ขั้นตอนวิธีเชิงขนานสำหรับการจำลองเชิงตัวเลขของพลศาสตร์ของร่องรอยการไหลแบบปั่นป่วนในของไหลที่เป็นชั้นๆ

นายสอาด ม่วงจันทร์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรดุษฎีบัณฑิต
สาขาวิชาคณิตศาสตร์ประยุกต์
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2550

PARALLEL ALGORITHMS FOR NUMERICAL SIMULATION OF TURBULENT WAKE DYNAMICS IN A STRATIFIED FLUID

Sa-at Muangchan

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Applied Mathematics

Suranaree University of Technology

Academic Year 2007

PARALLEL ALGORITHMS FOR NUMERICAL SIMULATION OF TURBULENT WAKE DYNAMICS IN A STRATIFIED FLUID

Suranaree University of Technology has approved this thesis submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy.

Thesis Examining Committee

(Asst. Prof. Dr. Eckart Schulz)
Chairperson

(Assoc. Prof. Dr. Nikolay Moshkin)
Member (Thesis Advisor)

(Assoc. Prof. Dr. Prapasri Asawakun)
Member

(Assoc. Prof. Dr. Ekachai Juntasaro)
Member

(Assoc. Prof. Dr. Varangrat Juntasaro)
Member

(Prof. Dr. Pairote Sattayatham)
Vice Rector for Academic Affairs

(Assoc. Prof. Dr. Prapan Manyum)
Dean of Institute of Science

สอาด ม่วงจันทร์ : ขั้นตอนวิธีเชิงขนานสำหรับการจำลองเชิงตัวเลขของพลศาสตร์ของร่องรอยการไหลแบบปั่นป่วนในของไหลที่เป็นชั้นๆ (PARALLEL ALGORITHMS FOR NUMERICAL SIMULATION OF TURBULENT WAKE DYNAMICS IN A STRATIFIED FLUID) อาจารย์ที่ปรึกษา : รองศาสตราจารย์ ดร.นิโคไลน์ มอสกิน, 102 หน้า.

งานวิจัยนี้ได้พัฒนาขั้นตอนวิธีเชิงขนานสำหรับการจำลองเชิงตัวเลขของพลศาสตร์ของร่องรอยการไหลแบบปั่นป่วนในของไหลที่เป็นชั้นๆ เพื่ออธิบายพลศาสตร์ของร่องรอยการไหลแบบปั่นป่วนที่ห่างจากด้านหลังของวัตถุที่เคลื่อนที่ด้วยแรงภายนอก และวัตถุที่เคลื่อนที่ได้ด้วยตัวเองในของไหลที่เป็นชั้นๆ โดยใช้แบบจำลองการไหลแบบปั่นป่วน แบบจำลองที่ซับซ้อนส่วนมากประกอบด้วยสมการเชิงอนุพันธุ์สำหรับการถ่ายเทของความเค้นเรย์โนลด์ งานวิจัยนี้ได้สร้างขั้นตอนวิธีเชิงขนานสองแบบ แบบที่หนึ่งเน้นวิธีการแบ่งแยกแบบฟังก์ชัน และแบบที่สองเป็นวิธีการแบ่งแยกแบบโดเมน ผลเฉลยเชิงตัวเลขของสมการเชิงอนุพันธ์ของการถ่ายเทหาด้วยวิธีขั้นเศษส่วน โดยเปรียบเทียบผลการคำนวณเชิงตัวเลขที่ได้กับข้อมูลที่ได้จากการทดลอง และเปรียบเทียบความเร็วของการคำนวณเชิงขนานกับผลการวิเคราะห์เชิงทฤษฎี โดยได้ทำการวิเคราะห์ความเร็วที่ขึ้นกับเวลาลาเทนซีและแบนด์วิดธ์ ขั้นตอนวิธีแบ่งแยกแบบโดเมนให้ค่าความเร็วที่ดีกว่าขั้นตอนวิธีแบ่งแยกแบบฟังก์ชัน กระบวนการทั้งหมดประมวลผลด้วยสองระบบกลุ่มคอมพิวเตอร์สมรรถนะสูง

งานวิจัยนี้ ได้พบว่าขั้นตอนวิธีเชิงขนานทั้งสองแบบเป็นเครื่องมือที่ใช้ได้กับการจำลองเชิงตัวเลขของพลศาสตร์ของร่องรอยการไหลแบบปั่นป่วนในของไหลที่เป็นชั้นๆ และยังสามารถนำไปประยุกต์ใช้ได้กับการทดลองเชิงตัวเลขของพลศาสตร์การไหลแบบปั่นป่วนในของไหลที่เป็นชั้นๆที่มีความซับซ้อนกว่าเช่น การจำลองเชิงตัวเลขของพลศาสตร์ของร่องรอยการไหลแบบไหลวนปั่นป่วนในของไหลที่เป็นชั้นๆ

สาขาวิชาคณิตศาสตร์ ลายมือชื่อนักศึกษา

ปีการศึกษา 2550 ลายมือชื่ออาจารย์ที่ปรึกษา

ลายมือชื่ออาจารย์ที่ปรึกษาร่วม

SA-AT MUANGCHAN : PARALLEL ALGORITHMS FOR NUMERICAL SIMULATION OF TURBULENT WAKE DYNAMICS IN A STRATIFIED FLUID. THESIS ADVISOR : ASSOC. PROF. NIKOLAY MOSHKIN, Ph.D. 102 PP.

PARALLEL ALGORITHM/ MPI/ FUNCTIONAL DECOMPOSITION/ DOMAIN DECOMPOSITION/ TURBULENT WAKE/ STRATIFIED FLUID

The research aims at developing parallel algorithms for numerical simulations of turbulent wake dynamics in a linearly stratified fluid. In order to describe the far turbulent wake flow behind a towed and self-propelled axisymmetric bodies in a linearly stratified medium, a hierarchy of semi-empirical turbulence models has been used. Most complex turbulence models are composed of differential equations for transport of normal Reynolds stresses. Two parallel algorithms for numerical models of turbulent wake dynamics in a stratified fluid have been developed. The first one is based on the functional decomposition approach. The second one is based on the domain decomposition technique. The numerical solutions of the transport differential equations are obtained by the fractional step method. The validation of parallel algorithms are done by comparing the numerical results with available experimental results. The speedups of both parallel algorithms are compared with theoretical estimates. The speedup depending on the latency time and bandwidth are analyzed. The technique of domain decomposition demonstrated better speedup than the functional decomposition approach. The computation in this research was conducted on the two cluster systems.

The present research shows that both developed parallel algorithms are viable tools to numerical simulations of turbulent wake dynamics in a stratified fluid and can serve as a basis for numerical experiments with more complicated

models of turbulence, for example, the numerical simulation of swirling turbulent wake dynamics in a stratified fluid.

School of Mathematics

Academic Year 2007

Student's Signature_____

Advisor's Signature_____

Co-advisor's Signature_____

ACKNOWLEDGEMENTS

I am profoundly grateful to my thesis advisor Assoc. Prof. Dr. Nikolay Moshkin and my thesis co-advisor Assoc. Prof. Dr. Prapasri Asawakun for their support, patient help and offering many useful suggestions.

I would like to acknowledge all the lecturers who taught and helped me during the course of studies at Suranaree University of Technology. They are Prof. Dr. Sergey Meleshko, Asst. Prof. Dr. Eckart Schulz, Asst. Prof. Dr. Arjuna Peter Chaiyasena and Dr. Ole Nielsen.

I also would like to express my appreciation to Assoc. Prof. Dr. Ekachai Juntasaro and Assoc. Prof. Dr. Varangrat Juntasaro for their valuable discussions and comments.

I would like to express my appreciation to the following for their sincere help. These include Asst. Prof. Dr. Jessada Tanthanuch and all of my friends at Suranaree University of Technology.

I acknowledge the financial support of Sakon Nakhon Rajabhat University. I am indebted to Department of Mathematics and Statistics, Sakon Nakhon Rajabhat University for grants to support my studies throughout.

Finally, I am deeply grateful to my parents, my brother, my sister and my love, Wilaiporn Peangnoo for support, understanding encouragement, and love.

Sa-at Muangchan

CONTENTS

Page

ABSTRACT IN THAI I

ABSTRACT IN ENGLISH II

ACKNOWLEDGEMENTS IV

CONTENTS V

LIST OF TABLES VI

LIST OF FIGURES VII

CHAPTER

I INTRODUCTION 1

1.1 Different Approaches to the Study of Turbulent Flow 1

1.2 General Context of Parallel Computer 3

1.3 Objectives and Overviews of the Thesis 7

II MATHEMATICAL MODELS OF TURBULENT WAKE DYNAMICS IN A STRATIFIED FLUID 9

2.1 Introduction 9

2.2 Governing Equations 10

2.3 Initial and Boundary Conditions 15

2.4 Mathematical Model of Dynamics of Passive Scalar in Turbulent Wakes in a Stratified Fluids 16

III SEQUENTIAL AND PARALLEL ALGORITHMS 18

3.1 Sequential Algorithm 18

3.2 Parallel Algorithms 26

CONTENTS (Continued)

Page

3.2.1 Parallel Functional Decomposition 27

3.2.2 Parallel Domain Decomposition 35

IV VALIDATION AND NUMERICAL RESULTS 45

4.1 Comparison of Parallel and Sequential Algorithms 45

4.2 Results of Experimental Performance of Parallel Algorithms 52

V CONCLUSIONS 68

5.1 Contributions 68

5.2 Conclusions 69

5.3 Recommendations for Future Research 70

REFERENCES 72

APPENDICES

APPENDIX A THE COMPUTATION OF THE THEORETICAL SPEEDUP 77

A.1 The Theoretical Speedup of the Domain Decomposition Algorithms 77

A.2 The Theoretical Speedup of the Functional Decomposition Algorithms 78

APPENDIX B TERMINOLOGY 81

B.1 Finite Difference Method 81

B.2 The Method of Stabilizing Corrections 81

B.3 The Elimination Method for Three-Point Equations (Samarskij (1989)) 83

B.4 Run Time and Speedup 86

CONTENTS (Continued)

