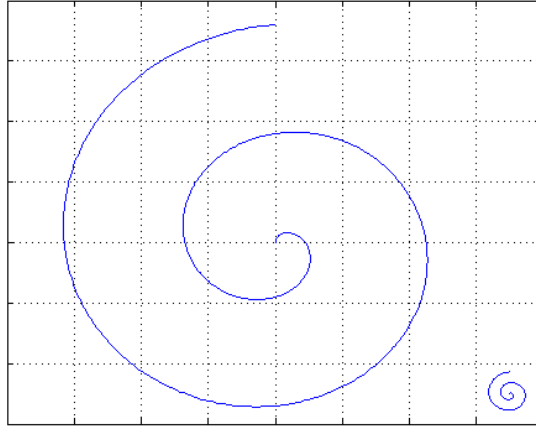


Figure 1.2: Schematic drawing which shows that when the mesh is not fine enough, some information might be lost.

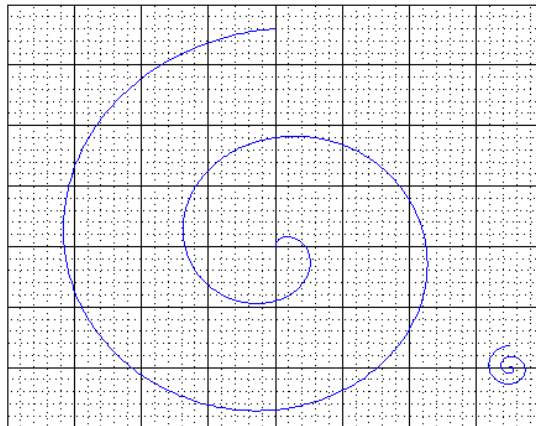


Figure 1.3: Schematic drawing which shows that when the mesh is refined enough, the small scales information is captured.

- It allows access to regions within the domain where the measurement instruments cannot be placed there.

1.4 Turbulence

The turbulence is the flow at which the viscosity force fails to regulate the flow facing a relatively high inertial force (i.e at high Reynolds number). As a result, each particle is subject to continuous fluctuations of its velocity in magnitude and direction, and flow suffers irregularity in time and space (see Fig. 1.4

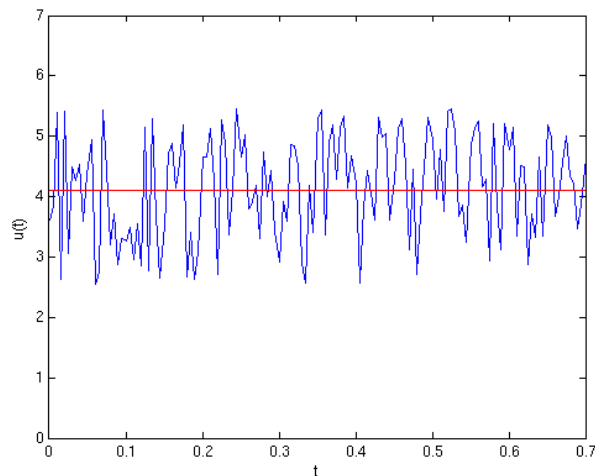


Figure 1.4: schematic drawing time evolution of the axial velocity component in a turbulent flow

The importance of studying turbulence comes from the fact that it appears mainly in all flows of practical interest either in geophysics or engineering sciences, and it significantly affects the global features of these flows (e.g. wall friction, heat transfer, proper mixing, dispersion of additives and pollutants). Turbulence has always been a challenging work, and so far no predictive theory has been found that can cover the diversity of turbulent flows.

Further explanations regarding the nature of the turbulence and the different well

known paradigms to tackle it will be discussed in the following sections.

1.4.1 Turbulence structures

It is believed that both laminar and turbulent flows are best governed by Navier-Stokes equations. Due to the non-linearities of the convection term of Navier-Stokes a broad range of spatial and temporal scales arises as a function of the flow Reynolds number.

The spatial turbulent scales are mainly classified into two categories, The Large Eddies or “Energy Carriers”, and the Small Eddies “Energy Dissipative”. According to the energy cascade and Kolomogorov’s hypothesis of the local isotropy and similarity, the following properties of these two categories are extracted.

The large eddies carry most of the turbulent kinetic energy, so they are mainly responsible for turbulent transport, thus they are responsible about the mass and momentum transfer. They are mainly anisotropic and directly affected by the boundary conditions. The large scales are broken down given their energy to the small scales in a forward energy cascade. The small eddies they are responsible for most of the energy dissipation, as they drain energy from the larger scales and transfer it to the smaller eddies. This process continues, until the smallest scale lose its energy by the viscous effect. At high Reynolds number the small scales are thought to be isotropic, homogeneous and simple in nature, so they can be easily modeled.

1.4.2 Different paradigms used to tackle turbulence

DNS

In DNS the governing equations are numerically solved respecting all scales of dynamic effect both spatial and temporal. Fig. 1.5 shows that in DNS all the energy spectrum is resolved.

Scale problem Any meaningful simulation has to cover both effects of the large and small eddies, so the resolution that is required to capture both scales dynamic effect is a function of the largest to the smallest eddy ratio.

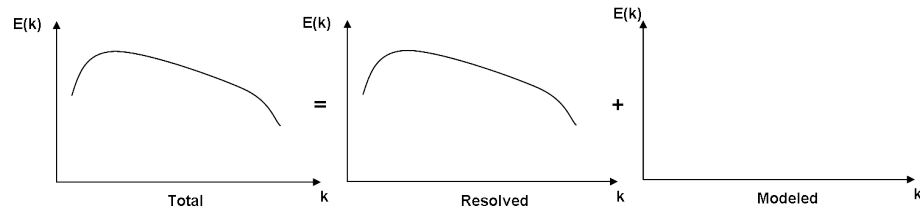


Figure 1.5: Schematic drawing of the resolved and modeled energy spectrum in DNS, k is the wave number, and $E(k)$ is the energy spectrum.

As the scale disparity is a rapidly growing function of Reynolds number, DNS is restricted to low Reynolds number flows and fails to simulate high Reynolds number flows using our current computer resources.

RANS

Reynolds-averaged Navier-Stokes is the second well known paradigm at which the degrees of freedom are reduced by averaging Navier-Stokes and calculating only the statistical average of the flow field.

The effect of the fluctuations on the averaged quantities is imposed by the means of models (i.e all turbulent scales are modeled). Fig. 1.6 is a schematic drawing of the resolved and modeled spectrum in RANS.



Figure 1.6: Schematic drawing of the resolved and modeled energy spectrum in RANS.

In RANS the space averaging is usually accompanied by time averaging, as a result the evaluative behavior cannot be described by means of RANS.

LES

In Large Eddy Simulation (LES) only the large turbulent structures or low frequency fluctuations are resolved (see Fig. 1.7). The effect of the small scales are imposed by the so called subgrid scale model.

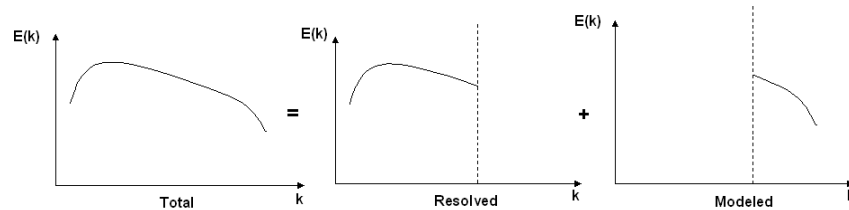


Figure 1.7: Schematic drawing of the resolved and modeled energy spectrum in LES.

For a quick survey about DNS, RANS, and LES the handbook of computational fluid mechanics [2] is recommended, and for more details about turbulent flows and LES, the books of Stephen B. Pope [3] and Pierre Sagaut [1] are recommend.

1.5 Structure of the thesis

Chapter 2 presents explanations for many topics regarding LES that will be the base and guide to understanding the work of this thesis. Chapter 3 outlines the Peano framework and its features. Chapter 4 describes the implementation issues of the LES based turbulent solver. In Chapter 5, The numerical simulations are reported. Chapter 6 draws the summary of conclusions and suggestions are outlined regarding possible further research work on the topic.

Chapter 2

Basics of LES

LES is a compromise between DNS, and statistical simulations. It is less costly than DNS, but much more expensive when compared to statistical simulations. In LES the large scales (grid scales) are resolved on the grid and the small scales (subgrid scales) are modeled. Thus, it allows the study of complex three dimensional and time dependent flows, when pure modeling based simulations fail and the resources needed for DNS are not available.

LES is based on the early mentioned hypothesis about turbulent structures (see section 1.4.1). The large scales are anisotropic and lose energy by transferring it to the small scales. The small scales work as energy dissipative. Therefore, the small scales effect has to be imposed in the simulation, otherwise the energy levels will keep growing and the simulation might fail, or produces invalid data. The small scales have isotropic nature, hence they are easy to be modeled.

In order to resolve for the large scales only, the subgrid scales have to be filtered. The filtering can be either explicit by means of convolution integral with the proper kernel function, or can be implicit by means of numerical discretization. In the following there are more details about both filters, the classical kernel functions, the proper cut-off width, and the so-called effective filter. At the end of this chapter the validation of the LES data is discussed. This chapter is the summery of the literature survey accomplished at the beginning of this work, in addition to personal explanations for many of the LES concepts. For further details about each of the

concepts discussed here the reader will be advised with a reference wherever it is possible.

2.1 Scale separation or filtering

It helps to understand the filtering process to think about the large eddies as large but slow fluctuations around a mean value (i.e. large amplitude and low frequency), and to think about the small eddies as small but very fast fluctuations about the large fluctuations. Now the flow field variables can be written as,

$$f(x, t) = \bar{f}(x, t) + f'(x, t) \quad (2.1)$$

This view makes it simpler to see the filtering processes in physical space as neglecting changes that happen in a length smaller than the smallest grid-scale (i.e with a cut-off width Δ larger than the largest subgrid scale), or in spectral space as neglecting fluctuations that happen with a frequency higher than that of the smallest grid scale (i.e with cut-off wave number k_c that is higher than highest grid scale wave number).

2.1.1 Explicit filtering “Convolution integral”

The convolution integral of f and h in physical space defined in Eq. (2.2) is the amount of overlap between f and h reversed and shifted, or in spectral space defined in Eq. (2.3) is the weighted amplitudes of the selected frequencies. Therefore, an intuitive idea was to implement the filter as a convolution product of the variable of interest and a proper filter function with the proper cut-off width, such that its overlap with the variable captures only the large scales. Hence, one can write the filtered field variable as in Eq. (2.4) for physical space filtering and as in Eq. (2.5) for spectral space filtering.

$$(f \otimes h)(t) = \int_a^b f(\tau)h(t - \tau) d\tau \quad (2.2)$$

$$(f \otimes h) = \hat{f}\hat{h} \quad (2.3)$$

$$\bar{f}(x, t) = \iiint \Pi_{i=1}^3 h(x_i - x'_i, \Delta_i) f(x', t) dx' dy' dz' \quad (2.4)$$

$$\tilde{f}(x, t) = \hat{f}\hat{h} \quad (2.5)$$

Filter properties Here, listed some important filter properties:

1. Linearity,

$$\overline{f_1 + f_2} = \bar{f}_1 + \bar{f}_2 \quad (2.6)$$

This property is matched automatically from the definition of the filter as a convolution product

2. Derivative-Filtering commutation,

$$\frac{\partial \bar{f}}{\partial x} = \overline{\frac{\partial f}{\partial x}} \quad (2.7)$$

this property is of high importance as it shows that; the solution of the differential equation to the filtered variable is equal to the solution of the filtered differential equation. Therefore, This property will be the base for filtering Navier-Stoks to resolve only for the filtered variables (i.e. Grid Scales).

3. Constants conservation,

$$\iiint h(t - \tau) dx dy dz = 1 \quad (2.8)$$

The filter should recover constants and uniform quantities.

4. Reynolds operator properties,

- The filtered product of the grid and sub-grid quantity vanishes identically,

$$\overline{\bar{f}f'} = 0 \quad (2.9)$$

- The operator is a projector,

$$\bar{\bar{f}} = \bar{f} \quad \text{and} \quad \bar{f}' = 0 \quad (2.10)$$

Filters in general don't have these properties, but special types like the sharp cut-off filter do.