Page

APPENDIX C FREQUENTLY USED MPI SUBROUTINES 88

C.1 Environmental Subroutines 88

C.2 Collective Communication Subroutines 90

C.3 Point-to-Point Communication Subroutines . . . 93

C.4 Derived Data Types 97

CURRICULUM VITAE 102

LIST OF TABLES

Table		Page
4.1	Axial values of the dimensionless turbulent energy, e, depending on the distance in the momentumless wake, $F_d = 280$	49
4.2	Axial values of the dimensionless turbulent energy, e, depending on the distance in the drag wake, $F_d = 280$	49
4.3	Axial values of the dimensionless axial velocity defect, U_d, depending on the distance in the momentumless wake, $F_d = 280$	50
4.4	Axial values of the dimensionless axial velocity defect, U_d, depending on the distance in the drag wake, $F_d = 280$	50
4.5	Axial values of the dimensionless turbulent dissipation, ε, depending on the distance in the momentumless wake, $F_d = 280$	51
4.6	Axial values of the dimensionless turbulent dissipation, ε, depending on the distance in the drag wake, $F_d = 280$	51
4.7	Speedup results of the functional decomposition algorithm on Cluster-I	56
4.8	Speedup results of the domain decomposition algorithm on Cluster-I	56
4.9	Speedup results of the functional decomposition algorithm on SUT-HPCC	57
4.10	Speedup results of the domain decomposition algorithm on SUT-HPCC	57
4.11	Efficiency results of the functional decomposition algorithm on Cluster-I	58

LIST OF TABLES (Continued)

Table		Page
4.12	Efficiency results of the domain decomposition algorithm on Cluster-I	58
4.13	Efficiency results of the functional decomposition algorithm on SUT-HPCC	59
4.14	Efficiency results of the domain decomposition algorithm on SUT-HPCC	59
A.1	The theoretical speedup of the domain decomposition algorithm for grid size=800×1550 on SUT-HPCC	79
A.2	The theoretical speedup of the functional decomposition algorithm for grid size=800×1550 on SUT-HPCC	80

LIST OF FIGURES

Figure		Page
1.1	Classification by Flynn (1966)	4
2.1	Turbulent wake in a stratified fluid	11
3.1	Staggered grid: filled circles denote $f = \langle p_1 \rangle$, $\langle \rho_1 \rangle$, U_d, e, ε or $\langle u_l' \, u_k' \rangle$, filled diamonds denote vertical W z-velocity, filled squares denote V y-velocity component	19
3.2	Flowchart of sequential algorithm for Model 4 with equation for passive scalar	24
3.3	Sketch of axial velocity profile at $x = x_0$ in case of momentumless and drag wakes	26
3.4	A sequential process	27
3.5	A parallel functional decomposition process	28
3.6	Sketch of parallel functional decomposition at the l^{th} stage	32
3.7	Data updating in parallel functional decomposition algorithm . . .	33
3.8	Flowchart of parallel functional decomposition algorithm	34
3.9	Flowchart of functional decomposition with $P = 2$	35
3.10	Flowchart of the functional decomposition with $P = 4$	36
3.11	The range of strip on $CPU(K-1)$	37
3.12	The sketch of domain decomposition process in direction of $Y-$axis	40
3.13	The sketch of domain decomposition process in direction of $X-$axis	42
3.14	Updating data blocks on each processor	43

LIST OF FIGURES (Continued)

Figure		Page
4.1	Comparison of the axial values of the longitudinal velocity component defect, $U_d(x)$ in the wake behind a towed body calculated by the sequential algorithm and parallel algorithms with Lin and Pao's experimental data and Hassid's computational results	47
4.2	Comparison of the axial values of the longitudinal velocity component defect, $U_d(x)$ in the wake behind a self-propelled body calculated by the sequential algorithm and parallel algorithms with Lin and Pao's experimental data and Hassid's computational results . .	47
4.3	Comparison of the axial values of the turbulent energy, $e_0(x)$ in the wake behind a towed body calculated by the sequential algorithm and parallel algorithms with Lin and Pao's experimental data and Hassid's computational results	48
4.4	Comparison of the axial values of the turbulent energy, $e_0(x)$ in the wake behind a self-propelled body calculated by the sequential algorithm and parallel algorithms with Lin and Pao's experimental data and Hassid's computational results	48
4.5	Isolines $\langle\rho_1\rangle/(aD\rho_0 Fr_D^{1/4})$ =const. Momentumless wake, $F_d = 280$. Functional and domain decomposition, (a) $t/T = 1$, (b) $t/T = 2$ and (c) $t/T = 3$	53
4.6	Isolines $\langle\rho_1\rangle/(aD\rho_0 Fr_D^{1/4})$ =const. Drag wake, $F_d = 280$. Functional and domain decomposition, (a) $t/T = 1$, (b) $t/T = 2$ and (c) $t/T = 3$	54

LIST OF FIGURES (Continued)

Figure		Page
4.7	Speedup results of grid size 400 × 750, 600 × 1150 and 800 × 1550 on Cluster-I	60
4.8	Speedup results of grid size 400 × 750, 600 × 1150 and 800 × 1550 on SUT-HPCC	61
4.9	Comparison between theoretical speedup and numerical speedups of parallel functional decomposition on Cluster-I and SUT-HPCC, grid size=400 × 750	62
4.10	Comparison between theoretical speedup and numerical speedups of parallel functional decomposition on Cluster-I and SUT-HPCC, grid size=600 × 1150	63
4.11	Comparison between theoretical speedup and numerical speedups of parallel functional decomposition on Cluster-I and SUT-HPCC, grid size=800 × 1550	64
4.12	Comparison between theoretical speedup and numerical speedups of parallel domain decomposition on Cluste-I and SUT-HPCC, grid size=400 × 750	65
4.13	Comparison between theoretical speedup and numerical speedups of parallel domain decomposition on Cluster-I and SUT-HPCC, grid size=600 × 1150	66
4.14	Comparison between theoretical speedup and numerical speedups of parallel domain decomposition on Cluster-I and SUT-HPCC, grid size=800 × 1550	67

CHAPTER I

INTRODUCTION

1.1 Different Approaches to the Study of Turbulent Flow

In everyday life, there are many opportunities to observe turbulent flow such as waterfalls, the buffeting of a strong wind, atmospheric flow, prediction in weather forecast, the flow around an aircraft, the wake behind bodies. Turbulent flow is always three-dimensional, unsteady, rotational, and irregular. The irregularity of turbulent motion is due to the inherent nonlinearity of the Navier-Stokes equations when the Reynolds number is beyond the critical value and turbulent flow is also stochastic and chaotic. To predict the gross or average behavior of turbulent flow, a *simulation approach* and a *mathematical model approach* must be used (Batchelor (2000)).

There are two simulation approaches: direct numerical simulation (DNS) and large-eddy simulation (LES). In DNS, the Navier-Stokes equations are solved to determine the flow characteristics. Turbulent flow is characterized by a wide range of length scales and time scales. The idea of the energy cascade is the kinetic energy entering the turbulence (through the production mechanism) at the largest scales of the motion. This energy is then transferred (by inviscid processes) to smaller and smaller scales until, at the smallest scales, the energy is dissipated by viscous action. Since all length scales and time scales have to be resolved during the simulation, so DNS and LES are therefore computationally expensive. In LES, the equations are solved for a 'filtered' velocity field, which represents the large-scale turbulent motions. The equations solved include a model for the influence

of the smaller-scale motions which are not directly represented. In particular, for DNS, the computational requirements rise so steeply with the Reynolds number that the approach is applicable only for the flows of low or moderate Reynolds numbers.

In the turbulence model approach, the equations are solved for mean quantities. It has been more than a century since Reynolds introduced the averaged Navier-Stokes equations. The averaging process produces a set of turbulent stresses, known as Reynolds stresses, which are additional unknown variables. The simplest Reynolds stress models is obtained from a turbulent viscosity hypothesis. The turbulent viscosity can be calculated from an algebraic relation or it can be obtained from turbulent quantities such as the turbulent energy e and the dissipation ε for which the transport model equations are solved. In Reynolds stress models, the transport model equations are solved for the Reynolds stresses. Not all models are applicable to all flows. In application to a particular flow, the accuracy of the model can be determined by comparing model calculations with experimental measurements. The discrepancy between measured and calculated flow properties arises from the inaccuracy of the model, numerical error, measurement error and discrepancies in the boundary conditions. The important conclusion is that a comparison between measured and calculated flow properties determines the accuracy of the model.

The most commonly studied turbulent free shear flows are jets, wakes and mixing layers. The term 'free' implies that these flows are remote from walls and turbulent flow arises because of mean-velocity differences. Turbulent wake behinds an axisymmetric body in homogeneous and stratified fluid has been considered in a number of experiments in (Spedding (2001), Gourlay *et al.* (2001), Dommermuth *et al.* (2002)). The flow that arises in a turbulent wake behind a body that

moves in a stratified fluid is rather peculiar. With a relatively weak stratification a turbulent wake first develops essentially in the same way as in a homogeneous fluid and extends symmetrically. However, buoyancy forces oppose the vertical turbulent diffusion. Therefore the wake has a flattened form at large distances from the body and, finally, ceases to extend in the vertical direction. Because of the turbulent mixing, the fluid density within the wake is distributed more uniformly than outside it. Buoyancy forces tend to restore the former unperturbed state of a stable stratification. As a result, convective flows, which give rise to internal waves in an ambient fluid, arise in the plane perpendicular to the wake axis. Shear flow is used to validate model of turbulence, because there is no influence of the boundary and they are sensitive to modifications of the Reynolds stresses. Turbulent wake is an example of free shear flow which is often selected to verify the predictability of a turbulent model. Axis pressure gradient and viscous diffusion are small and a major role in determine flow field more sensitive to modelling of the Reynolds stress, kinetic energy and dissipation.

As a rule mathematical modelling requires the solution of a large system of nonlinear differential equation in a complex domain. Finite differential schemes are widely applied to approximate solutions of boundary value problems. The finite different method requires the introduction of grid in physical domain of interest. Even for a small number of grid points in the three-dimensional case, this modelling process can involve trillions of operations. Thus, these problems take an unacceptably long time to solve on supercomputer.

1.2 General Context of Parallel Computer

In 1966, Flynn (1966) categorized parallel computer architectures according to how the data stream and instruction stream are organized. His idea is depicted

in Figure 1.1. The multiple instruction, single data (MISD) class describes an

		Instruction stream	
		Single	Multiple
Data stream	Single	SISD	MISD
	Multiple	SIMD	MIMD

Figure 1.1 Classification by Flynn (1966)

empty set. The single instruction, single data (SISD) class contains the normal single processor computer. The single instruction, multiple data (SIMD) class has parallelism at the instruction level while the multiple instruction, multiple data (MIMD) class has parallelism at the level of program execution (each processor runs its own code).

Recently, many parallel computer systems have appeared, such as IBM, SUN, SGI, and etc. There are classified by memory access (Schönauer (2000), Kumar *et al.* (1994), Hwang (1993), Damaj (2006)), there are two memory classification as follows:

- Shared memory: Shared memory is memory that is accessed by several competitive processes at the same time. Multiple memory requests are handled by hardware or software protocols. Each processor has access to all of the data. Below is a brief description of three different models to construct shared memory systems (Hwang (1993)).
  - The uniform memory access (UMA) model: All processors have equal access time to the whole memory which is uniformly shared by all pro-

cessors.

  - The nonuniform memory access (NUMA) model: the access time to the shared memory varies with the location of the processor.
  - The cache only memory access (COMA) model: all processors use only their local cache memory, so that this memory model is a special case of the NUMA model.

- Distributed memory: Distributed memory is a collection of memory pieces where each of them can be accessed by only one processor. If one processor requires data stored in the memory of another processor, then communication between these processors is necessary.

The way to program a distributed memory parallel computer is message passing. The Message Passing Interface (MPI) is a standard developed by the Message Passing Interface Forum (MPIF). It specifies a portable interface for writing message-passing programs, and aims at practicality, efficiency, and flexibility at the same time. MPIF with the participation of more than 40 organizations started working on the standard in 1992. The first draft was published in 1994. The latest release of the first version is offered as an update to the previous release and is contained in the second versions (MPI-2). The design goal of MPI is quoted from "*MPI:A Message-Passing Interface Standard*" as follows:

- Design an application programming interface.
- Allow efficient communication: Avoid memory-to-memory copying and allow overlap of computation and communication and offload to communication co-processor, where available.
- Allow for implementations that can be used in a heterogeneous environment.

- Allow convenient C and Fortran bindings for the interface.
- Define an interface that can be implemented on many platforms.
- The interface should be designed to allow for thread-safety.

Nowadays, all vendors of parallel computers offer standard parallel libraries or similar extensions of the operating system, which include all necessary compilers, libraries, etc. such as High Performance Fortran (HPF), Vienna Fortran, and OpenMPI extensions to C, C++, and Fortran.

Parallel computing is a technology which given an appropriate setup, can solve the problems faster than by serial computer. A problem may have different parallel formulations, which result in varying benefits, and all problems are not equally amenable to parallel processing. To use parallel computing effectively, the following issues need to be examined:

1 Design of parallel computing: It is important to design parallel computing that can scale up to a large number of processors and are capable of supporting fast communication and data among processors.

2 Design of efficient algorithms: A parallel computer is of little use unless efficient parallel algorithms are available. The design of parallel algorithms is different from that of the sequential algorithms.

3 Methods for evaluating parallel algorithms: Given a parallel computer and a parallel algorithm, we need to evaluate the performance of the resulting system. Performance analysis allows us to answer questions such as "*How fast can a problem be solved using parallel processing?*" and "*How efficiently are the processors used?*"

4 Parallel computer languages: Parallel algorithms are implemented on parallel computers using a programming language. The language must be flexible enough to allow the efficient implementation and must be easy to program.

5 Portable parallel programs: Portability is one of the main problems with current parallel computers. Typically, a program written for one parallel computer requires an extensive modification to make it run on another parallel computer.

1.3 Objectives and Overviews of the Thesis

In this research, the mathematical models of the turbulent wake dynamics in a stratified fluid is considered under the following main objectives:

The main objectives of the research work presented in the thesis are

1 to develop the parallel algorithms for the numerical simulation of turbulent wake dynamics in a stratified fluid, and to analyze the performance, accuracy, and range of applicability of numerical models and parallel codes.

2 to apply parallel algorithms to several numerical models of turbulent wake behind towed and self-propelled bodies in a stratified fluid.

3 to validate the parallel algorithms by comparing the results of numerical solutions with the experimental and numerical data on decay of turbulent wake behind towed and self-propelled bodies in a stratified fluid.

4 to estimate the speedup and efficiency of developed algorithms using an example of a particular model of turbulent wake dynamics in a stratified fluid.

The thesis writing is organized as the following. The mathematical models of turbulent wake dynamics in a stratified fluid is described in Chapter II. The details of the sequential algorithms and developed parallel function and domain decomposition algorithms are discussed in Chapter III. The results of validation of parallel algorithms and numerical experiments are reported and discussed in Chapter IV. Conclusion, some general comments and some recommendation for the future work are provided in Chapter V.

CHAPTER II

MATHEMATICAL MODELS OF TURBULENT WAKE DYNAMICS IN A STRATIFIED FLUID

2.1 Introduction

The flow that arises in a turbulent wake behind a body that moves in a stratified fluid is rather peculiar. With a relatively weak stratification, a turbulent wake first develops essentially in the same way as in a homogeneous fluid and extends symmetrically. However, buoyancy forces oppose the vertical turbulent diffusion. Therefore, the wake has a flattened form at large distances from the body and, finally, ceases to extend in the vertical direction. Because of the turbulent mixing, the fluid density within the wake is distributed more uniformly than outside it. Buoyancy forces tend to restore the former unperturbed state of a stable stratification. As a result, convective flows, which give rise to internal waves in an ambient fluid, arise in the plane perpendicular to the wake axis. Shear flows are used to validate model of turbulence, because there is no influence of boundary and they are sensitive to modifications of the Reynolds stresses. Turbulent wake is an example of free shear flow which is often selected to verify the predictability of a turbulent model. Axis pressure gradient and viscous diffusion are small and a major role in determination of the flow field more sensitive to modelling of the Reynolds stress, kinetic energy and dissipation.

2.2 Governing Equations

Turbulent wakes in a stratified fluid have been studied by many researchers. See for example Chernykh and Voropayeva (1999), Spedding (2001), Gourlay *et al.* (2001), Dommermuth *et al.* (2002), Meunier and Spedding (2006) and Moshkin *et al.* (2001) where one can find a sufficiently complete survey.

In a series of papers Chernykh *et al.* (1999, 2008), Moshkin *et al.* (2001) and Voropaeva *et al.* (2000, 2002, 2006, 2008), a hierarchy of semi-empirical turbulence models of second order are involved for the description of fluid flow in far turbulent wake behind a towed and momentumless bodies. Most complex of models include the differential equations for normal Reynolds stresses transfer as well as equations for triple correlations of the vertical velocity fluctuations.

To describe a far turbulent wake flow behind an axisymmetric body in a stratified medium, we use the three-dimensional parabolized system of averaged NavierStokes equations in the Oberbeck-Boussinesq approximation as follows:

$$U_0\frac{\partial U_d}{\partial x}+V\frac{\partial U_d}{\partial y}+W\frac{\partial U_d}{\partial z}=\frac{\partial}{\partial y}\langle u'v'\rangle+\frac{\partial}{\partial z}\langle u'w'\rangle \tag{2.1}$$

$$U_0\frac{\partial V}{\partial x}+V\frac{\partial V}{\partial y}+W\frac{\partial V}{\partial z}=-\frac{1}{\rho_0}\frac{\partial\langle p_1\rangle}{\partial y}-\frac{\partial}{\partial y}\langle v'^2\rangle-\frac{\partial}{\partial z}\langle v'w'\rangle \tag{2.2}$$

$$U_0\frac{\partial W}{\partial x}+V\frac{\partial W}{\partial y}+W\frac{\partial W}{\partial z}=-\frac{1}{\rho_0}\frac{\partial\langle p_1\rangle}{\partial z}-\frac{\partial}{\partial y}\langle v'w'\rangle-\frac{\partial}{\partial z}\langle w'^2\rangle-g\frac{\langle\rho_1\rangle}{\rho_0} \tag{2.3}$$

$$U_0\frac{\partial\langle\rho_1\rangle}{\partial x}+V\frac{\partial\langle\rho_1\rangle}{\partial y}+W\frac{\partial\langle\rho_1\rangle}{\partial z}+W\frac{d\rho_s}{dz}=-\frac{\partial}{\partial y}\langle v'\rho'\rangle-\frac{\partial}{\partial z}\langle w'\rho'\rangle \tag{2.4}$$

$$\frac{\partial V}{\partial y}+\frac{\partial W}{\partial z}=\frac{\partial U_d}{\partial x}. \tag{2.5}$$

In equations (2.1)-(2.5), U_0 is the free stream velocity; $U_d = U_0 - U$ is the defect of the mean free stream velocity component; U, V, W, are velocity components of the mean flow in the direction of the axes x, y, z; $\langle p_1\rangle$ is the deviation of the averaged pressure from the hydrostatic one conditioned by the stratification ρ_s; g is the gravity acceleration; $\langle\rho_1\rangle$ is the mean density defect: $\rho_1 = \rho - \rho_s$, $\rho_s = \rho_s(z)$ is

the undisturbed fluid density: $d\rho_s/dz \leq 0$ (stable stratification), $\rho_0 = \rho_s(0)$; the dash denotes the pulsation components; the symbol $\langle\cdot\rangle$ denotes the averaging. The coordinate system is related to the moving body in such a way that the velocity of its motion is equal to $-U_0$, and the z-axis is directed vertically upwards, in the counter-gravity direction. The fluid density is assumed to be a linear function of temperature and the stratification is assumed to be weak. Both small items involving the derivative with respect to the variable x and the factors in the form of a coefficient of laminar viscosity or diffusion have been omitted in the right hand sides of equations 2.1-2.4. The schematic diagram of the present problem is drawn in Figure 2.1.