Classical explicit filters Here are the classical well known filters and their kernel functions in spectral space:

- i) Sharp cut-off filter (see Fig. 2.1),

$$\hat{h}_i(k) = \begin{cases} 1 & \text{if } |k| \leq k_c \\ 0 & \text{otherwise} \end{cases}$$

- ii) Top-hat filter (see Fig. 2.2),

$$\hat{h}_i(k) = \frac{\sin(k\Delta/2)}{k\Delta/2}$$

(2.11)

- iii) Gaussian filter (see Fig. 2.3),

$$\hat{h}_i(k) = \exp\left(\frac{-\Delta^2 k^2}{4\gamma}\right) \quad (2.12)$$

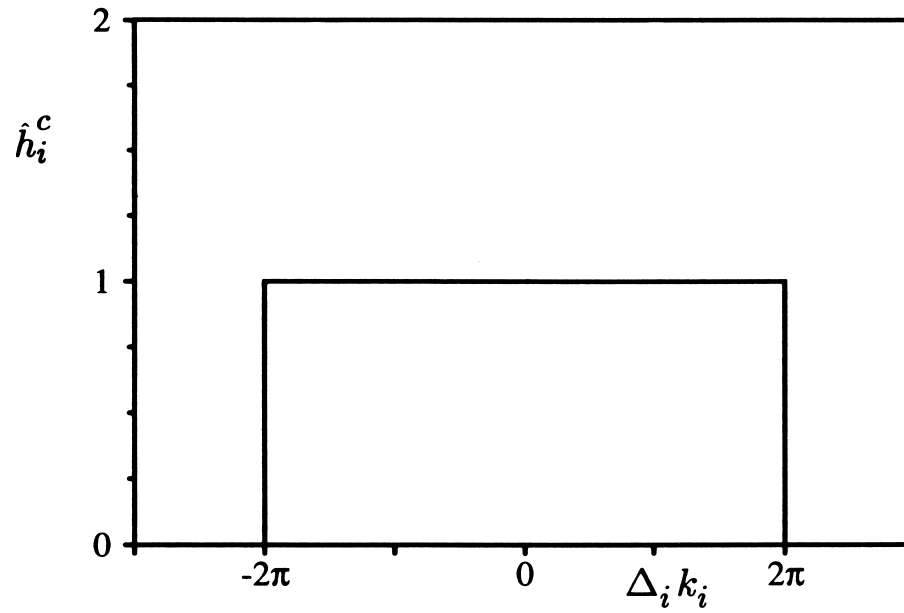


Figure 2.1: The sharp cut-off filter in spectral space. Source [2].

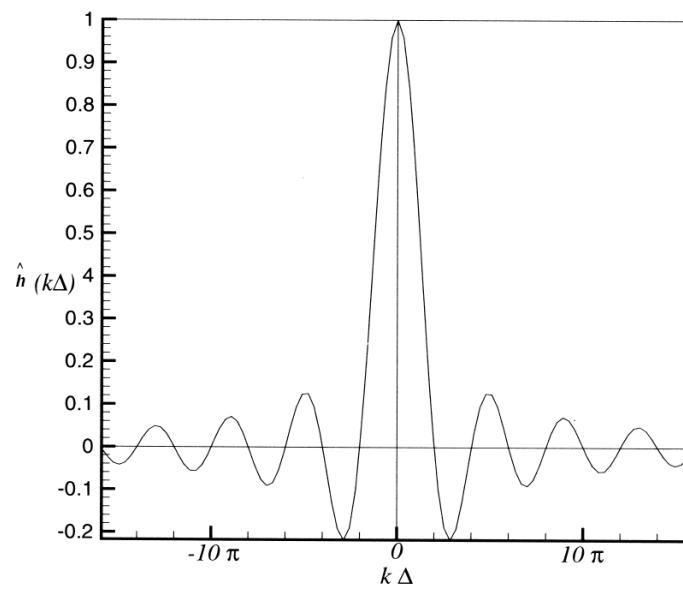


Figure 2.2: The top-hat filter in spectral space. Source [1].

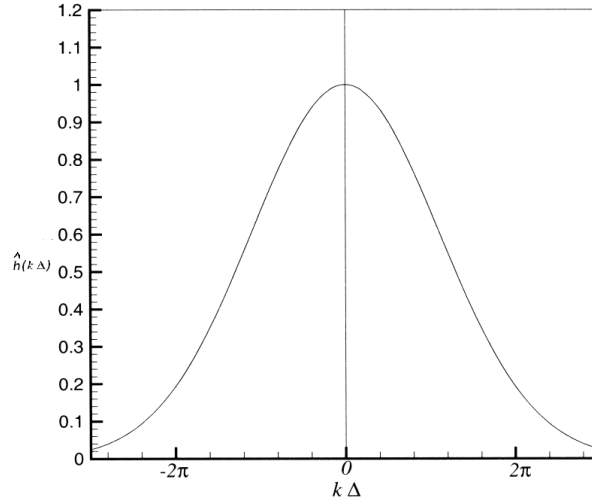


Figure 2.3: The Gaussian filter in spectral space. Source [1].

Where k is the wave number, k_c is the cut-off wave number, Δ is the cut-off width, and γ is a constant that scales the Gaussian distribution.

2.1.2 Implicit filtering “Discretization Effect”

Numerical discretization or equivalently replacing the infinite Fourier transformation with a finite transformation by neglecting high wave numbers, imposes an implicit filter. This filter is problematic as its cut-off length is extremely difficult or sometimes impossible to evaluate. Many solvers implementation for the filter is based on the implicit filtering by using a cors grid.

2.1.3 The effective filter

The effective filter is the resultant of both the explicit and implicit filtering, and subsequently its cut-off width is as well extremely difficult to evaluate, this difficulty is a result of having the discretization effect involved.

2.2 Filtered Navier-Stokes

According to the derivative-filtering commutation property shown in Eq. (2.7), the solution of Navier-Stokes to the filtered quantities is equivalent to the solution of the filtered Navier-Stokes.

2.2.1 Filtered Navier-Stokes

The application of the filter to Navier-Stokes equation results in,

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial}{\partial x_j} (\overline{u_i u_j}) = -\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial}{\partial x_j} \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right), \quad i = 1, 2, 3 \quad (2.13)$$

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0 \quad (2.14)$$

Eq. (2.13) is very similar to Navier-stokes for the filtered quantities, except for the nonlinear term $(\overline{u_i u_j})$, and in order to make it useful, one has to express this non-linear term as a function of \bar{u} and u' .

2.2.2 Leonard's decomposition

According to Leonard, the non-linear term can be decomposed as the following,

$$\begin{aligned} \overline{(u_i u_j)} &= \overline{(u'_i + \bar{u}_i)(u'_j + \bar{u}_j)} \\ &= \overline{(u'_i u'_j)} + \overline{(u'_i \bar{u}_j)} + \overline{(u'_j \bar{u}_i)} + \overline{(\bar{u}_i \bar{u}_j)} \\ &= \overline{(u'_i u'_j)} + \overline{(u'_i \bar{u}_j)} + \overline{(u'_j \bar{u}_i)} + \overline{(\bar{u}_i \bar{u}_j)} \end{aligned} \quad (2.15)$$

Now τ the sub-grid stress tensor can be defined as,

$$\tau_{ij} = \overline{(u_i u_j)} - (\bar{u}_i \bar{u}_j) \quad (2.16)$$

$$\tau_{ij} = \overline{u'_i u'_j} + \overline{(u'_i \bar{u}'_j + u'_j \bar{u}'_i)} + (\bar{u}_i \bar{u}_j - \bar{u}_i \bar{u}_j) \quad (2.17)$$

$$\tau_{ij} = R + C + L \quad (2.18)$$

Where R is the Reynolds sub-grid stress tensor that represents the interaction between subgrid scales, C is the cross stress tensor that represents the interaction between the grid and sub-grid scales, and L is the Leonard stress tensor that represents the interaction between the grid scales .

Now the Filtered Navier-Stokes can be rewritten as the following,

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial}{\partial x_i} (\bar{u}_i \bar{u}_j) = -\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial}{\partial x_j} \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) - \frac{\partial \tau_{ij}}{\partial x_j}, \quad i = 1, 2, 3 \quad (2.19)$$

2.2.3 The closure problem

For the filtered field in three-dimensional flows, there are four independent equations governing the velocity field; the three components of the filtered Navier-Stokes, and the continuity equation. Thus the number of unknowns is greater than the number of equations. This set of equations is called unclosed, and it cannot be solved unless the subgrid stress tensor τ is determined.

Since LES is used to reduce the degree of freedom of the solution by resolving the large scales only, the information of the sub-grid scales are lost and their effect is grouped in the subgrid stress tensor. Hence, the subgrid scales effect modeling should consist of approximating the coupling terms on the basis of the information contained on the resolved grid.

2.3 Smagorinsky model

After the survey of the mathematical, functional and structural models was accomplished, it was decided to implement the Smagorinsky eddy viscosity model within the activities of this thesis. The following sections explain the base hypotheses and the model structure.

2.3.1 Interaction between grid and sub-grid scales

In the case of fully developed isotropic turbulence, the description of the scales interactions is reduced to that of the kinetic energy transfer. These transfer mechanisms were analyzed and the results motivated for the following hypotheses; energy cascade and eddy viscosity.

Energy cascade There are two energy transfer mechanisms between the large and small turbulence structures (see Fig. 2.4):

- A strong drainage of energy from the large scales by the sub-grid scales (forward energy cascade phenomenon).
- A weak feedback of energy to the resolved scales (backward energy cascade phenomenon).

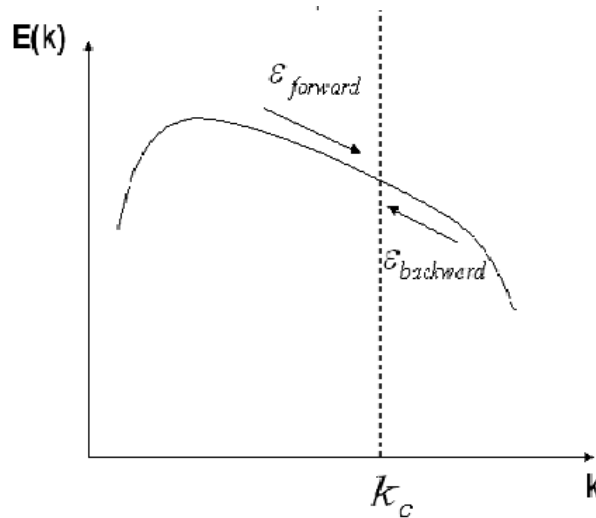


Figure 2.4: The forward and backward energy cascades.

Eddy viscosity The energy transfer from the resolved to the subgrid scales is analogous to the molecular mechanisms represented by the diffusion term in which the viscosity appears. As a result the effect of the subgrid scales can be modeled either:

- Explicitly by a term that has a mathematical form similar to the sheer stresses, or
- Implicitly by means of numerical dissipation.

2.3.2 The model structure

In the Smagorinsky model the effect of the subgrid scales is imposed explicitly by means of additional subgrid scale viscosity ν_{sgs} . Thus the subgrid stress tensor can be written as,

$$\tau_{ij} = -\nu_{smag} \bar{S}_{ij} \quad (2.20)$$

where the subgrid scale viscosity is defined as;

$$\nu_{smag}(x, t) = (C_{smag} \Delta)^2 (2 \|\bar{S}(x, t)\|^2)^{1/2} \quad (2.21)$$

where Δ is the filter width, C_{smag} is the Smagorinsky constant that varies according to the flow scenario between 0.1 and 0.2, and \bar{S} is the strain rate of the filtered quantities defined as,

$$\bar{S}_{ij} = \frac{1}{2} \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) \quad (2.22)$$

by definition the subgrid viscosity is a space and time dependent. Thus, Navier-Stokes with the eddy viscosity model implanted can be written as,

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial}{\partial x_j} (\bar{u}_i \bar{u}_j) = -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} ((\nu + \nu_{smag}) \bar{S}_{ij}) \quad (2.23)$$

2.4 Validation of LES data

After the literature study of the LES basic concepts and models was accomplished, the validation mechanisms for the simulation data had to be designed. This step was performed in an early stage so that everything would be ready to easily test and validate the implementation.

This section consists of two main parts which are the validation techniques, and the description of the channel flow scenario that was selected as a benchmark. The former explains many aspects like how to set-up the validation criterion, equivalence classes, what quantities to be compared, and finally how LES and DNS data have to be manipulated to have them in a comparable form.

The latter describes the test case scenario, its assumptions, exhibited features, how to run similar simulations, and finally what quantities can be extracted and should be used for validation. Also the similarity concept or non-dimensionalization of variables is discussed.

2.4.1 Validation techniques

There are mainly two common validation techniques to test a given combination of filter, model, and numerical scheme, these are the Prior testing, and the Posterior testing¹.

Prior testing This test allows testing various hypotheses and models.

Description In this test there is no simulation run. The full field DNS data is filtered analytically resulting in an exact large structure field \bar{u}_{exact} . Then the exact subgrid scales contribution can be easily calculated from,

$$u'_{exact} = u_{exact} - \bar{u}_{exact} \quad (2.24)$$

Afterward the proposed model is used to calculate the sub grid stresses from the exact filtered velocity field \bar{u}_{exact} . On the other hand the exact sub grid stresses are

¹For more detail about the validation techniques the book of Pierre Sagaut [1] is recommended.

calculated from the subgrid velocity field u'_{exact} . Finally, the exact subgrid stresses and these calculated using the model are compared to validate the model.

Drawback

- The full field information of the DNS should be available, as a result extremely large storage is required.
- It does not take into account the effect of the implicit filter and the truncation errors, as there was no real simulation.
- Prior testing has proved disappointing results, because the exact and modeled stresses of many models turned to be poorly correlated, while these models were successful using posterior testing.

Posterior testing

Description The experiment or the case is simulated using a complete LES code, the statistics is to be collected during the run, and then compared with the reference statistics.

Drawback If LES results agree with these of DNS, there will not be much understanding about the reasons of the agreement, or disagreement of such results. This is simply because of two reasons:

- If there was something wrong in the results, the test information is too poor to decide what was wrong (the complex code, the numerical scheme, the filter, or the model),
- If the results are fine, then that might be a result of some compensating errors for that scenario, but there is no proof that this combination might work fine for different scenarios.