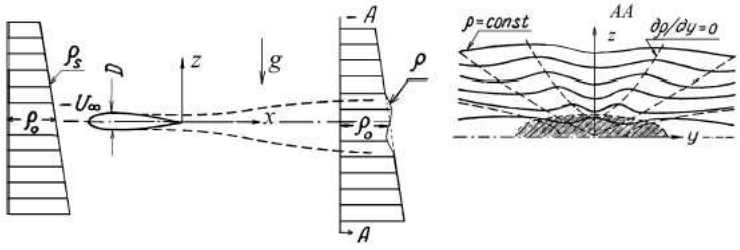

Figure 2.1 Turbulent wake in a stratified fluid

The system of equations (2.1)–(2.5) is not closed. Below we consider 5 models. For Model 1, the unknown values of the Reynolds stresses $\langle u_i'^2\rangle$, $i = 1,2,3$, $\langle u'v'\rangle = \langle u_1'u_2'\rangle$, $\langle u'w'\rangle = \langle u_1'u_3'\rangle$ and the turbulent fluxes $\langle u_i'\rho'\rangle$, $i = 1,2,3$, are determined by the algebraic approximations (see Rodi (1987), Chernykh and Voropayeva (1999) and Chernykh *et al.* (2001)):

$$\frac{\langle u_i'u_j'\rangle}{e} = \frac{2}{3}\delta_{ij} + \frac{1-c_2}{c_1}\left(\frac{P_{ij}}{\varepsilon} - \frac{2}{3}\delta_{ij}\frac{P}{\varepsilon}\right) + \frac{1-c_3}{c_1}\left(\frac{G_{ij}}{\varepsilon} - \frac{2}{3}\delta_{ij}\frac{G}{\varepsilon}\right), \tag{2.6}$$

$$P_{ij} = -\left\{\langle u_i'u_k'\rangle\frac{\partial U_j}{\partial x_k} + \langle u_j'u_k'\rangle\frac{\partial U_i}{\partial x_k}\right\}, \tag{2.7}$$

$$G_{ij} = \frac{1}{\rho_0}(\langle u_i'\rho'\rangle g_j + \langle u_j'\rho'\rangle g_i),\ i,j,k = 1,2,3; \tag{2.8}$$

$$\vec{g} = (0, 0, -g),\ 2P = P_{ii},\ 2G = G_{ii},\ U_1 = U,\ U_2 = V,\ U_3 = W, \tag{2.9}$$

$$-\langle u'\rho'\rangle = \frac{e}{c_{1T}\varepsilon}\left[\langle u'w'\rangle\frac{\partial\langle\rho\rangle}{\partial z} + (1-c_{2T})\langle w'\rho'\rangle\frac{\partial U}{\partial z}\right], \tag{2.10}$$

$$-\langle v'\rho'\rangle = \frac{\langle v'^2\rangle}{c_{1T}}\frac{e}{\varepsilon}\frac{\partial\langle\rho\rangle}{\partial y} = K_{\varrho y}\frac{\partial\langle\rho\rangle}{\partial y}, \tag{2.11}$$

$$\langle\rho'^2\rangle = -\frac{2}{c_T}\frac{e}{\varepsilon}\langle w'\rho'\rangle\frac{\partial\langle\rho\rangle}{\partial z}, \tag{2.12}$$

$$-\langle w'\rho'\rangle = \frac{e}{C_{1T}\varepsilon}\left[\langle w'^2\rangle\frac{\partial\langle\rho\rangle}{\partial z} + (1-c_{2T})\frac{g}{\rho_0}\langle\rho'^2\rangle\right] = \tag{2.13}$$

$$= \frac{e\langle w'^2\rangle}{c_{1T}\varepsilon\left(1 - 2\dfrac{1-c_{2T}}{c_{1T}c_T}\dfrac{g}{\rho_0}\dfrac{e^2}{\varepsilon^2}\dfrac{\partial\langle\rho\rangle}{\partial z}\right)}\frac{\partial\langle\rho\rangle}{\partial z} = K_{\rho z}\frac{\partial\langle\rho\rangle}{\partial z}.$$

Here and below the summation is assumed over repeating indices. To determine the values of the turbulent kinetic energy e, the dissipation ε and the shear Reynolds stress $\langle v'w'\rangle$, we make use of the differential equations:

$$U_0\frac{\partial e}{\partial x} + V\frac{\partial e}{\partial y} + W\frac{\partial e}{\partial z} = \frac{\partial}{\partial y}K_{ey}\frac{\partial e}{\partial y} + \frac{\partial}{\partial z}K_{ez}\frac{\partial e}{\partial z} + P + G - \varepsilon, \tag{2.14}$$

$$U_0\frac{\partial\varepsilon}{\partial x} + V\frac{\partial\varepsilon}{\partial y} + W\frac{\partial\varepsilon}{\partial z} = \frac{\partial}{\partial y}K_{\varepsilon y}\frac{\partial\varepsilon}{\partial y} + \frac{\partial}{\partial z}K_{\varepsilon z}\frac{\partial\varepsilon}{\partial z} + c_{\varepsilon 1}\frac{\varepsilon}{e}(P+G) - c_{\varepsilon 2}\frac{\varepsilon^2}{e}, \tag{2.15}$$

$$U_0\frac{\partial\langle v'w'\rangle}{\partial x} + V\frac{\partial\langle v'w'\rangle}{\partial y} + W\frac{\partial\langle v'w'\rangle}{\partial z} = \frac{\partial}{\partial y}K_{ey}\frac{\partial\langle v'w'\rangle}{\partial y} + \tag{2.16}$$

$$+\frac{\partial}{\partial z}K_{ez}\frac{\partial\langle v'w'\rangle}{\partial z} + (1-c_2)P_{23} + (1-c_3)G_{23} - c_1\frac{\varepsilon}{e}\langle v'w'\rangle,$$

where the turbulent viscosity coefficients are defined from simplified relation (2.6) as follows

$$K_{ey} = \frac{1-c_2}{c_1}\cdot\frac{e\langle v'^2\rangle}{\varepsilon},\quad K_{\varepsilon y} = \frac{K_{ey}}{\sigma},$$

$$K_{ez} = \frac{\left[(1-c_2)e\langle w'^2\rangle - \dfrac{(1-c_3)(1-c_{2T})}{c_{1T}}\dfrac{e^2}{\varepsilon}\dfrac{g}{\rho_0}\langle w'\rho\rangle\right]}{c_1\varepsilon\left(1 - \dfrac{(1-c_3)}{c_1c_{1T}}\dfrac{g}{\rho_0}\dfrac{e^2}{\varepsilon^2}\dfrac{\partial\langle\rho\rangle}{\partial z}\right)};\quad K_{\varepsilon z} = \frac{K_{ez}}{\sigma},$$

So that

$$-\langle u'v'\rangle = K_{ey}\frac{\partial U}{\partial y}, \quad -\langle u'w'\rangle = K_{ez}\frac{\partial U}{\partial z}.$$

The quantities $c_1,\ c_2,\ c_3,\ c_{1T},\ c_{2T},\ c_T,\ c_{\varepsilon 1},\ c_{\varepsilon 2},\ \sigma$ are empirical constants. Their values are taken to be equal to 2.2, 0.55, 0.55, 3.2, 0.5, 1.25, 1.45, 1.9, 1.3 , respectively. The choice of this model of turbulence is due to the following reasons: it is close to the standard $e-\varepsilon$ model of turbulence and we can take into account the anisotropy of the turbulence characteristics in the wakes in a stratified fluid.

Model 2 is similar to the one presented by Hassid (1980). The main difference from Model 1 is in the use of modified local equilibrium approximation ($P=\varepsilon$) for the determination of the components of the tensor of Reynolds stresses (instead of isotropic relations)

$$\frac{\langle u_i'u_j'\rangle}{e} = -\frac{2}{3}\frac{(1-c_2-c_1)}{c_1}\delta_{ij} + \frac{(1-c_2)}{c_1}\frac{P_{ij}}{\varepsilon} + \frac{(1-c_3)}{c_1}\frac{G_{ij}}{\varepsilon}. \tag{2.17}$$

For Model 3 (unlike Model 1), we use the following representation of the turbulent viscosity coefficients

$$K_{ey} = C_s\frac{e\langle v'^2\rangle}{\varepsilon}, \quad K_{ez} = C_s\frac{e\langle w'^2\rangle}{\varepsilon}, \quad K_{\varepsilon y} = \frac{K_{ey}}{\sigma}, \quad K_{\varepsilon z} = \frac{K_{ez}}{\sigma}, \quad C_s = 0.25. \tag{2.18}$$

Otherwise, Model 3 is analogous to Model 1.

For Model 4, the values $\langle u_i'^2\rangle$ $(i=1,2,3)$ are calculated by solving the corresponding transport differential equations:

$$U_0\frac{\partial\langle u'^2\rangle}{\partial x} + V\frac{\partial\langle u'^2\rangle}{\partial y} + W\frac{\partial\langle u'^2\rangle}{\partial z} = \frac{\partial}{\partial y}K_{ey}\frac{\partial\langle u'^2\rangle}{\partial y} + \frac{\partial}{\partial z}K_{ez}\frac{\partial\langle u'^2\rangle}{\partial z} + P_{11} + G_{11}$$
$$-\frac{2}{3}\varepsilon - C_1\frac{\varepsilon}{e}\left(\langle u'^2\rangle - \frac{2}{3}e\right) - C_2\left(P_{11} - \frac{2}{3}P\right) - C_2\left(G_{11} - \frac{2}{3}G\right), \tag{2.19}$$

$$U_0\frac{\partial\langle v'^2\rangle}{\partial x} + V\frac{\partial\langle v'^2\rangle}{\partial y} + W\frac{\partial\langle v'^2\rangle}{\partial z} = \frac{\partial}{\partial y}K_{ey}\frac{\partial\langle v'^2\rangle}{\partial y} + \frac{\partial}{\partial z}K_{ez}\frac{\partial\langle v'^2\rangle}{\partial z} + P_{22} + G_{22}$$

$$-\frac{2}{3}\varepsilon - C_1\frac{\varepsilon}{e}\left(\langle v'^2\rangle - \frac{2}{3}e\right) - C_2\left(P_{22} - \frac{2}{3}P\right) - C_2\left(G_{22} - \frac{2}{3}G\right), \qquad (2.20)$$

$$U_0\frac{\partial\langle w'^2\rangle}{\partial x} + V\frac{\partial\langle w'^2\rangle}{\partial y} + W\frac{\partial\langle w'^2\rangle}{\partial z} = \frac{\partial}{\partial y}K_{ey}\frac{\partial\langle w'^2\rangle}{\partial y} + \frac{\partial}{\partial z}K_{ez}\frac{\partial\langle w'^2\rangle}{\partial z} + P_{33} + G_{33}$$

$$-\frac{2}{3}\varepsilon - C_1\frac{\varepsilon}{e}\left(\langle w'^2\rangle - \frac{2}{3}e\right) - C_2\left(P_{33} - \frac{2}{3}P\right) - C_2\left(G_{33} - \frac{2}{3}G\right), \qquad (2.21)$$

$$K_{ey} = C_s\frac{e}{\varepsilon}\langle v'^2\rangle,\ K_{ez} = C_s\frac{e}{\varepsilon}\langle w'^2\rangle,\ e = \left(\langle u'^2\rangle + \langle v'^2\rangle + \langle w'^2\rangle\right)/2.$$

$$P_{11} = 2\left(K_y\left(\frac{\partial U_d}{\partial y}\right)^2 + K_z\left(\frac{\partial U_d}{\partial z}\right)^2\right) = 2P,\quad G_{11} = 0,\ P_{22} = 0,\quad G_{22} = 0,$$