What to compare

In LES the large turbulent structures are resolved while the information of the small scales is lost, as a result point-to-point consistency between LES and DNS data shouldn't be expected. But what actually LES results should recover correctly is the large structures statistics.

Equivalence classes

Equivalence classes are statistical classes (i.e. moments) of the turbulent quantities usually of the first and second order, and rarely of the third order. The main purpose of setting up these classes is to have quantities that are sufficient to prove the equivalence of the velocities and stresses of two different simulations. It is of great importance to mention that direct comparison of LES statistics with DNS statistics is theoretically incorrect. This incorrectness can be easily understood by looking to the average of the full field DNS information shown in Eq. (2.25). The average of the DNS data is the sum of the average of the grid and the subgrid scales, while the LES average data is the average of the large scales only. This problem is getting more complicated in the stresses averaging.

$$\langle u_{DNS} \rangle = \langle \bar{u} \rangle + \langle u' \rangle \quad (2.25)$$

A correction is done by either bringing LES to the form of the DNS information by reconstructing a full field out of the large structures “De-filtering”, or the other way around by filtering the DNS full field.

Data manipulation

As mentioned above, it is theoretically incorrect to directly compare the DNS statistics with these of LES. Thus data manipulation for one of them should be done first to bring it to a form that is consistent with the other. The next section describes the manipulation of the DNS data to reduce it to a large structure information by means of filter. Only the latter is discussed here.

DNS data manipulation As researchers has different preferences regarding the selection of the filter, it makes no sense to provide them with already filtered reference data. Hence, mainly unfiltered data only is available. The unfiltered reference data has to be then filtered using the LES simulation filter. Based on the fact that the filtering and averaging commute, a great reduction of storage needed can be achieved by providing directly the averaged quantities instead of the full field. For general flows, or in the case of inhomogeneous and anisotropic flows, a heterogenous filter with variable cut-off length is required, so to construct a filtered statistical profiles, the two-point correlation function of the reference data should be provided. This information is of high dimension and very expensive to store. In the special cases of the homogeneous and isotropic flows only the single point correlation function is required.

What is done in practice

In practice it is very difficult to filter the DNS data, in addition many researchers use the discretization as a filter where the effective cut-off length is not known implicitly. Thus almost everybody directly compares the LES statistics with these of the DNS

Quantities to be compared

An LES simulation should recover the large turbulent structures mean flow velocities, energy budgets, and stresses including normal, viscous shear, and Reynolds's stresses².

2.4.2 Benchmark

The channel flow was selected as a benchmark to validate the results of the LES implementation within the Peano. This choice is due to its practical importance, and reference data availability. The next sections are furnished to demonstrate the scenario and its setup.

²Reynolds's stresses are the stresses arises from the velocities fluctuations

The channel flow

The channel flow is the scenario at which the fluid flows through a rectangular duct (see Fig. 2.5). The scenario assumptions and the flow features are described here.

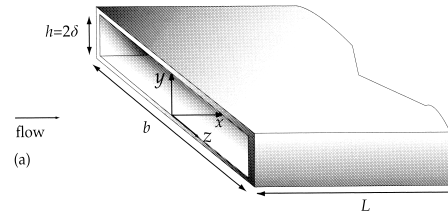


Figure 2.5: The channel geometry. Source: [3].

Assumptions The duct is assumed to be very long³ with a very large aspect ratio⁴. As a result of these two assumptions the dominant mean flow is in x direction, and the flow statistics are independent in z direction. The simulation is assumed to be performed at sufficiently large x to have a fully developed turbulence with no entrance effects. Thus the flow statistics are can be considered x independent. The flow then can be considered statistically one-dimensional, where the flow statistics depend only on y direction. If the simulation domain is designed properly such that the periodicity is assured on the two statistically homogenous directions x and z , the periodic boundary conditions should be imposed.

Flow features The analysis of the mean governing equations in the light of the previously mentioned assumptions⁵ is recommended. resulting in the following flow features,

³ The duct is considered very long when $L_x/\delta \gg 1$, where $L_x/$ is the duct length, and δ is the half of the duct height

⁴ The duct aspect ration considered very large when $b/\delta \gg 1$, where b is the duct width

⁵For detailed discussion about the balance of mean forces in such a channel flow scenario the book of Pope [3]

- Uniformity of the mean axial pressure gradient,

$$\frac{\partial \langle p \rangle}{\partial x} = \frac{dp_w}{dx} \quad (2.26)$$

Eq. (2.26) shows that the mean axial pressure gradient is uniform across the flow, where p is the axial pressure, and p_w is the mean axial pressure at the wall

- Linearity of the pressure and shear stress,

$$\frac{d\tau}{dy} = -\frac{dp_w}{dx} \quad (2.27)$$

Where τ is the total shear stress defined as the sum of the viscous and the Reynolds's stresses. Eq. (2.27) implies that in a fully developed turbulent in channel flow, the normal pressure gradient is in balance with the total shear stress gradient.

It also implies that the normal pressure and the total shear stress gradients are constants, and that is because τ depends only on y and p_w depends only on x . Thus one can write the total shear stress as a function of the wall shear as followed,

$$\tau(y) = \tau_w \left(1 - \frac{y}{\delta}\right) \quad (2.28)$$

One can also evaluate the necessary pressure gradient to impose a certain wall shear, or the other way around as following,

$$\frac{dp_w}{dx} = -\frac{\tau_w}{\delta} \quad (2.29)$$

Reference data For the channel flow there are two reference data sources that have been used here; (Mansour, Moser and Kim) [8], and Tokyo database[9]. The two sources have slight differences in the statistics they offer. The Tokyo database offer the velocity root mean square profiles in addition to the mean velocity filed. Mansour, Moser and Kim offer the velocity variance.

Similarity concept and universal profiles for the channel flows In order to get universal profiles for the flow variables of similar flows⁶ and to be able to run similar flow scenarios to these of the reference data, the flow quantities have to be non-dimensionalized and the characteristic non-dimensional numbers have to be respected (e.g. Reynolds number). An example for understanding the universal profile concept can be illustrated by dividing Eq. (2.28) by the wall shear, that results in,

$$\frac{\tau(y)}{\tau_w} = \left(1 - \frac{y}{\delta}\right) \quad (2.30)$$

Eq. (2.30) shows that the total shear divided by the wall shear which is a flow specific results in a quantity that is not a flow specific but has a universal character. Analogously, many other flow quantities might be non-dimensionalized using the proper scaling quantities. Some of the flow quantities have only universal profiles that are restricted on some characteristic non-dimensional numbers. Thus, these characteristic non-dimensional numbers have to be grouped and respected by the simulation to run a similar scenario. Listed below the important scaling quantities for the channel flow.

- The velocity scale,

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \quad (2.31)$$

u_τ is called the friction velocity, and ρ is the density. u_τ is used to non-dimensionalize the velocities to get universal velocities profile.

- The viscous length scale,

$$\delta_\nu = \frac{\nu}{u_\tau} \quad (2.32)$$

is used to non-dimensionalize the wall distance y resulting in,

$$y^+ = \frac{y}{\delta_\nu} \quad (2.33)$$

mainly all the statistics are plotted versus the wall units y^+ .

⁶For detailed discussion about similarity concept and similar flows [6] is recommended.

Finally to have similar channel flow scenarios and consistent statistics profiles to these of the reference data, the simulations has to be conducted at friction Reynolds number Re_τ that is identical to these of the reference data. Re_τ is defined by,

$$Re_\tau = \frac{u_\tau \delta}{\nu}. \quad (2.34)$$

Chapter 3

Basics of Peano

Since the thesis work is about integrating an LES turbulence solver to the Peano framework, this chapter is furnished to familiarize the reader with Peano. The Peano and its main features are discussed lightly with more emphasis on the fluid component.

3.1 The framework Peano

Peano is not just a solver, it's an object oriented PDE framework. It's written in C++, but that is not the only reason why it's object oriented, it's so because many aspects of object oriented terminology were employed (e.g. hierarchy, inheritance, etc.).

As Peano is a PDE framework it allows adding applications that are based on solving PDE systems. Currently Peano has two applications, Poisson's equation solver and CFD component. Within the CFD component there are two subcomponents, which are the steady state-solver and the turbulent solver.

3.2 Features

The Peano framework is an optimization aware, efficiency aware, usability aware, and hardware aware implementation. These advantages were achieved by means of the

features shown in Fig. (3.1). These features are explained in more details in the following sections.

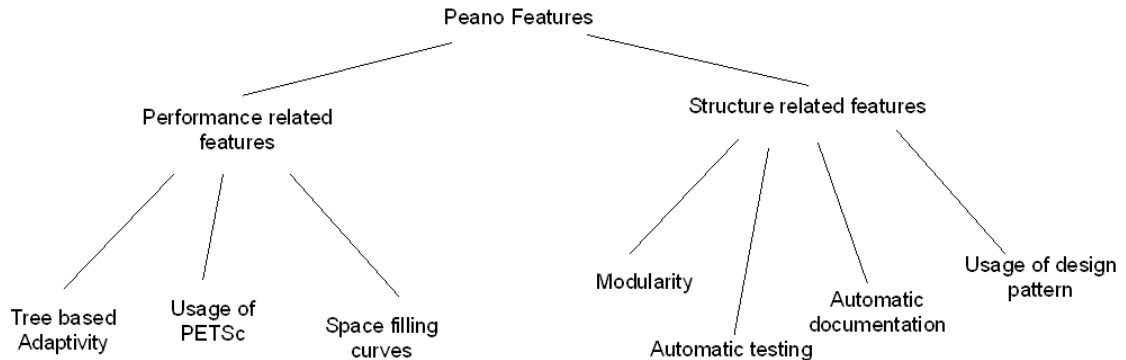


Figure 3.1: Peano Framework Features.

3.2.1 Performance related features

Tree based adaptivity Peano is working on Cartesian grids, both regular and tree based adaptive grids. The adaptive part uses the space tree approach.

Space filling curves The Peano framework employs the space filling Peano curve for traversing the adaptive tree. Together with the stack data structure concept¹ they assure the dynamic load balancing, and a compromise between achieving locality in accessing memory locations (i.e. cache optimization), and good reduction in communication overhead.

The usage of PETSc Peano uses PETSc² library, because PETSc has large variety of linear solvers and preconditioners. The use of PETSc solver is temporary until the chair fully develops its geometric multigrid solver.

¹For more details regarding the Peano curve and the stack data structures [13] and [14] are recommended.

²For more information about PETSc, see PETSc webpage [10].

3.2.2 Structure related features

In addition to utilizing hierarchy, inheritance, and polymorphism, many other object-oriented concepts were employed in Peano to fulfill the modern software requirements. These structure related features are listed below.

Modularity The code is built in component-wise structure, which enables installing only the components that are really needed. The component-wise structure implies modularity, which means that modifying a component has no influence on the other components as long as the interface is persevered. As different components might need to perform different actions on the grid, the adapter concept was employed in Peano. The adapter concept is used to redirect the grid traversal to a specific actions.

Design patterns Design patterns³ are employed in Peano as one way to allow maintainability, extendability, and to avoid code duplication.

Automatic testing All functionalities implemented in Peano are tested on both unit and integration levels. The unit and integration tests are automated (i.e. One button press enables testing all installed Peano components).

Automatic documentation Most of the Peano source code is automatically documented by using doxys⁴.

3.3 CFD component

CFD is one of the two current applications integrated within the Peano framework. Here are listed the most important features of this component.

Space and time discretization The CFD component uses d-linear finite element discretization. Both regular and adaptive grids are supported. The numerical scheme

³For more details regarding design pattern [5] is recommended.

⁴More details about doxys can be found on doxys webpage [12].

is semi-explicit⁵, where the pressure Poisson equation is solved implicitly and the velocity update is done explicitly.

The parallelization status Currently the parallel running of the CFD component is not yet fully implemented, but work is undergoing on this side.

Restart mechanisms A full restart capability for both regular and adaptive grids were implemented within the activities of this thesis. It allows both restart from time zero and continue run options. The restart implementation included checkpointing.

Subcomponents

- steady state
Uses non-linear implicit solver and this is to be extended to use it in a fully implicit time scheme.
- turbulence
After this work was done, an LES based turbulent component is currently available, with the averaging capability needed for validating the results.

⁵Details regarding semi-explicit schemes can be found in the book of J.Chorin [15].

Chapter 4

Implementation Aspects

This chapter aims to introduce the reader to the work accomplished in this thesis. The chapter is not meant to be a documentation for the code ¹. Nevertheless some lines of code are included wherever it was thought to be necessary.

Objective The objective of this work was to implement an LES solver based on Smagornisky model into the Peano framework. In order to achieve this objective many tasks had to be performed. Some tasks needed only to extend an existing functionality, others needed to add completely new ones. Here are the major tasks, and in the following sections come more details about them.

Major tasks

- Enlarging the CFD component to accommodate the turbulence subcomponent.
- Implementing the statistical averaging.
- Implementing the restart mechanism including checkpointing.
- Implementing the Smagornisky model including the Van Driest damping.

¹Full documentation of the implemented classes, methods, etc. can be found on the Peano's documentation webpage [11]

These tasks were first performed for the regular² grid, but at the early stage of numerical simulations it was found that the resolution near the wall is not sufficient. To afford the sufficient resolution on the trivialgrid that has only cubes with no stretching, in addition to the fact that the code is still running only in serial, one simulation would take months³. Thus, each of this major tasks were extended for the Peano's adaptive grid⁴, to enable the refinement near the wall hoping that this would give a sufficient resolution to instantiate turbulence and preserve it.

4.1 Enlarging the CFD component

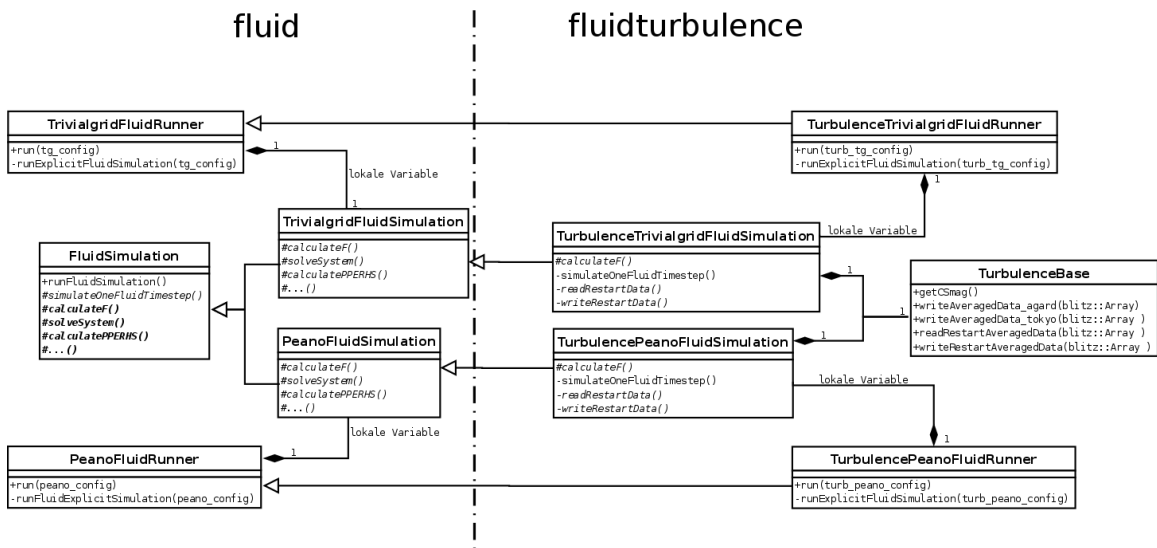


Figure 4.1: The extension of the fluid component to accommodate the turbulence subcomponent

The extension of the fluid component to accommodate turbulence (Fig. 4.1) included tasks like extending the configuration file, implementing the periodic boundary conditions, enabling the domain initialization with a parabolic profile, including the

²The Peano's regular grid is called trivialgrid

³More detail about the numerical simulations are discussed in a separate chapter

⁴The Peano's adaptive grid is called Peano

random noise, and creating the turbulence simulation classes and the turbulence runners for both trivial and adaptive grid.

The configuration class and configuration file were extended to have the turbulence sub-tag for the quantities needed in the turbulence simulations. This was mainly done by introducing a subclass turbulence configuration.

The periodic boundary conditions were implemented for both regular and adaptive grids to enable the periodicity in x or z directions or in both of them simultaneously. The structure of the fluid component was extended by creating the `TurbulenceTrivialgridFluidRunner`, and the `TurbulenceTrivialgridFluidSimulation` classes, that inherit from the `TrivialgridFluidRunner`, and `TrivialgridFluidSimulation` respectively. The same was done for the adaptive Peano grid (see Fig. 4.1). In addition to these tasks many other minor and rather technical tasks were performed to extend the CFD component.

4.2 Statistical averaging

Since LES data cannot be compared in a point-to-point comparison with DNS, having statistical data is the only chance to really validate the results.

Objective

The objective of this task was to implement the functionality of collecting the statistics that are averaged in both space and time for the channel flow simulations. There are two main types of statistics of interest, which are the first order moments (i.e. mean values), and the second order moments. The latter had to be generated in both groups of Tokyo [9], and Mansour, Moser and Kim [8].

General idea of the averaging in space and time

Whether it is a measure of central tendency or dispersion, or it is an averaging on the regular or the adaptive grid, they all have the same general idea for the implementation.

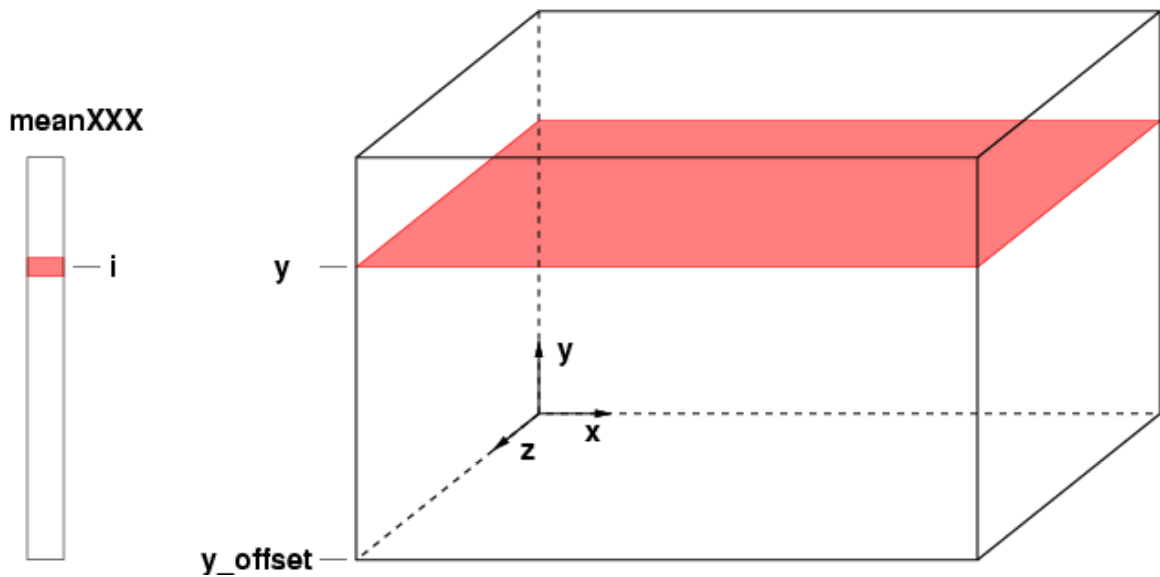


Figure 4.2: General idea of space averaging.

General idea of space averaging While traversing over the grid vertices that have the velocity data, each vertex on a y -layer adds its contribution to the average data array⁵, specifically to the array location of the y -layer index (see Fig. 4.2). The same idea is applied for the pressure data lies on the cells centers while the cells traversal. As a result, the information on each y -layer is reduced to a point on the y axis. For the adaptive grid, the data that lies on one of the two adjacent periodic boundaries is excluded (see Fig. 4.3), this has to be applied for the trivialgrid, but is not done yet at the moment.

In the current implementation the averaging in space is restricted only to the case of the two statistically homogenous directions in x and z .

General idea of the time averaging The time averaging is either done explicitly, or implicitly depending on the quantities of interest. In explicit time averaging, at each time step the current space averaged data saved to a temporary array is averaged in time with the time averaged data of the previous time step (see Fig. 4.4). The two arrays have to be averaged with the proper weighting factor. The weighing factors

⁵The grid traversal is not performed layer by layer in y -direction

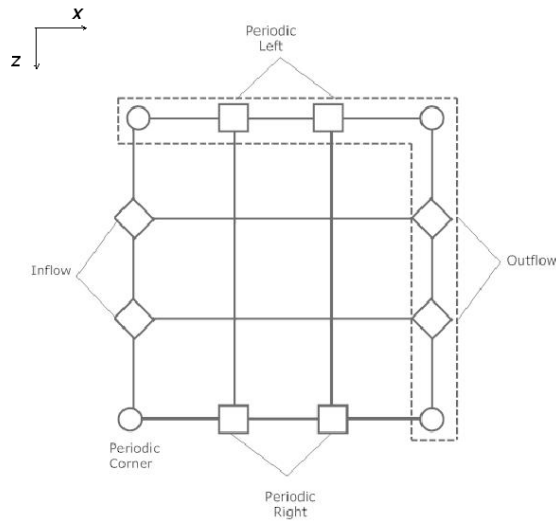


Figure 4.3: The data that lies on one of the two adjacent periodic boundaries is excluded from averaging.

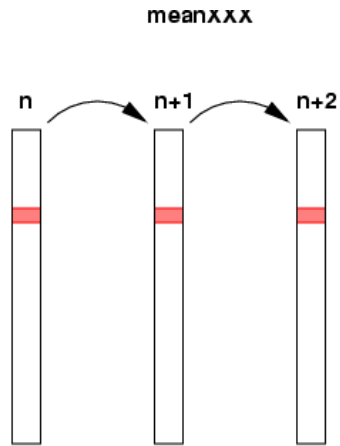


Figure 4.4: The data that lies on one of the two adjacent periodic boundaries is excluded.

are evaluated to satisfy,

$$\text{The time average of } Q = \frac{1}{N_t} \sum_{i=1}^N Q_i \tag{4.1}$$

Where Q is any space averaged statistic.

In implicit time averaging, the space averaging is combined with the time averaging simultaneously while traversing the grid. This saves the use of temporary arrays.

4.2.1 The averaging on the regular grid

The mean velocity and mean pressure are averaged in space on the regular grid according to,

$$\langle u \rangle = \frac{1}{N} \sum_{i=1}^N u_i, \quad (4.2)$$

and

$$\langle P \rangle = \frac{1}{N} \sum_{i=1}^N P_i \quad (4.3)$$

respectively. The velocity and the pressure RMS is calculated according to,

$$U_{rms} = \sqrt{\frac{\sum_{i=1}^N (\langle u \rangle - u_i)^2}{N}}, \quad (4.4)$$

and

$$P_{rms} = \sqrt{\frac{\sum_{i=1}^N (\langle P \rangle - P_i)^2}{N}} \quad (4.5)$$

respectively. Finally the Reynolds's stresses is calculated according to,

$$u_1 u_2 = \frac{\sum_{i=1}^N (\langle u_1 \rangle - u_{1i})(\langle u_2 \rangle - u_{2i})}{N} \quad (4.6)$$

$$u_1 u_3 = \frac{\sum_{i=1}^N (\langle u_1 \rangle - u_{1i})(\langle u_3 \rangle - u_{3i})}{N} \quad (4.7)$$

$$u_2 u_3 = \frac{\sum_{i=1}^N (\langle u_2 \rangle - u_{2i})(\langle u_3 \rangle - u_{3i})}{N} \quad (4.8)$$

where N is the total number of cells per y-layer for the pressure averaging, and the total number of the vertices per y-layer for the velocity averaging.

I The implementation of the computations of the mean values:

Since the averaging needs grid traversal, a new adapter for each type of the average statistics was created. For the computation of the mean values a new adapter called `TrivialgridEventHandle2AverageTurbulentDataAdapter` was created, and the method `simulateOneFluidTimestep()` of the subclass `TurbulenceTrivialgridFluidSimulation` was extended to call the `iterate` method of the grid⁶ on the `TrivialgridEventHandle2AverageTurbulentDataAdapter` adapter, this call comes after the call of `simulateOneFluidTimestep()` of the base class `TrivialgridFluidSimulation`. Since the pressure information lies on cells, the implementation of the pressure averaging Eq. (4.3) was done within the method `HandleElement()` of the `TrivialgridEventHandle2AverageTurbulentDataAdapter()` (Listing 4.1 shows the actual implementation of the method)

The velocity data lies on the vertices, thus the implementation of the velocity averaging Eq.(4.2) was done within `touchVertexLastTime()` (see Listing 4.1). The time averaging was done implicitly, and the averaged data is weighted within the method `endIteration()` at the end of the traversal to have them ready to receive the space and time contributions of the next time step (see Listing 4.1).

II The implementation of the computation of the second order moments:

A new adapter called `TrivialgridEventHandle2AverageRmsTurbulentDataAdapter` was created. The `simulateOneFluidTimestep()` of the subclass `TurbulenceTrivialgridFluidSimulation` was extended to call the `iterate` method of the trivial grid on the adapter `Trivialgrid-EventHandle2AverageRmsTurbulentDataAdapter`. The call had to be allocated after the mean value calculations, because it is needed for the standard deviation calculations. For the same reason the `TrivialgridEventHandle2AverageRmsTurbulentDataAdapter` had to implement two setters for the mean velocity and mean pressure data to enable the second moments calculations. The pressure RMS computation Eq.(4.5) is implemented within the adapter's `handleElement()` by accumulating the the square of the deviation at the cell center to the corresponding location of temporary array. At the end of the iteration the square root is taken on the whole temporary array, then it is averaged in time explicitly with the old pressure RMS data. The same is

⁶The `iterate` method is responsible only about traversing the grid while the traversal is directed to a specific action according to the adaptor passed as a parameter.

done for the velocity RMS and Reynolds stresses except for that the square root is not taken for the Reynolds stresses, and the implementation is done within the adapter's `touchVertexLastTime()` (see Listing 4.2).