$P_{33} = 0$,

$$G_{33} = -2\frac{g}{\rho_0}\langle w'\rho'\rangle = 2G,\quad P = \langle u'v'\rangle\frac{\partial U_d}{\partial y} + \langle u'w'\rangle\frac{\partial U_d}{\partial z} = K_y\left(\frac{\partial U_d}{\partial y}\right)^2 + K_z\left(\frac{\partial U_d}{\partial z}\right)^2,$$

$G = -\frac{g}{\rho_0}\langle w'\rho'\rangle = \frac{g}{\rho_0}K_{\rho z}\frac{\partial\rho}{\partial z}$. Coefficients of viscosity are given by equations (2.18)

Model 5 includes the transport differential equation for the triple correlation of vertical fluctuation component $\langle w'^3\rangle$

$$U_0\frac{\partial\langle w'^3\rangle}{\partial x} + V\frac{\partial\langle w'^3\rangle}{\partial y} + W\frac{\partial\langle w'^3\rangle}{\partial z} = \frac{\partial}{\partial y}K_{3y}\frac{\partial\langle w'^3\rangle}{\partial y} + \frac{\partial}{\partial z}K_{3z}\frac{\partial\langle w'^3\rangle}{\partial z}$$
$$-3\left\{\langle v'w'\rangle\frac{\partial w'^2}{\partial y} + \langle w'^2\rangle\frac{\partial w'^2}{\partial z}\right\} - 3\frac{g}{\rho_0}\langle w'^2\rho'\rangle - C_{3w}\frac{\langle w'^3\rangle\,\varepsilon}{e}, \qquad (2.22)$$

where

$$K_{3y} = \frac{1}{c_4}\frac{e}{\varepsilon}\frac{\langle v'^2\rangle}{1 - \frac{1}{c_4c_{4\theta}}\frac{e^2}{\varepsilon^2}\frac{\partial\langle\rho\rangle}{\partial z}},\quad K_{3z} = \frac{4}{c_4}\frac{e}{\varepsilon}\frac{\langle w'^2\rangle}{1 - \frac{4}{c_4c_{4\theta}}\frac{g}{\rho_0}\frac{e^2}{\varepsilon^2}\frac{\partial\langle\rho\rangle}{\partial z}}$$

$$-\langle w'\rho'\rangle = 2c_{s\varphi}\frac{e}{\varepsilon}\left(\langle w'^2\rangle\frac{\partial\langle w'\rho'\rangle}{\partial z}\right);$$

$$c_{3w} = 13.6,\ c_4 = 2c_{3w} - 2,\ c_{4\theta} = 2/c_{s\varphi} - 1,\ \ c_{s\varphi} = 0.11$$

The applicability of Models 1-4 to the calculation of momentumless wakes has been analyzed in detail in Chernykh and Voropayeva (1999). However, turbulent wakes behind towed bodies even in a homogeneous fluid substantially differ from

those behind self-propelled bodies. Model 5 is a simplified variant of a closer model considered in Voropaeva *et al.* (2002). Therefore, the applicability of Models 1-5 to the calculation of turbulent wakes behind towed bodies in a linearly stratified medium can be decided after detailed numerical experiments.

2.3 Initial and Boundary Conditions

The marching variable x in equations (2.1)-(2.4), (2.14)-(2.16), (2.19)-(2.21) and (2.22) plays the role of time. At the distance $x = x_0$ from the body the following initial conditions are specified

$$U_d(x_0, y, z) = \Theta_1(r), e(x_0, y, z) = \Theta_2(r), \varepsilon(x_0, y, z) = \Theta_3(r), r^2 = y^2 + z^2, 0 \le r < \infty;$$

$$\langle v'w' \rangle = \langle \rho_1 \rangle = V = W = 0, \ \ -\infty < z < \infty, \ \ -\infty < y < \infty, x = x_0.$$

Here $\Theta_1(r)$, $\Theta_2(r)$ and $\Theta_3(r)$ are the functions consistent with the experimental data of Lin and Pao (1979) and Hassid (1980) in the homogeneous fluid. There are additional initial data in the case of Model 4 and Model 5 $\langle u_i'^2 \rangle = \frac{2}{3}e, \ \ i = 1, 2, 3,$ $\langle w'^3 \rangle = -c_s \frac{e\langle w'^2 \rangle}{\varepsilon} \frac{\partial \langle w'^2 \rangle}{\partial z}$. At $r \to \infty$ the free stream conditions are specified (Models 1-3)

$$U_d = V = W = \langle \rho_1 \rangle = e = \varepsilon = \langle v'w' \rangle = 0, \quad x \ge x_0. \tag{2.23}$$

From the symmetry considerations, the solution is determined only in the first quadrant of the (y, z) plane using the following boundary conditions:

$$\langle v'w' \rangle = \frac{\partial \langle \rho_1 \rangle}{\partial y} = V = \frac{\partial W}{\partial y} = \frac{\partial U_d}{\partial y} = \frac{\partial e}{\partial y} = \frac{\partial \varepsilon}{\partial y} = 0, \ y = 0, \ z \ge 0,$$

$$\langle v'w' \rangle = \langle \rho_1 \rangle = W = \frac{\partial V}{\partial z} = \frac{\partial U_d}{\partial z} = \frac{\partial e}{\partial z} = \frac{\partial \varepsilon}{\partial z} = 0, \ z = 0, \ y \ge 0.$$

Models 4 and 5 are supplemented by symmetry conditions for normal components of Reynolds stresses $\langle u_i'^2 \rangle, \quad i = 1, 2, 3$ and $\langle w'^3 \rangle = 0, \ z = 0, \ y \ge 0; \quad \frac{\partial \langle w'^3 \rangle}{\partial y} =$

$0, \quad y = 0, \ z \geq 0$. In the numerical solution of the problem, the boundary conditions (2.23) corresponding to $r \to \infty$ are translated to the boundaries of a sufficiently large rectangle $0 \leq y \leq y_*$; $0 \leq z \leq z_*$.

The problem variables can be made dimensionless by using the characteristic length D, the body diameter, and the velocity scale U_0. As a result, the value $4\pi^2/{F_d}^2$ will appear in the dimensionless equations instead of g, where F_d is the density Froude number defined as

$$F_d = \frac{U_0 T}{D}, \ \ T = \frac{2\pi}{\sqrt{ag}} = \frac{1}{N}, \ \ a = -\left(\frac{1}{\rho_0}\right)\frac{d\rho_s}{dz},$$

where T, N are the Brunt-Vaisala period and frequency. For the interpretation of the computational results, it is convenient to introduce the time t related to the distance from the body

$$t = \frac{x}{U_0}, \ t^* = \frac{t}{T} = \frac{xD}{U_0 DT} = \frac{x^*}{F_d}.$$

2.4 Mathematical Model of Dynamics of Passive Scalar in Turbulent Wakes in a Stratified Fluids

There is almost no research devoted to the dynamics of passive scalar in the turbulent wakes behind self-propelled and towed bodies in a stratified fluids. Passive scalar transport in turbulent wakes behind a body of revolution in a linearly stratified media is considered in Moshkin *et al.* (2004). Along with the above Models 1-5, the equations for averaged concentration of passive scalar Θ and dispersion of fluctuations $\langle \theta'^2 \rangle$ are solved and they are given by

$$U_0\frac{\partial \Theta}{\partial x} + V\frac{\partial \Theta}{\partial y} + W\frac{\partial \Theta}{\partial z} = \frac{\partial}{\partial y}K_{\Theta y}\frac{\partial \Theta}{\partial y} + \frac{\partial}{\partial z}K_{\Theta z}\frac{\partial \Theta}{\partial z},$$

$$\frac{\partial \langle \theta'^2 \rangle}{\partial z} = \frac{\partial}{\partial y}K_{1\Theta y}\frac{\partial \langle \theta'^2 \rangle}{\partial y} + \frac{\partial}{\partial z}K_{1\Theta z}\frac{\partial \langle \theta'^2 \rangle}{\partial z} - 2\langle v'\theta' \rangle \frac{\partial \Theta}{\partial y} - 2\langle w'\theta' \rangle \frac{\partial \Theta}{\partial z} - N_\Theta,$$

where

$$K_{\Theta y} = \frac{e\,\langle v'^2\rangle}{C_{1T}\varepsilon},\quad K_{\Theta z} = \frac{e\,\langle w'^2\rangle}{C_{1T}\varepsilon}\,,\ K_{1\Theta y} = \frac{C_\phi\,\langle v'^2\rangle\,e}{\varepsilon},$$

$$K_{1\Theta z} = \frac{C_\phi\,\langle w'^2\rangle\,e}{\varepsilon},\ \langle v'\theta'\rangle = -K_{\Theta y}\frac{\partial\Theta}{\partial y},\quad \langle w'\theta'\rangle = -K_{\Theta z}\frac{\partial\Theta}{\partial z},\ N_\Theta = C_T\frac{\langle\theta'^2\rangle\,\varepsilon}{e}\,.$$

The quantities c_{1T} , c_φ are empirical constants. The bell-shaped functions are used as initial conditions for Θ , $\langle\theta'^2\rangle$.

CHAPTER III

SEQUENTIAL AND PARALLEL ALGORITHMS

In this chapter, we describe the sequential algorithm and the idea of parallel functional and domain decomposition algorithms to solve the mathematical models of turbulent wake dynamics in a stratified fluid.

3.1 Sequential Algorithm

For the construction of a finite difference scheme, the new independent variables are introduced

$$x' = x, \;\; \xi = \chi_1(y), \;\; \eta = \chi_2(z), \;\; (x = x', \;\; y = \phi_1(\xi), \;\; z = \phi_2(\eta)). \tag{3.1}$$

This mapping is used to transform the nonuniform mesh in a physical space (x, y, z) into a uniform rectangular mesh in a computational domain (x', ξ, η). The governing equations (2.1)-(2.5) and (2.14)-(2.16) recast according to mesh transformation (3.1). The functions ϕ_1 and ϕ_2 establish the one-to-one correspondence between nodes of a uniform mesh in the computational domain and nodes of a nonuniform mesh in the physical domain. The functions ϕ_1 and ϕ_2 are constructed by tubular assigning points in the physical plane to the corresponding points in computational domain. The metrics coefficients are computed by using finite differences. The choice of mapping (3.1) enables us to condense the mesh nodes in the turbulent wake neighborhood. In the computational domain (x', ξ, η) the nodes of the mesh

in the (ξ, η) plane are distributed uniformly:

$$\xi_i = i \cdot \triangle\xi, \;\; \eta_j = j \cdot \triangle\eta, \;\; i = 0, \ldots, N, \;\; j = 0, \ldots, M,$$

$$\varphi_1(\xi_N) = y_*, \;\; \varphi_2(\eta_M) = z_*.$$

A staggered arrangement of the unknown functions is used:

- Scalar functions as $\langle p_1 \rangle$, $\langle \rho_1 \rangle$, U_d, e, ε, and components of the Reynolds stresses, $\langle u_l' u_k' \rangle$ are located at the cell center $(\xi_i, \eta_j) = (i \cdot \triangle\xi, j \cdot \triangle\eta)$.
- The horizontal V and vertical W velocity components of the mean velocity vector are located at the centers of the cell sides normal to them.

In Figure 3.1, a staggered grid is sketched.

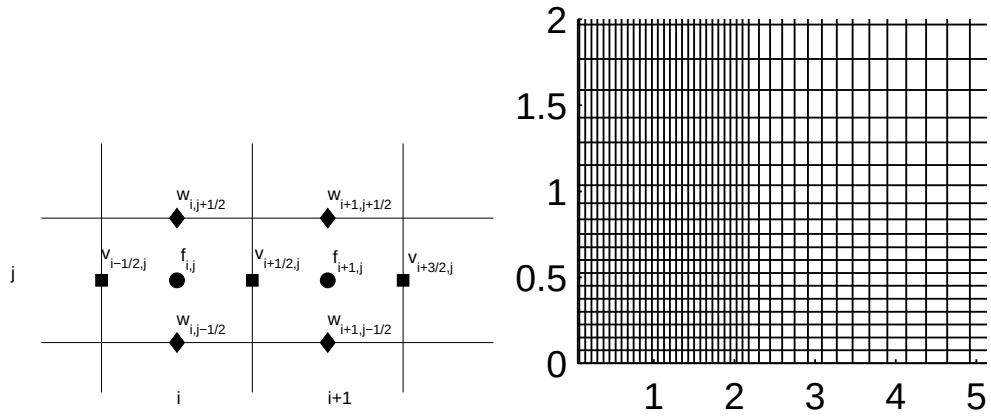

Figure 3.1 Staggered grid: filled circles denote $f = \langle p_1 \rangle$, $\langle \rho_1 \rangle$, U_d, e, ε or $\langle u_l' \, u_k' \rangle$, filled diamonds denote vertical W z-velocity, filled squares denote V y-velocity component

Let us use the following notations: the upper index n denotes values of variables in the wake cross section $x = x^n = x^{n-1} + \triangle x^n$, lower i, j indexes denote the quantities at the node $\xi_i = i \cdot \triangle\xi$, $\eta_j = j \cdot \triangle\eta$, and lower indexes $i{+}1/2$, $j{+}1/2$ correspond to the node $\xi_{i+1/2} = (i{+}1/2) \cdot \triangle\xi$, $\eta_{j+1/2} = (j{+}1/2) \cdot \triangle\eta$.

Algorithm of the problem solution is based on the implicit splitting into space variables for the equations (2.1), (2.4), (2.14)-(2.16), (2.19)-(2.21), and (2.22) and on the application of an explicit splitting into physical processes to the system of equations (2.2), (2.3), and (2.5). Let all unknown functions be given at $x = x^n$. The solution procedure consists of the following steps for computing all unknown functions in the x direction.

1. The defect of the mean streamwise velocity component U_d is computed by evaluating the finite-difference approximation of equation (2.1).

2. The velocity components of averaged motion V^{n+1}, W^{n+1}, and the pressure deviation from hydrostatic, $\langle p_1 \rangle^{n+1}$, are computed by using equations (2.2), (2.3), and (2.5). Here we utilize the idea of the splitting method in which the computational process is divided into three stages. The first stage consists in computing the provisional velocity components V^*, W^* with explicit approximation of equations (2.2), (2.3). Then in a second stage, we compute $\langle p_1 \rangle^{n+1}$ from solution to the Poisson equation with the Neumann boundary conditions along axis of symmetry and the Dirichlet boundary conditions along far boundaries. The method of stabilizing correction (Yanenko (1971)) is used in this stage. Finally, in the third stage, the new velocity vector components V^{n+1}, W^{n+1} are computed from requirement of vanish divergence.

3. Equations (2.4), (2.14)-(2.16), (2.19)-(2.21), and (2.22) are sequentially integrated with the use of the method of splitting in term of spatial variables (Yanenko (1971)).

From the consideration of a simple computer code implementation, we have used the idea of a block analogy of the well-known Seidel method. At the compu-

tation of the functions $\langle\rho_1\rangle^{n+1}$, e^{n+1}, ε^{n+1}, $\langle v'w'\rangle^{n+1}$ we have used the quantities already known at this level $x = x^{n+1}$, the remaining functions are taken from the previous level $x = x^n$. As an example, we give the time and space discretization used in order to solve the transport equations (2.1), (2.4), (2.14)-(2.16), (2.19)-(2.21), and (2.22) by splitting scheme. Let us consider typical transport equation written in new coordinates (3.1)

$$\frac{\partial f}{\partial x}+\frac{1}{J}\frac{\partial(z_\eta V f)}{\partial\xi}+\frac{1}{J}\frac{\partial(y_\xi W f)}{\partial\eta}=\frac{1}{J}\frac{\partial}{\partial\xi}\left(\frac{z_\eta}{y_\xi}K_y\frac{\partial f}{\partial\xi}\right)+\frac{1}{J}\frac{\partial}{\partial\eta}\left(\frac{y_\xi}{z_\eta}K_z\frac{\partial f}{\partial\eta}\right)+Q_f. \quad (3.2)$$

Here z_η, y_ξ denote derivatives with respect lower index, $J=\dfrac{\partial(x,y,z)}{\partial(x',\xi,\eta)}=y_\xi\ z_\eta$ is the Jacobian of the transformation (3.1) and f denotes one of unknown functions $U_d,\ e,\ \varepsilon,\ \langle\rho_1\rangle$, and so on, Q_f is corresponding source terms. To simplify the notations, we introduce the quantities at the nodes obtained by interpolation

$$f_{i,j+1/2}=(f_{i,j+1}+f_{i,j})\times 0.5,\quad f_{i+1/2,j}=(f_{i+1,j}+f_{i,j})\times 0.5,$$

and derivatives of coordinate transformations

$$(z_\eta)_{i,j+1/2}=\frac{z_{j+1}-z_j}{h_\eta},\quad (y_\xi)_{i+1/2,j}=\frac{y_{i+1}-y_i}{h_\xi}.$$

The splitting scheme for equation (3.2) is the following

$$\frac{(f)^{n+1/2}_{i,j}-(f)^n_{i,j}}{h_x}+\frac{1}{J_{i,j}}\frac{(z_\eta V^n f^{n+1/2})_{i+1/2,j}-(z_\eta V^n f^{n+1/2})_{i-1/2,j}}{h_\xi}=(Q_f)_{i,j}$$
$$+\frac{1}{J_{i,j}}\frac{\left(\widehat{K}_y\right)_{i+1/2,j}\left((f)^{n+1/2}_{i+1,j}-(f)^{n+1/2}_{i,j}\right)-\left(\widehat{K}_y\right)_{i-1/2,j}\left((f)^{n+1/2}_{i,j}-(f)^{n+1/2}_{i-1,j}\right)}{h_\xi^2}, \quad (3.3)$$

$$\frac{(f)^{n+1}_{i,j}-(f)^{n+1/2}_{i,j}}{h_x}+\frac{1}{J_{i,j}}\frac{(y_\xi W^n f^{n+1})_{i,j+1/2}-(y_\xi W^n f^{n+1})_{i,j-1/2}}{h_\eta}=$$
$$\frac{1}{J_{i,j}}\frac{\left(\widehat{K}_z\right)_{i,j+1/2}\left((f)^{n+1}_{i,j+1}-(f)^{n+1}_{i,j}\right)-\left(\widehat{K}_z\right)_{i,j-1/2}\left((f)^{n+1}_{i,j}-(f)^{n+1}_{i,j-1}\right)}{h_\eta^2}. \quad (3.4)$$

Here $\widehat{K}_y = z_\eta / y_\xi K_y$, $\widehat{K}_z = y_\xi / z_\eta K_z$. Equations (3.3) and (3.4) are solved sequentially by using direct method for tridiagonal algebraic system.

Equations (2.2), (2.3), and (2.5) for the mean deviation of pressure and velocity vector components in wakes cross section are similar to $2-D$ incompressible Navier-Stokes equations in which variable x plays role of the time. In the new coordinate system dimensionless equations (2.2), (2.3), and (2.5) have the following form

$$\frac{\partial V}{\partial x} + \frac{1}{J}\frac{\partial (z_\eta V^2)}{\partial \xi} + \frac{1}{J}\frac{\partial (y_\xi WV)}{\partial \eta} = -\frac{\partial z_\eta \langle p_1 \rangle}{\partial \xi} + F_1(\langle \rho_1 \rangle, e, \varepsilon, \langle v'w' \rangle), \tag{3.5}$$

$$\frac{\partial W}{\partial x} + \frac{1}{J}\frac{\partial (z_\eta VW)}{\partial \xi} + \frac{1}{J}\frac{\partial (y_\xi W^2)}{\partial \eta} = -\frac{\partial y_\xi \langle p_1 \rangle}{\partial \eta} + F_2(\langle \rho_1 \rangle, e, \varepsilon, \langle v'w' \rangle), \tag{3.6}$$

$$\frac{1}{J}\frac{\partial}{\partial \xi}(z_\eta V) + \frac{1}{J}\frac{\partial}{\partial \eta}(y_\xi W) = \frac{\partial U_d}{\partial x}. \tag{3.7}$$

Here F_1, F_2 are represent right-hand side terms in equations (2.2), (2.3). The partial derivative $\dfrac{\partial U_d}{\partial x}$ in (3.7) is approximated by the forward finite difference

$$\left(\frac{\partial U_d}{\partial x}\right)^{n+1}_{i,j} \simeq \frac{(U_d)^{n+1}_{i,j} - (U_d)^n_{i,j}}{h_x^{n+1}} = \alpha^{n+1}_{i,j}.$$