4.2.2 The averaging on the adaptive grid

The averaging in space and time on the adaptive grid follows the same general concept (see section 4.2), but is more complicated, because of the variable mesh sizes that had to be respected on the vertices and the cells while computing their contribution. There are two main differences between the implementation of the averaging on the regular and adaptive grid due to the variable mesh sizes; the use of standard sets to store the average data, and the variable space weighting.

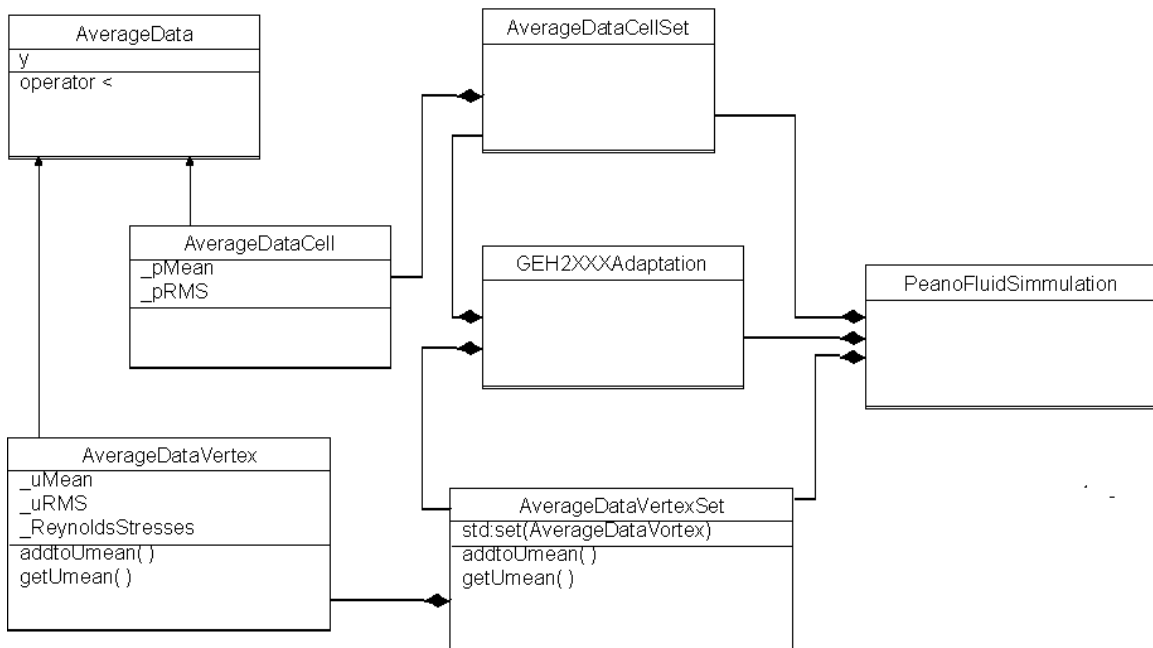


Figure 4.5: The class diagram for the created Average data types and the wrappers for the standard sets.

The use of sets to store the average data In the adaptive grid the index of a y -layer cannot be computed directly by dividing the y -position by the mesh size in

y-direction. Thus, a new idea had to be implemented based on using the standard sets of two classes, the `AverageDataCell` and the `AverageDataVertex`. Then other two classes were created to work as a wrappers for the sets, the `AverageDataCellset`, and `AverageDataVertexset`(see Fig. 4.5).

The classes `AverageDataCellSet` and `AverageDataVertexSet` are sorted sets, where the sorting done by internal methods using y coordinate. One big advantage of this implementation is that the data sets are accessible by the adapters. Thus, there is no need for implementing setters within the adapter.

The variable space weighting In the regular grid implementation the space weight was taken uniformly as $\frac{1}{N}$ for all the vertices and cells lie on all y -layers. This is not valid for the adaptive case, thus in the adaptive grid implementation, the weight was implemented as $\frac{A_i}{A}$ where A is the y -layer area, and A_i is the area that the data represent on y -layer area, defined as,

$$A_i = h_x \times h_z \quad (4.9)$$

where h_x , and h_z are the local cell mesh sizes in X and z directions respectively.

Besides these two differences that had to be respected, there were no other main differences of the implementation except for technical issues related to the differences between the adaptive and regular grid implementations.

4.3 Restart Mechanism

The main objective of implementing the restart mechanism (see Fig. 4.6) is to enable the capability of continuing a run after a planned stop. Then the implementation was extended by means of checkpointing (see Fig. 4.7) to allow quick recovery of a recent state of the simulation if an unplanned stop occurs(e.g. simulation crash). The implementation is not restricted to the turbulence subcomponent, but serve for the fluid component in general.

The restart mechanism offers two possibilities, either to continue the simulation

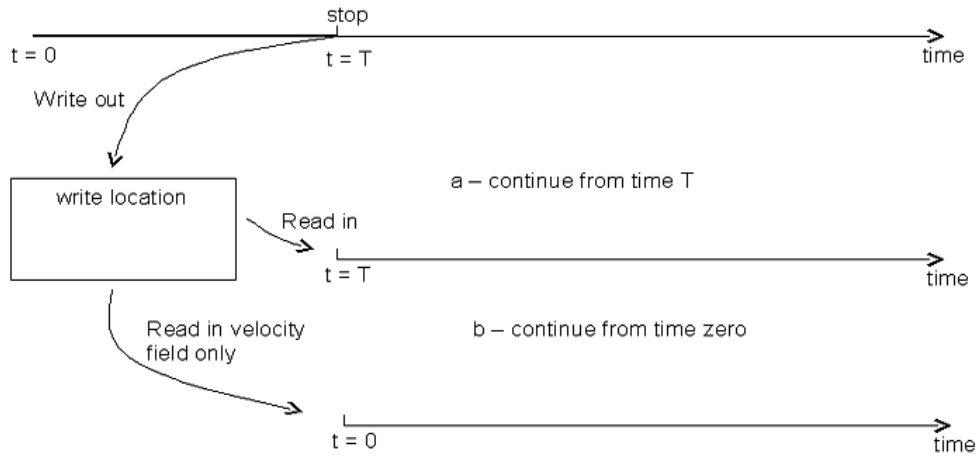


Figure 4.6: Schematic drawing for the idea of the restart capability.

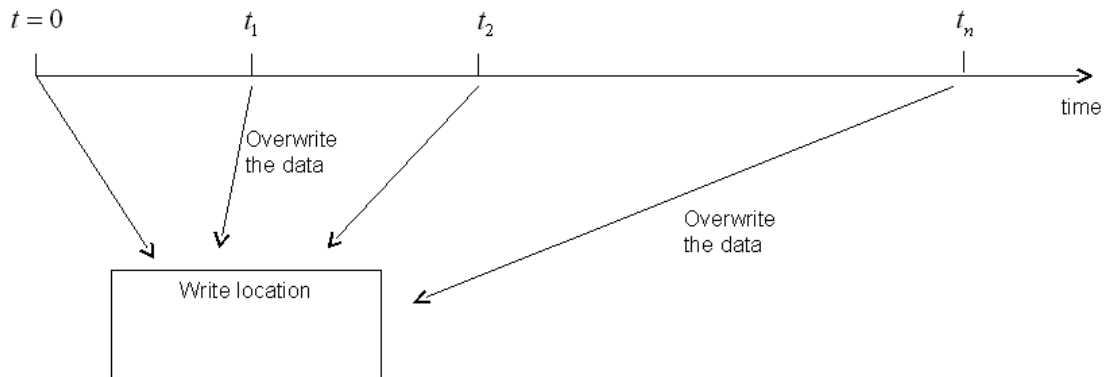


Figure 4.7: Schematic drawing for the idea of the checkpointing.

from the time it was stopped at, or to continue from time zero. The latter has more importance for turbulence simulations than the other flow scenarios, because there might be a need to continue out with no statistical history or to change some parameters to steer the simulation.

The restarts mechanism consists of two main parts, writing out the data and reading it in. There are three type of data that are written out or read in; namely the velocity field data in binary format, the configuration ⁷ as xml file, and the average data needed for the results validation as text. The pressure data is not written because it can be easily recovered after a few iterations. Writing out and reading in the average data for the regular and the adaptive grids is identical, because at the end both produces vectors of the statistics that are of the same type. This was one reason to create a central class named `TurbulenceBase` (see Fig. 4.1) to implement the average data manipulation to avoid code duplication. The averaged data vectors are passed to the base class write or read methods to preform the necessary actions. The average data is written out into two different files for the two different groups of statistics that are consistent with these of Tokyo data base[9] , and Mansure, Moser, and Kim[8] to enable the comparison with both of them. The checkpointing implementation has a weak point, that the new data overwrites the old. Thus, in the highly unlikable case were the simulation crashes during writing out the data, all the simulation data might be lost.

4.4 LES Implementation Details

4.4.1 The filtering

The filtering of the Navier-Stokes is done implicitly by using a coarse mesh for the discretization.

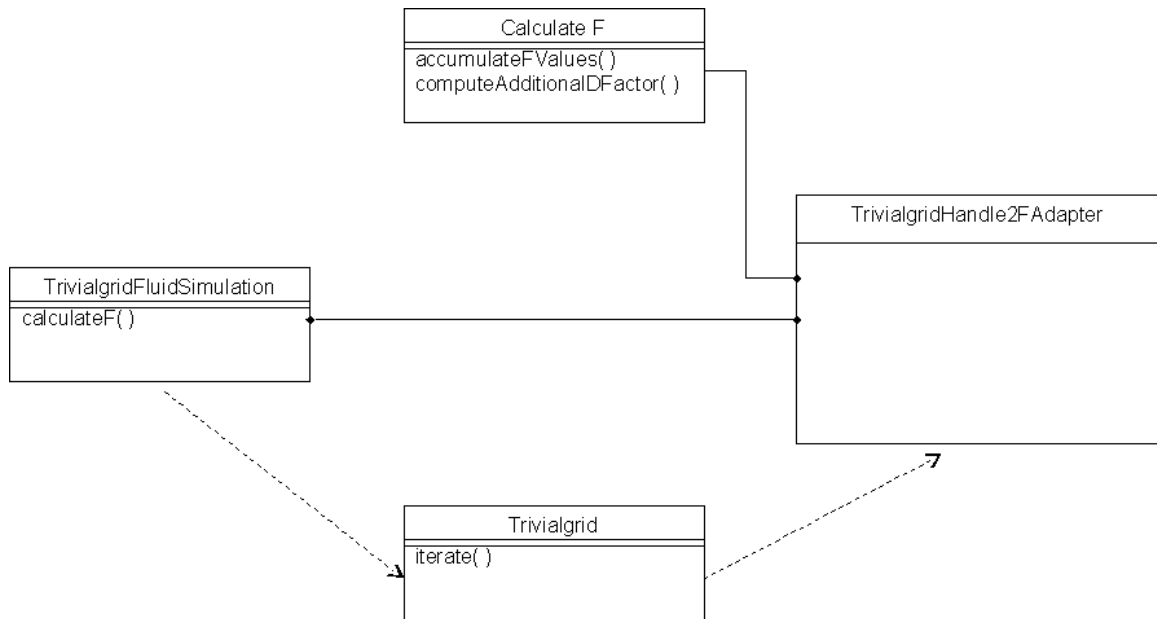


Figure 4.8: The original implementation of the diffusion operator computation.

4.4.2 The Smagornisky model

Since Navier-Stokes is filtered implicitly, the only difference between the implementation of filtered and the unfiltered equations for the Smagornisky model is adding the subgrid viscosity ν_{smag} that appears in the diffusion term. Thus, the implementation of the Smagorisky model was divided into two main tasks, which are extending the implementation of the diffusion operator computation to accommodate the new subgrid viscosity ν_{smag} , and the computation of ν_{smag} itself.

To enable the understanding of the extensions done to the old implementation (see Fig. 4.8) of the diffusion operator computation, the old implementation is described first on an abstract level.

The method `calculateF()` lies within the class `TrivialgridFluidSimulation` is implemented to enable the computation of the diffusion and convection operator at the centers of the grid cells. `calculateF()` mainly call the grid `iterate` method of the

⁷The read and write of the configuration is of importance to enable checking geometry and grid consistency with these read from the configuration file of the new run.

regular grid with `TrivialgridEventHandle2FAdapter` as a parameter ⁸.

The adapter `TrivialgridEventHandle2FAdapter` is constructed with an object of a class `CalculateF` as an adaptee. The adapter redirects the method `handleElement()` to a call of the method `accumulateFValues()` of the adaptee class. `accumulateFValues()` mainly computes and adds the convection and diffusion contribution of the vertices on the cell corners to the cell center. Within the `accumulateFValues()` lies the method `computeCandD()`. `computeCandD()` is the method at which the kinematic viscosity is used to compute the diffusion operator, and actually at which the ν_{smag} had to be added to the kinematic viscosity.

The extension of this implementation (see Fig. 4.9) was done by creating a subclass `LESCalculateF` of the super class `CalculateF` at which the subgrid viscosity is computed. The super class `CalculateF` was also extended by adding a new virtual method `computeAdditionalDFactor()` that the subclass `LESCalculateF` overwrites its implementation. While the super class implementation returns zero, the subclass returns the computed ν_{smag} .