The three stage computational process is as follows:

- First, equations (3.5), (3.6) are solved without pressure terms. We use explicit approximation, terms F_1, F_2 are evaluated on level $x = x^n$

$$\frac{\widetilde{V}_{i+1/2,j} - V^n_{i+1/2,j}}{h_x^{n+1}} + \frac{1}{J_{i+1/2,j}}\frac{(z_\eta V^2)^n_{i+1,j} - (z_\eta V^2)^n_{i,j}}{h_\xi} +$$

$$+\frac{(y_\xi WV)^n_{i+1/2,j+1/2} - (y_\xi WV)^n_{i+1/2,j-1/2}}{J_{i+1/2,j} h_\eta} = (F_1(\langle \rho_1 \rangle, e, \varepsilon, \langle v'w' \rangle))_{i+1/2,j}, \tag{3.8}$$

$$\frac{\widetilde{W}_{i,j+1/2} - W^n_{i,j+1/2}}{h_x} + \frac{1}{J_{i,j+1/2}}\frac{(z_\eta VW)^n_{i+1/2,j+1/2} - (z_\eta VW)^n_{i-1/2,j+1/2}}{h_\xi} +$$

$$+\frac{1}{J_{i,j+1/2}}\frac{(y_\xi W^2)^n_{i,j+1} - (y_\xi W^2)^n_{i,j}}{h_\eta} = (F_2(\langle \rho_1 \rangle, e, \varepsilon, \langle v'w' \rangle))_{i,j+1/2}. \tag{3.9}$$

- In the second stage, $\langle p_1 \rangle^{n+1}$ is computed from approximate solution of the Poisson equation

$$\frac{\partial}{\partial \xi}\left[\frac{z_\eta}{y_\xi}\frac{\partial \langle p_1 \rangle}{\partial \xi}\right]^{n+1} + \frac{\partial}{\partial \eta}\left[\frac{y_\xi}{z_\eta}\frac{\partial \langle p_1 \rangle}{\partial \eta}\right]^{n+1} =$$

$$\frac{1}{h_x^{n+1}}\left\{\frac{\partial}{\partial \xi}\left(z_\eta \widetilde{V}\right) + \frac{\partial}{\partial \eta}\left(y_\xi \widetilde{W}\right) - J \cdot \alpha^{n+1}\right\}. \tag{3.10}$$

  As mentioned above, the iterative scheme of stabilizing correction is utilized to solve (3.10). Zero Direchlet boundary conditions are used at far boundaries $z = z_*$ and $y = y_*$ ($\langle p_1 \rangle_{i,N_{z_*}}^{n+1} = 0$, $\langle p_1 \rangle_{N_{y*},j}^{n+1} = 0$). At the axes of symmetry $z = 0$ and $y = 0$ the Neumann boundary conditions are approximated in the following way

$$\frac{\langle p_1 \rangle_{2,j} - \langle p_1 \rangle_{1,j}}{h_\xi} = 0, \quad \frac{\langle p_1 \rangle_{i,2} - \langle p_1 \rangle_{i,1}}{h_\eta} = 0. \tag{3.11}$$

- In the third stage, the velocity components W^{n+1}, V^{n+1} are determined as follows:

$$V_{i+1/2,j}^{n+1} = \widetilde{V}_{i+1/2,j} - h_x^{n+1}\frac{1}{J_{i+1/2,j}} \cdot \left(\frac{(z_\eta \langle p_1 \rangle)_{i+1,j}^{n+1} - (z_\eta \langle p_1 \rangle)_{i,j}^{n+1}}{h_\xi}\right), \tag{3.12}$$

$$W_{i,j+1/2}^{n+1} = \widetilde{W}_{i,j+1/2} - h_x^{n+1}\frac{1}{J_{i,j+1/2}} \cdot \left(\frac{(y_\xi \langle p_1 \rangle)_{i,j+1}^{n+1} - (y_\xi \langle p_1 \rangle)_{i,j}^{n+1}}{h_\eta}\right). \tag{3.13}$$

Flowchart of the sequential algorithm of turbulent wake dynamics in a stratified fluid is depicted in Figure 3.3.

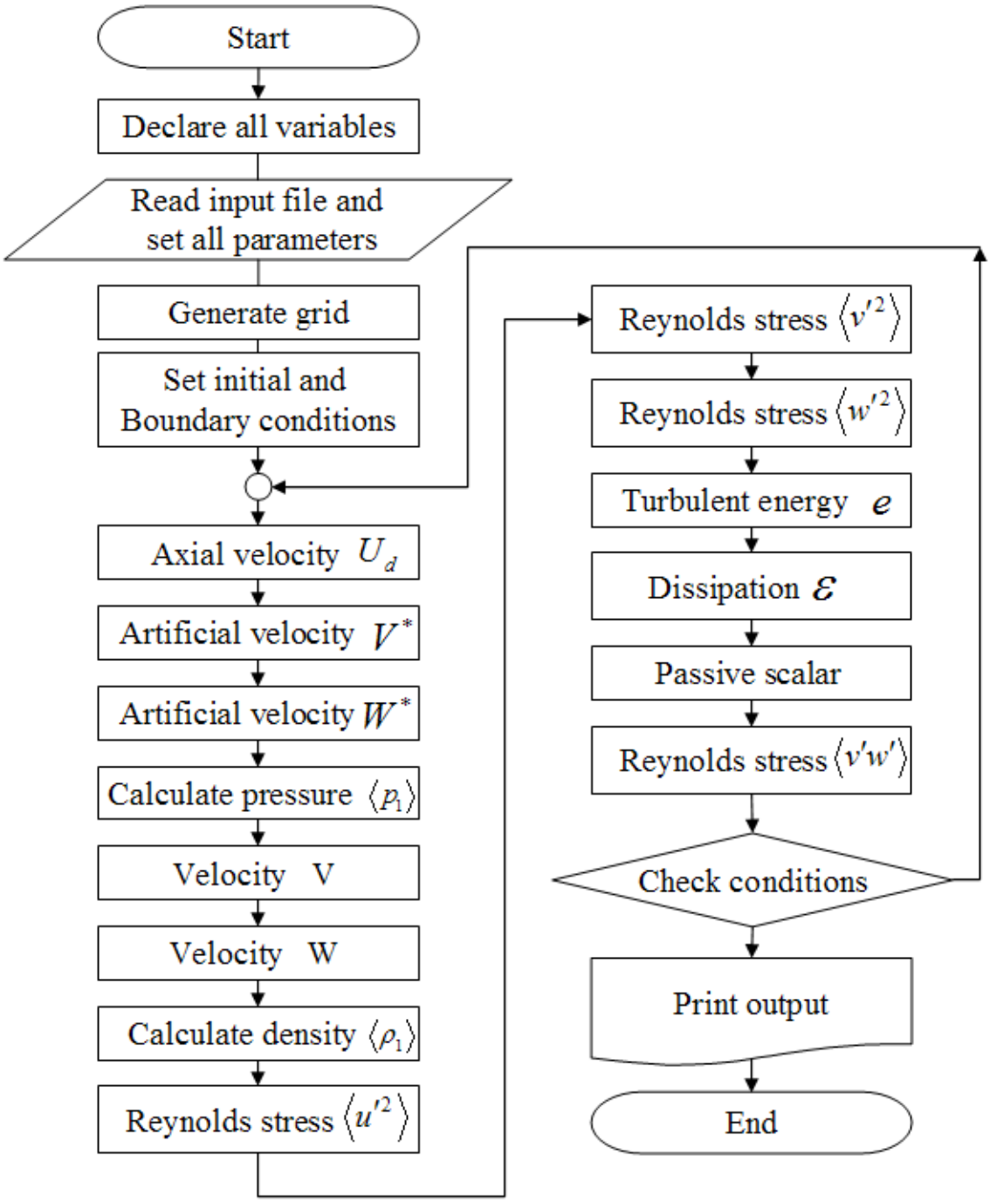

Figure 3.2 Flowchart of sequential algorithm for Model 4 with equation for passive scalar

In order to check the accuracy and the efficiency of the mathematical models and numerical algorithms, we have carried out a series of numerical experi-

ments. The calculations are conducted on a grid sequence and are compared with experimental data of Lin and Pao (1979) and Hassid (1980) on the decay of the momentumless and drag turbulent wakes in a linearly stratified medium. At $x = x_0$ the initial conditions were specified according to Hassid (1980), which agreed with the experimental data of Lin and Pao (1979) on the decay of a turbulent wake in a homogeneous fluid.

a) Momentumless wake

$$U_d(x_0, y, z) = \Theta_1(r) = U_{d0}\left(1 - \frac{8r^2}{D^2}\right)\exp\left(-\frac{8r^2}{D^2}\right),$$

$$e(x_0, y, z) = \Theta_2(r) = E_0 \cdot \exp\left(-\frac{4r^2}{D^2}\right),$$

$$\varepsilon(x_0, y, z) = \Theta_3(r) = \frac{\sqrt{12}}{D}E_0^{3/2} \cdot \exp\left(-\frac{6r^2}{D^2}\right),$$

b) Drag wake

$$U_d(x_0, y, z) = \tilde{\Theta}_1(r) = U_{d0}\exp\left(-\frac{r^2}{A_0}\right),$$

$$e(x_0, y, z) = \tilde{\Theta}_2(r) = E_0 \cdot \exp\left(-\frac{r^2}{A_0}\right),$$

$$\varepsilon(x_0, y, z) = \tilde{\Theta}_3(r) = \sqrt{\frac{3}{A_0}} \cdot E_0^{3/2} \cdot \exp\left(-\frac{3}{2}\frac{r^2}{A_0}\right),$$

$$A_0 = \frac{c_d D^2 U_0}{8 \cdot U_{d0}}.$$

Here c_d is a drag coefficient, and the values of E_0 and U_{d0} are the initial values of the turbulent energy and the velocity centerline defect, respectively. These quantities are chosen to satisfy the experimental data of Lin and Pao (1974,1979) at $x = x_0$.