The `accumulateFValues()`, and `computeCandD()` were extended to receive the `additionalDFactor()` to add it to the kinematic viscosity before the calculation of the diffusion term. The `TurbulenceTrivialgridFluidSimulation` constructs the `TrivialgridEventHandle2FAdapter` with an object of the subclass `LESCalculateF` as an adaptee, and overwrite the implementation of the `calculateF()` of the superclass `TrivialgridFluidSimulation`, by calling the `iterate` method on the adapter that has `LESCalculateF` as adaptee.

Now the use of the `computeAdditionalDFactor()` of the super, or the sub class is decided during run time according to which of them was used as adaptee when constructing the `TrivialgridEventHandle2FAdapter` is according to which simulation is running. The `LESCalculateF` implements additional methods to these of the base class to enable the computation of the ν_{smag} (see section 2.3.2). These methods are `calculateGradU()` that calculates the $S_{i,j}$, `calculateNormSIJ()` that calculates the norm of $S_{i,j}$, `computeNuSGSSmagornisky()` that computes ν_{smag} for the Smagornisky model, and `computeNuLES()` that calls `computeNuSGSSmagornisky()` to enable further extensions

⁸The `iterate` method is responsible only about traversing the grid while the traversal is directed to a specific action according to the passed as a parameter.

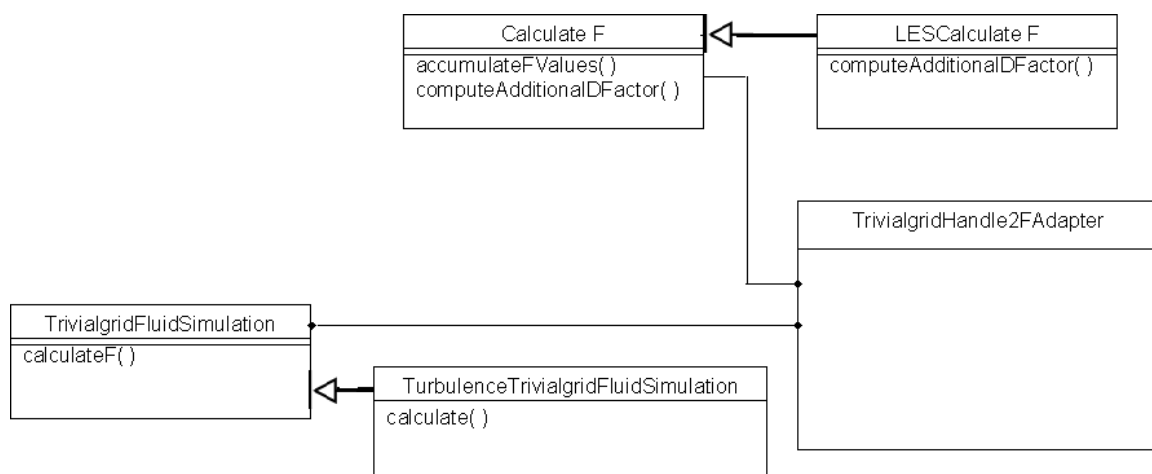


Figure 4.9: The extensions of the implementation to add the subgrid contribution to the diffusion operator computation.

of the project, when the call for a different model implementation might be needed. In addition to these main functional methods there are `computeVDD()` that will be discussed in the next section and some other methods that do minor tasks.

4.4.3 Van Driest damping

At the early stage of the numerical experiments, it was found that the turbulence viscosity near the walls was too high to instantiate turbulence. Then the so-called Van Driest damping had to be introduced within the calculations of ν_{smag} to damp the viscous effect near the wall. This damping is defined as,

$$VanDriestdampin\ factor = 1 - \exp\left(\frac{-y^+}{A^+}\right) \quad (4.10)$$

where A^+ is the Van driest damping constant and taken to be 25 as used by Kim and Moin [4].

The implementation of the Van Driest damping was done within the `LESCalculateF`,

and the usage of it is optional via a flag setting in the configuration file. The implementation is restricted to the channel flow, because y^+ is calculated directly from the y position. Thus for more complicated geometry where this is not valid, the method has to be modified.

Listing 4.1: Extracted and reduced source code for the averaging implementation on the regular grid.

```

void fluidturbulence::TrivialgridEventHandle2AverageTurbulentDataAdapter::
    handleElement(
        fluid::TrivialgridFluidVertexWithPersistentCellNumber* vertices,
        fluid::TrivialgridFluidCellWithPersistentCellNumber& cell,

        int vertexIndex[NUMBER_OF_VERTICES_PER_ELEMENT],
        const Vector& h,
        const Vector& position
    ) {
    //sum up
    _pMean(computeYIndex(position(1))) += (cell.getP() / _noCellsPerYLayer) /
                                           _currentTimeStepNumber;
    _cellsVisited ++;
}

void fluidturbulence::TrivialgridEventHandle2AverageTurbulentDataAdapter::
    touchVertexLastTime(
        fluid::TrivialgridFluidVertexWithPersistentCellNumber& fluidVertex,
        const Vector& position
    ){
    // sum up
    for (int d=0; d<DIMENSIONS; d++) {
        _uMean(computeYIndex(position(1)), d) += (fluidVertex.getU()(d) /
                                                  _noVerticesPerYLayer) / _currentTimeStepNumber;
    }
    _verticesVisited ++;
}

void fluidturbulence::TrivialgridEventHandle2AverageTurbulentDataAdapter::
    endIteration() {
    _currentTimeStepNumber ++;

    // prepare time averaging for next run
    _uMean *= (_currentTimeStepNumber-1.0)/_currentTimeStepNumber;
    _pMean *= (_currentTimeStepNumber-1.0)/_currentTimeStepNumber;
}

```

Listing 4.2: Extracted and reduced source code for calculating the second order statistics on the regular grid.

```

void fluidturbulence::TrivialgridEventHandle2AverageRmsTurbulentDataAdapter::
                                touchVertexLastTime(
fluid::TrivialgridFluidVertexWithPersistentCellNumber&  fluidVertex,
const Vector&                                           position
){
    const int yIndex = computeYIndex(position(1));

    // sum up
    for (int d=0; d<DIMENSIONS; d++) {
        _uRms(yIndex, d) += (_uMean(yIndex, d) - fluidVertex.getU()(d)) *
                            (_uMean(yIndex, d) - fluidVertex.getU()(d)) /
                            _noVerticesPerYLayer;
    }
    _reynoldsStress(yIndex,0) += ((_uMean(yIndex,0) - fluidVertex.getU()(0)) *
                                   (_uMean(yIndex,1) - fluidVertex.getU()(1))) /
                                   _noVerticesPerYLayer;

                                #ifdef Dim3
    _reynoldsStress(yIndex,1) += ((_uMean(yIndex,0) - fluidVertex.getU()(0)) *
                                   (_uMean(yIndex,2) - fluidVertex.getU()(2))) /
                                   _noVerticesPerYLayer;
    _reynoldsStress(yIndex,2) += ((_uMean(yIndex,1) - fluidVertex.getU()(1)) *
                                   (_uMean(yIndex,2) - fluidVertex.getU()(2))) /
                                   _noVerticesPerYLayer;
                                #endif _verticesVisited ++;
}

void fluidturbulence::TrivialgridEventHandle2AverageRmsTurbulentDataAdapter::
                                handleElement(
fluid::TrivialgridFluidVertexWithPersistentCellNumber*  vertices,
fluid::TrivialgridFluidCellWithPersistentCellNumber&    cell,

int                                                       vertexIndex[NUMBER_OF_VERTICES_PER_ELEMENT],
const Vector                                           h,
const Vector&                                           position
) {
    //sum up
    _pRms(computeYIndex(position(1))) +=
        ((_pMean(computeYIndex(position(1)))- cell.getP()) *
         (_pMean(computeYIndex(position(1)))- cell.getP()) /
         _noCellsPerYLayer ); /// _currentTimeStepNumber;

    _cellsVisited ++;
}

void fluidturbulence::TrivialgridEventHandle2AverageRmsTurbulentDataAdapter::
                                beginIteration() {
    _uRms = 0.0;
    _pRms = 0.0;
    _reynoldsStress = 0.0;
}

void fluidturbulence::TrivialgridEventHandle2AverageRmsTurbulentDataAdapter::
                                endIteration() {
    // prepare time averaging for next run
    _uRms = sqrt(_uRms);
    _pRms = sqrt(_pRms);
    _oldURms = (_oldURms *(_currentTimeStepNumber -1.0)/_currentTimeStepNumber) +
                (_uRms* (1.0 / _currentTimeStepNumber));
    _oldPRms = (_oldPRms *(_currentTimeStepNumber -1.0)/_currentTimeStepNumber) +
                (_pRms * (1.0/ _currentTimeStepNumber));
    _oldReynoldsStress = (_oldReynoldsStress *(_currentTimeStepNumber -1.0)/
                           _currentTimeStepNumber) +
                (_reynoldsStress * (1.0/ _currentTimeStepNumber));
}

```

Chapter 5

Numerical Experiments

5.1 The scenario

After the implementation was completely done the numerical simulations for the selected scenario of the channel flow became possible. The geometry and the main features of this scenario were described in 2.4.2. For running numerical experiments that are corresponding to reference data from Tokyo database [9], and Mansour, Moser and kim [8], the following parameters had to be specified for all experiments that were conducted within the configuration file¹.

I. Domain dimensions

The channel width δ was taken to be 1, and to assure periodicity in x and z direction. The length of the channel L_x was taken to be 6 as approximation of $3\pi\delta$, and L_z was taken to be 3 as approximation for $\pi\delta$.

II. Boundary conditions

The boundary conditions at the walls were set to no-slip, but at the streamwise and spanwise boundaries the periodic boundary conditions were imposed.

III. The friction Reynolds number

The experiments were designed such that Re_τ is equal to 395 identically to these

¹ An example of how the turbulence Peano configuration file looks like is given in appendix A

used by [9] and [8]. Nevertheless, much higher Reynolds were used to instantiate turbulence, that is because Re_τ of 395 is too low to instantiate turbulence.

IV. Fluid properties

Air with kinematic viscosity of ν of $1.76 \exp -5$, and ρ equal 1.

V. The pressure gradient

The scenarios were designed such that the flow is pressure driven. The pressure gradient was calculated such that the resulting Re_τ is 395. This was done as following,

given

$$\delta, \quad L_x, \quad L_y$$

$$\nu, \quad \rho$$

then

1. from (2.34),

$$u_t = \frac{Re_\tau \nu}{\delta}$$

2. from (2.31),

$$\tau_w = \rho u_\tau^2$$

3. substituting (2.29),

$$\Delta_P = -\frac{\tau_w}{\delta} L_x$$

The channel velocity field was initiated with a parabolic profile of bulk velocity 1.0, and a random noise 0.1 of $u_0(x)$ was added inside the domain, where $u_0(x)$ is the initial velocity field. The periodic boundaries are excluded from adding the noise to preserve the periodicity. The Van Driest damping was used mainly in all experiments since it was added to the code.

5.2 The experiments

The objective of the experiments done within this work was to instantiate turbulence (i.e. get turbulence starting data) and then run similar scenarios to these of the reference data to validate the implementation. In order to instantiate turbulence much higher Re_τ is used at the beginning, then when the turbulence develop the simulation should be restart with the real Re_τ of interest. Some simulations were conducted on the regular grid and many were conducted on the adaptive grid after it had been added to the code.

5.2.1 Experiments on the regular grid

After the implementation for the regular grid was done, some runs with a few time steps were done to estimate the execution time to develop turbulence at different resolutions. By experience the turbulence development needs to take as long as a fluid particle takes to cycle through the domain more than 10 times. A fluid particle cannot traverse more than one cell per time step according to the stability conditions. Thus the time needed to develop a turbulence is lower bounded by:

$$\text{Execution time} = 10 \times \text{No. of cells in x direction} \times \text{CPU time part time step} \quad (5.1)$$

Conducting simulations at higher resolution than that of $(\frac{1}{32})$ was found to be impractical in the sense of the execution time and storage, and at that of $(\frac{1}{32})$ was found to be impractical in the sense of the execution time. Thus the experiments that were conducted on the regular grid were only at mesh size of $(\frac{1}{16})$. Fig. 5.1 shows an image of the flow field at Re_τ of 3950 after 3000 time steps, the flow develop no turbulence because the resolution was not sufficient to resolve the viscous layer near the wall. As a rule of thumb for wall flows, the boundary layer has to be resolved with a at least few mesh points, that is equivalent to y^+ around 1.3 or smaller. This means a resolution of approximately 1250 in y direction, running for such a resolution in serial on the Peano's regular isn't yet affordable, and it actually makes no sense as this resolution is not really needed elsewhere away from the wall.

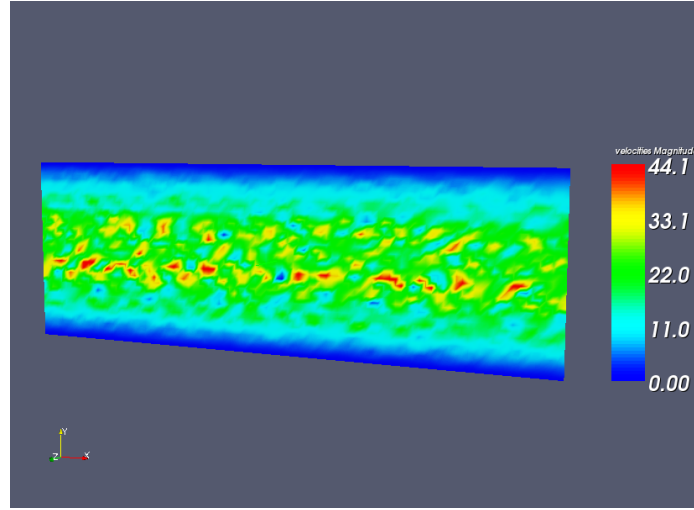


Figure 5.1: The flow velocity field at Re_τ of 3950 and mesh size of $(\frac{1}{16})$ on the regular grid after 3000 time steps

5.2.2 Experiments on the adaptive grid

After the trials to get turbulence starting data on the regular grid didn't succeed even with the Van Driest damping, the proposed idea was to extend the implementation to the Peano's adaptive grid to better resolve near the wall. Once the implementation was done the estimation of the execution time to develop turbulence at different resolutions was performed. The tests were done for the cases of adaptivity in y direction with maximum mesh size of $(\frac{2}{27})$ and different minimum mesh sizes. The affordable and reasonable resolution was thought to be of $(\frac{2}{27})$ as maximum mesh size, and $(\frac{2}{81})$ as minimum mesh size. Many simulations at this resolution were performed for different Re_τ and C_{smag} (see table 5.1 that shows the different combinations). Each run was conducted for 3000 time steps, that was estimated to be sufficient for a fluid particle to cycle through the domain more than 10 times.

The simulations were mainly monitored and steered based on the visualized flow field. Although at the adaptive grid simulations a higher resolution were afforded, non of these simulation runs developed any turbulence. The most promising run Sim3950_100_3000 (see table 5.1) was picked and let run for more 9000 time steps, then restarted with the scenario's real Reynolds number, and fresh statistics for 3000

time steps. The visualized flow field (see Fig. 5.2 , 5.3 , and 5.4) showed the same behavior of no turbulence development.

C_{smag}	$Re_\tau = 3950$	$Re_\tau = 39500$
0.1	Sim3950_100_3000	Sim39500_100_00
0.125	Sim3950_125_3000	Sim39500_125_3000
0.15	Sim3950_150_3000	Sim39500_125_3000

Table 5.1: The different stimulations that were conducted at the adaptive grid for a maximum mesh size of 2/27 and minimum mesh size of 2/81 in y direction, and the way that they were coded.

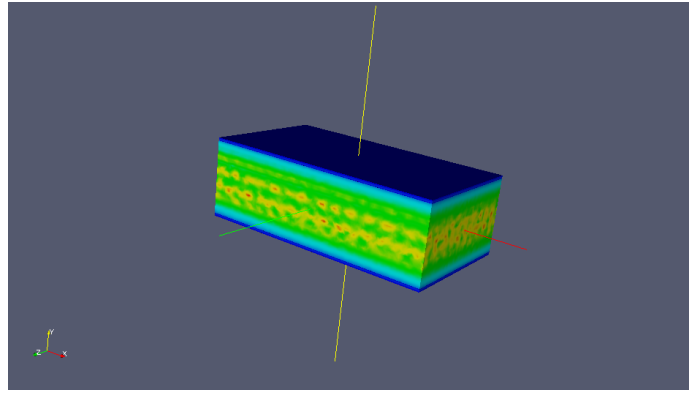


Figure 5.2: The three dimensional velocity field in a surface representation of the run at Re_τ of 395 , and C_{smag} of 0.1 after 3000 time steps

A deeper analysis were performed for this case with the matlab comparison tool implemented within the activities of this thesis. Fig.(5.10, 5.11, 5.12, 5.13, and 5.14) shows the plots of the simulation statistics corresponding to Tokyo available statistics [9] normalized by u_τ versus the references data. Fig??. Fig.(5.5, 5.6, 5.7, 5.8, and 5.9) shows the plots of the simulation statistics corresponding to Mansour, Moser, and Kim available statistics [8] normalized by u_τ versus the references data. These plots show large difference between the fully developed turbulence DNS data and these of the experiment. This difference is due to the fact that the simulation run developed no turbulence data. This conclusion is supported by the plots in Fig.(5.15, 5.16, 5.17, 5.18, and 5.19) where normalization was undone. The plots shows that

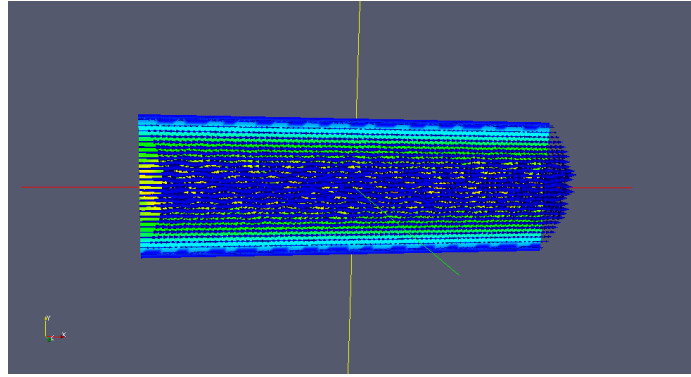


Figure 5.3: An arrow representation of the velocity field at a slice taken at the center of the channel, and normal to the spanwise direction. The run was at Re_τ of 395 , and C_{smag} of 0.1 after 3000 time steps

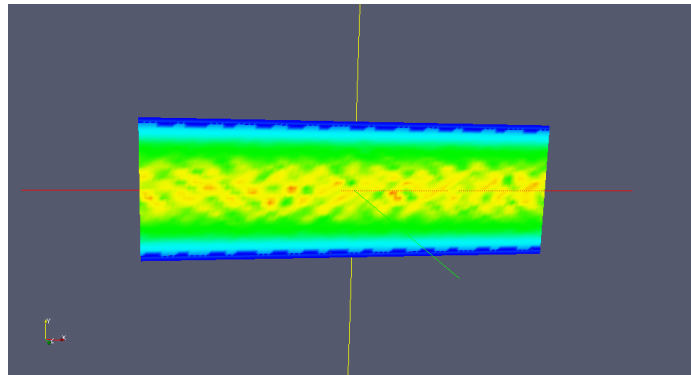


Figure 5.4: A slice taken at the center of the channel, and normal to the spanwise direction. The run was at Re_τ of 395 , and C_{smag} of 0.1 after 3000 time steps

the mean velocity simply has its initial parabolic profile of a bulk velocity near to 1, and the fluctuations are almost equal zero except close to the channel center due the the random noise effect.

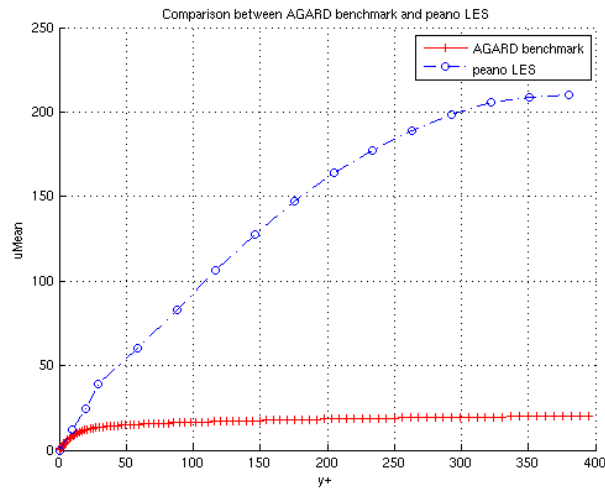


Figure 5.5: The Peano's LES mean velocity normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]

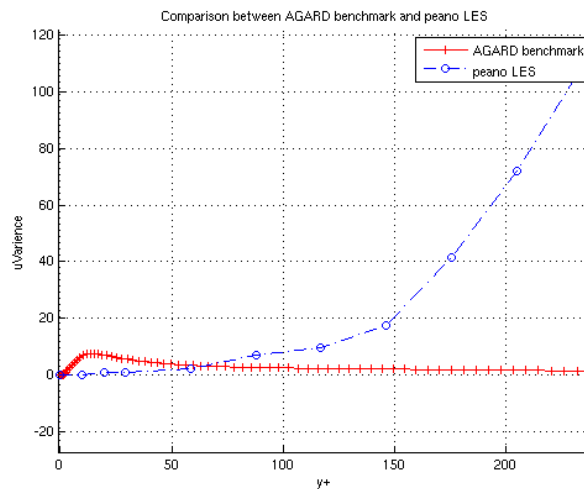


Figure 5.6: The Peano's LES streamwise velocity variance normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]

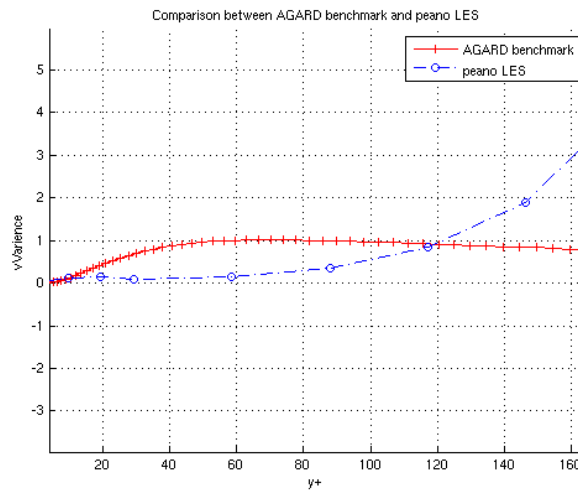


Figure 5.7: The Peano’s LES wall-normal velocity variance normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]

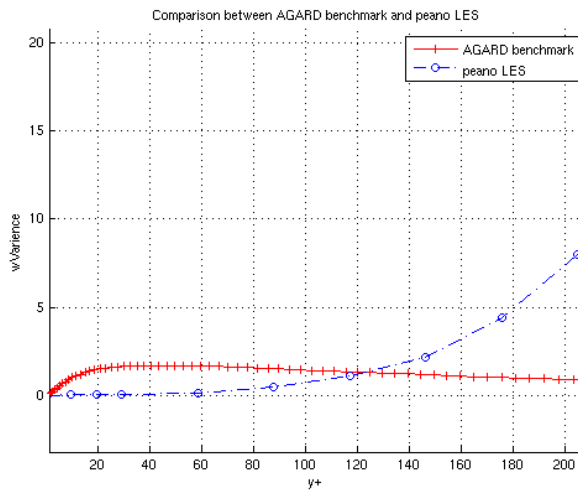


Figure 5.8: The Peano’s LES spanwise velocity variance normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]

5.2.3 Conclusions

In all the simulations conducted the resolution near the wall was not sufficient to resolve the boundary layer and allow the turbulence to develop. Therefore, in order to get the turbulence starting data or to judge the implementation of the model, a

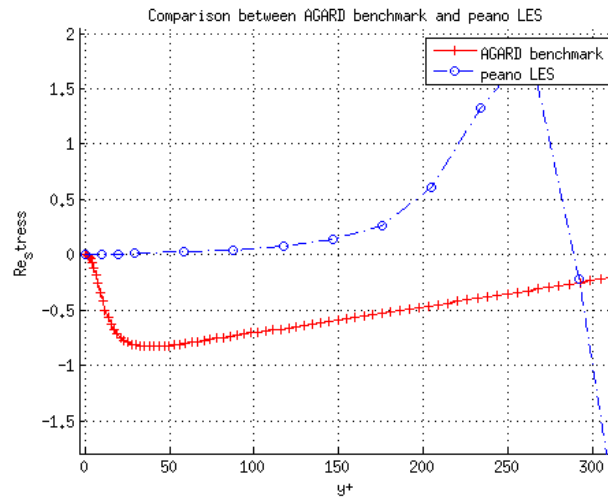


Figure 5.9: The Peano’s LES uv normalized by u_τ vs. the reference DNS data of Mansour, Moser and Kim[8]

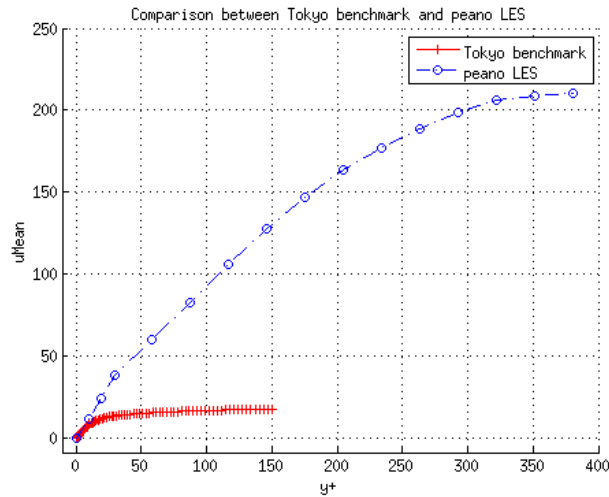


Figure 5.10: The Peano’s LES mean velocity normalized by u_τ vs the reference data of Tokyo[9]

finer resolution near the wall have to be afforded.

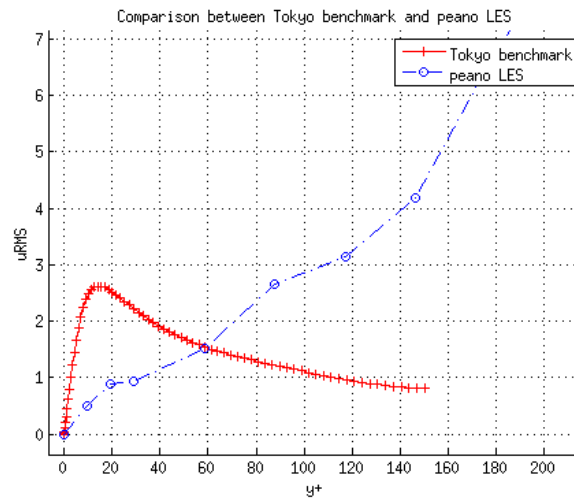


Figure 5.11: The Peano’s LES streamwise velocity RMS normalized by u_τ vs the reference data of Tokyo[9]

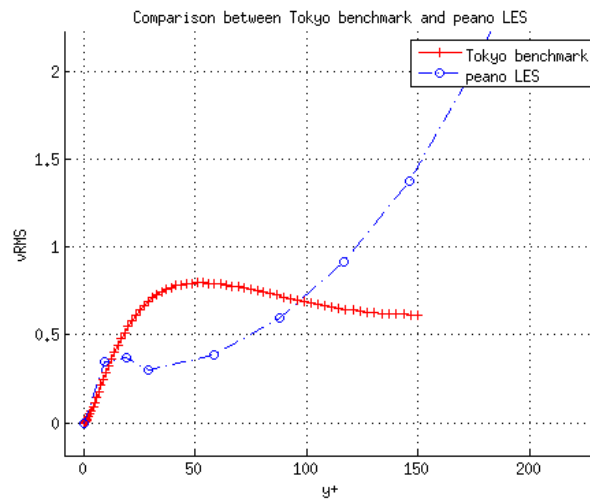


Figure 5.12: The Peano’s LES wall-normal velocity RMS normalized by u_τ vs the reference data of Tokyo[9]

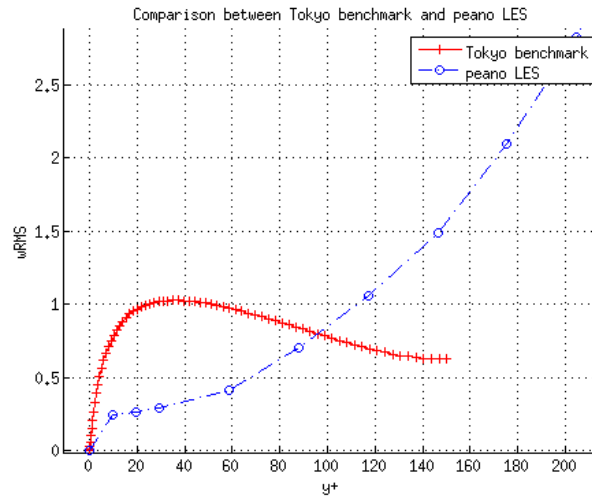


Figure 5.13: The Peano’s LES spanwise velocity RMS normalized by u_τ vs the reference data of Tokyo[9]

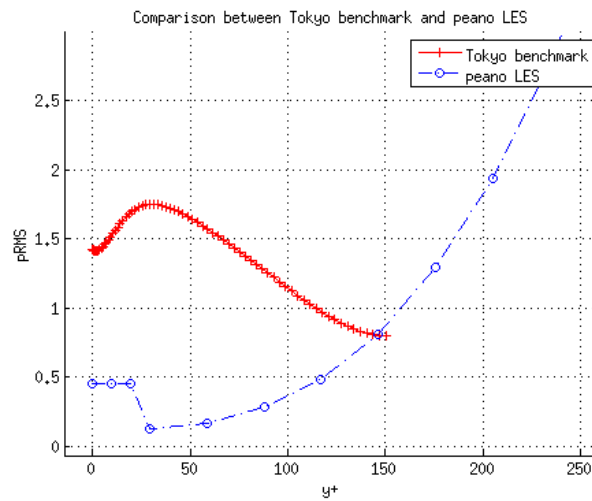


Figure 5.14: The Peano’s LES pressure RMS normalized by u_τ vs the reference of Tokyo[9]

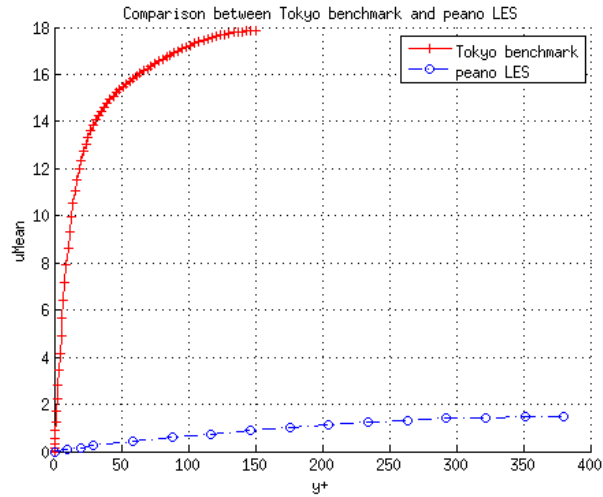


Figure 5.15: The mean velocity vs. the reference data of Mansour, Moser and Kim[8]

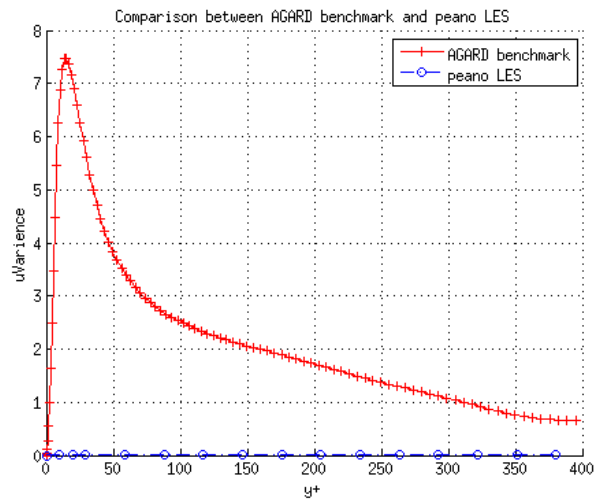


Figure 5.16: The streamwise velocity variance vs. the reference data of Mansour, Moser and Kim[8]

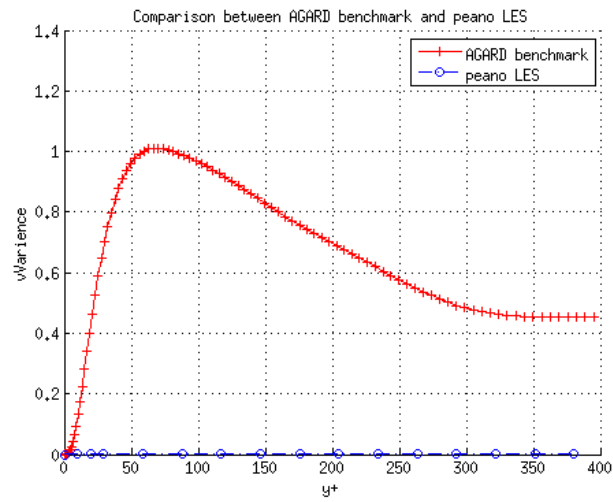


Figure 5.17: The wall-normal velocity variance vs. the reference data of Mansour, Moser and Kim[8]

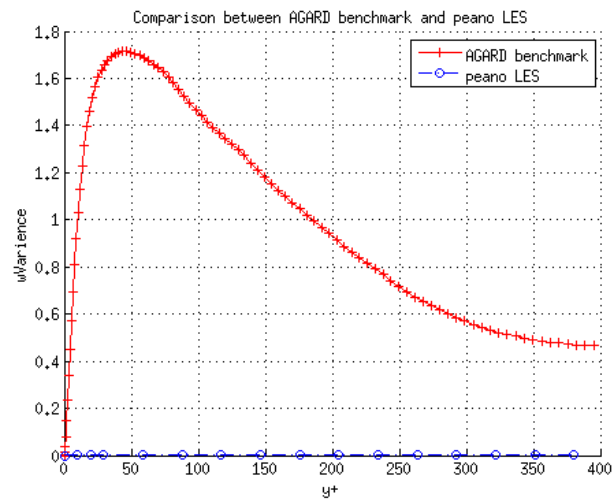


Figure 5.18: The spanwise velocity variance vs. the reference data of Mansour, Moser and Kim[8]

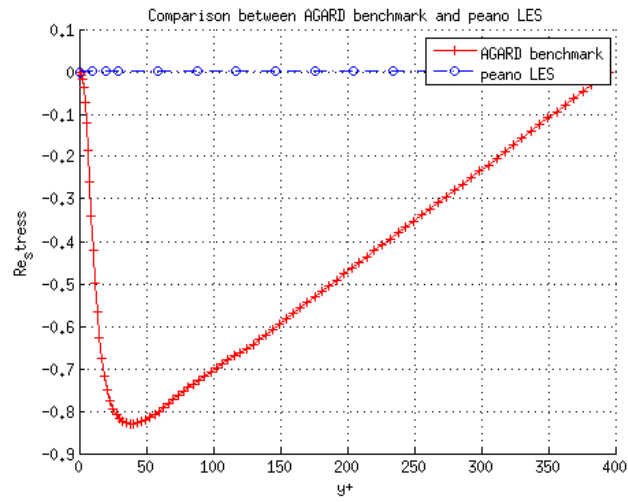


Figure 5.19: The uv vs the reference data of Mansour, Moser and Kim[8]

Chapter 6

Conclusion and Outlook

The objective of the thesis was to integrate an LES based turbulence solver into the Peano framework, then to test and validate the implementation. The primary objective was met and the implementation was done on the Peano's regular grid and has been extended to the adaptive grid afterward. The implementation was tested on both integration and unit levels. Many side tasks were performed to allow the steering of the simulations and analyzing the results, the most important are the restart mechanism including the checkpointing, and the averaging including a matlab code to perform the comparison with the reference data. Many numerical simulations were performed on both the regular and the adaptive grids, but no valid turbulence data was produced. The implementation could not be validated, as the sufficient resolution near the wall could not be afforded in the serial mode neither for the regular nor for the adaptive grids. The simulations that were performed on the adaptive grid seemed to be more promising when visualized, because a finer resolution were afforded near the wall. Nevertheless, they didn't develop valid turbulence data. A conviction has been reached that this is due to the insufficient resolution near the wall.

Two suggested solutions are proposed. The first, is to run simulations on the adaptive grid with higher resolution when the parallel version is available. Nevertheless, that would still be costly. The latter, is to implement a stretched mesh that enables refitment in y direction without the need to respect that on the other two coordinate

directions. A possible extension on the model level is to implement the dynamic Smagorisky model, such an extension is possible and easy due to the structure of the current implementation described earlier.

Bibliography

- [1] Pierre Sagaut. (1989). *Large Eddy Simulation for Incompressible Flows, An Introduction, 3rd edition*. New York, US: Springer Berlin Heidelberg.
- [2] Roger Peyret. (ed). (2000). *Handbook of Computational Fluid Mechanics*. London: Academic Press.
- [3] Stephen B. Pope. (2000). *Turbulent Flows*. United States: Cambridge University Press.
- [4] Parviz Moin and John Kim. (1982). Numerical Investigation of Turbulent Channel Flow. *J. Fluid Mech.* (Vol. 118), No. 12, Pages 314–337.
- [5] Bernd Bruegge and Allen H. Dutiot. (1993). *Object-Oriented Software Engineering, Using UML, Patterns, and JavaTM, 2nd edition*. Pearson Prentice Hall: Alan R. Apt.
- [6] Victor L. Streeter and E. Benjamin Wylie. (1983). *Fluid Mechanics, First SI Metric Edition, International Student Edition*. Singapore: McGraw-Hill Book Co.
- [7] Robert L. Daugherty, Joseph B. Franzini and E. John Finnemore. (1985). *Fluid Mechanics, 8th edition, International Student Edition*. Singapore: McGraw-Hill Book Co..
- [8] Robert D. Moser, John Kim and Nagi N. Mansour. (1999). *DNS data for Turbulent Channel Flow*[Internet] Available from: <http://turbulence.ices.utexas.edu/data/MKM/>

- [9] A. Kuroda and N. Kasagi. (1990). *Fully Developed 2-D Channel Flow*[Internet]
Available from: http://www.thtlab.t.u-tokyo.ac.jp/DNS/dns_database.html
- [10] *PETSc Documentation*[Internet] Available from: <http://www-unix.mcs.anl.gov/petsc/petsc-2/documentation/index.html>
- [11] *The Peano's Documentation*[Internet] Available from:
<http://www5.in.tum.de/forschung/CFD/doxys/peano/index.html>
- [12] *Doxys, a C++ documentation tool*[Internet] Available from:
http://www.doxys.dk/doxys_homepage/index.html
- [13] F. Günther and M. Mehl and M. Pögl and C. Zenger. (2006). A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves *SIAM J. Sci. Comput.*. (Vol. 28), No. 5, Pages 1634–1650.
- [14] M. Brenk and H.-J. Bungartz and M. Mehl and I.L. Muntean and T. Neckel and T. Weinzierl. (2008). Numerical Simulation of Particle Transport in a Drift Ratchet *SIAM Journal of Scientific Computing*. (Vol. 28), No. 5, Pages 1634–1650.
- [15] A. J. Chorin. (1968). Numerical Solution of the Navier-Stokes Equations *Math. Comp.*. (Vol. 22), Pages 745–762.