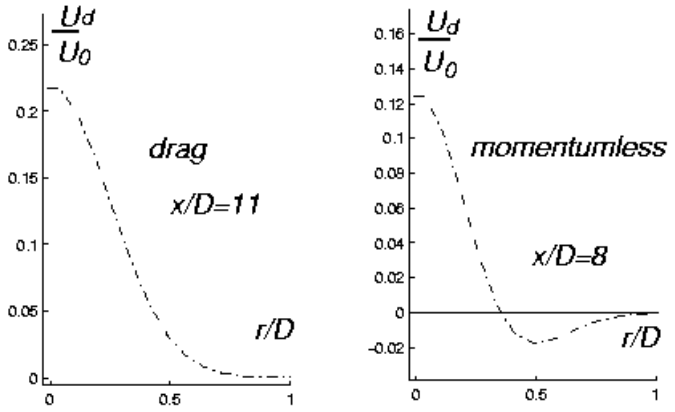

Figure 3.3 Sketch of axial velocity profile at $x = x_0$ in case of momentumless and drag wakes

The main calculations are performed on a grid with 72×37 nodes in the yz-plane. The nodes of the grid in domain are distributed as follows

$$y_i = i \cdot h_y, \;\; i = 0, \ldots, 31; \;\; y_i = y_{i-1} \cdot q_y, \;\; i = 32, \ldots, 72, \;\; q_y = 1.06,$$

$$z_j = j \cdot h_z, \;\; j = 0, \ldots, 11; \;\; z_j = z_{j-1} \cdot q_z, \;\; j = 12, \ldots, 37, \;\; q_z = 1.113.$$

where $h_y = h_z = 0.075$. The step in marching direction h_x^n is varied from $h_x^0 = 0.055$ to $h_x^{max} = 2.0$ by the formula $h_x^{n+1} = h_x^n + 0.055$ and is further assumed to be constant. The refinement of the mesh cell sizes in the wake neighborhood has led to the deviations in the quantities $\sqrt{e_0}$, U_{D0} which do not exceed $1 - 3\%$.

3.2 Parallel Algorithms

We analyze the sequential algorithm and find out that some components can be executed simultaneously, for example, the Reynolds stresses can be computed simultaneously. So in this section, we introduce the techniques of parallel functional and domain decomposition algorithms and how to apply these techniques

to the mathematical models of turbulent wake dynamics behind an axisymmetric body in a stratified fluid.

3.2.1 Parallel Functional Decomposition

For a parallel functional decomposition, the functions to be performed on data are split into multiple jobs. These jobs can then be performed concurrently by different processes on different data. This decomposition also has an important role to play as a program structure technique. The parallel functional decomposition that partitions not only the computation to be performed but also the codes performing the computation is likely to reduce the complexity of the overall design. This is often the case in the computer models of complex systems. For example, a simulation of the earth's climate may comprise components representing the atmosphere, ocean, hydrology, ice, carbon dioxide sources, and so on.

The problem of turbulent wake dynamics in a stratified fluid is reduced to the successive integration system of the transport equations to find averaged velocity components, turbulent energy, dissipation rate, components of the Reynolds stress tensor and so on.

Suppose we have M jobs J_i, $i = 1, 2, \ldots, M$ which are executed in sequential order as shown in Figure 3.4.

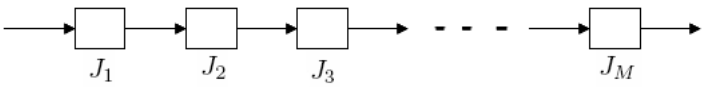

Figure 3.4 A sequential process

Let t_i be the run time of job J_i for $i = 1, 2, \ldots, M$. Hence, the run time of a

sequential program T_S is

$$T_S = \sum_{i=1}^{M} t_i.$$

To utilize the functional decomposition technique, let us assume that the sequential processes can be executed as k groups of disjoint tasks as depicted in Figure 3.5. where $\sum_{j=1}^{k} m_j = M$, t_i^j are the run time of job J_i^j and $\tilde{t}_i^j$ are the time of

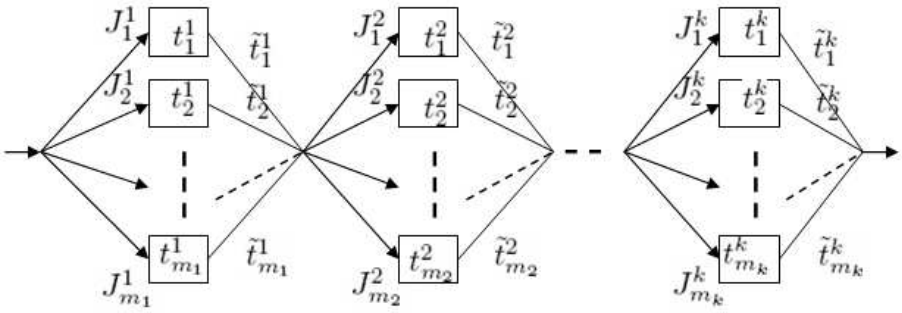

Figure 3.5 A parallel functional decomposition process

communications for $i = 1, 2, \ldots, m_j$ and $j = 1, 2, \ldots, k$. Let P be the number of processors and $P \leq \max\{m_j\}_{j=1,2,\ldots,k}$. The estimate of parallel run time T_P is

$$T_P = \sum_{j=1}^{k} \left(T^j + T^j_{comm} \right),$$

where T^j and T^j_{comm} are the execution time and the communication time for j^{th} group $j = 1, 2, \ldots, k$, respectively. So we have

$$T^j \approx \left\lceil \frac{m_j}{P} \right\rceil \max_{1 \leq i \leq m_j} t_i^j \quad \text{and} \quad T^j_{comm} \approx \sum_{i=1}^{m_j} (P-1)) \, \tilde{t}_i^j.$$

Here, $\lceil \cdot \rceil$ denotes the ceiling function. Since, $\tilde{t}_i^j = t_l + \frac{MessageSize_i^j}{BandWidth}$ where t_l is the latency time of parallel network and $MessageSize_i^j$ is the message size of job J_i^j for $j = 1, 2, \ldots, k$; $i = 1, 2, \ldots, m_j$. Therefore, a speedup is

$$S = \frac{\sum_{i=1}^{M} t_i}{\sum_{j=1}^{k} \left(\left\lceil \frac{m_j}{P} \right\rceil \max_{1 \leq i \leq m_j} t_i^j + \sum_{i=1}^{m_j} (P-1) \left(t_l + \frac{MessageSize_i^j}{BandWidth} \right) \right)}. \tag{3.14}$$

For example, let us assume that we have k groups where each group has m jobs and each job uses t seconds for the execution time and the message size of the data for all jobs are the same, $MessageSize_i^j = MessageSize$. So, the speedup is

$$S \approx \frac{m \times t}{\left(\left\lceil \frac{m}{P} \right\rceil \times t + m \times (P-1)\left(t_l + \frac{MessageSize}{BandWidth}\right)\right)}.$$

If the term of communication time is zero, then the speedup will be about P times of the sequential algorithm which corresponds to the ideal speedup.

The latency time, bandwidth, and message size of data are important for speedup as well as the run time of each jobs. Let us derive some relations between t_i, number of processors, message size, and bandwidth when speedup can be greater than 1 as follows

$$\begin{aligned} S &= \frac{\sum_{i=1}^{M} t_i}{\sum_{j=1}^{k}\left(\left\lceil \frac{m_j}{P} \right\rceil \max_{1\le i\le m_j} t_i^j + \sum_{i=1}^{m_j}(P-1)\left(t_l + \frac{MessageSize_i^j}{BandWidth}\right)\right)} \\ &< \frac{\sum_{i=1}^{k} \hat{t}^j m_j}{\sum_{j=1}^{k}\left\lceil \frac{m_j}{P} \right\rceil \hat{t}^j + \sum_{j=1}^{k}\sum_{i=1}^{m_j}(P-1)\left(t_l + \frac{MessageSize^j}{BandWidth}\right)} \end{aligned} \tag{3.15}$$

where $\hat{t}^j$ is the maximum of run time and $MessageSize^j$ is the minimum of data size on stage j^{th}. If the right-hand side of (3.14) is less than 1, then it is clear that there can be no speedup. Hence, we consider the case of the right-hand side of (3.15) greater than 1, in which we have the following relations

$$\sum_{j=1}^{k}\left(\left\lceil \frac{m_j}{P} \right\rceil \hat{t}^j\right) + \sum_{j=1}^{k}\sum_{i=1}^{m_j}(P-1)\left(t_l + \frac{MessageSize^j}{BandWidth}\right) < \sum_{i=1}^{k} \hat{t}^j m_j$$

or

$$\sum_{j=1}^{k}\left\lceil \frac{m_j}{P} \right\rceil \hat{t}^j - \sum_{j=1}^{k} \hat{t}^j m_j + \sum_{j=1}^{k} m_j (P-1)\left(t_l + \frac{MessageSize^j}{BandWidth}\right) < 0. \tag{3.16}$$

Thus, we separate inequality (3.16) into 2 cases which are possible to gain the speedup as follows.

Case 1: $\sum_{j=1}^{k} \left\{ \left\lceil \frac{m_j}{P} \right\rceil \hat{t}^j - \hat{t}^j m_j \right\} < 0$.

It is clear that $\left\lceil \frac{m_j}{P} \right\rceil < m_j$ for all $j = 1, \ldots, k$. So, it is the trivial case.

Case 2: $\sum_{j=1}^{k} \left\{ -\hat{t}^j m_j + m_j(P-1)\left(t_l + \frac{MessageSize^j}{BandWidth}\right) \right\} < 0$.

If $\hat{t}^j > (P-1)\left(t_l + \frac{MessageSize^j}{BandWidth}\right)$ for all $j = 1, \ldots, k$, it possible to get speedup greater than 1.

Next, we consider the organization of computational processes at the l^{th} stage, we have m_l jobs which can be computed independently. So, we use the `PARA_RANGE` subroutine to dividing jobs to each processor as follows:

```
SUBROUTINE
PARA_RANGE(range_start,range_end,nprocs,myid,my_start,my_end)
   iwork1 = (rang_end-range_start+1)/nprocs
   iwork2 = MOD(rang_end-range_start+1,nprocs)
   my_start = myid*iwork1+range_start+MIN(myid,iwork2)
   my_end =  my_start+iwork1-1
   IF (iwork2>myid) my_end=my_end+1
 END SUBROUTINE
```

when `range_start` and `range_end` are the first point and the last point of range; `nprocs` is the number of processors; `myid` is the identity of each processor; `my_start` and `my_end` are the start and end point of range for the identity processor.

Then, we can define the number of jobs for each processor by the following algorithm:

```
CALL PARA_RANGE(1,NJOBs,nprocs,myid,NSTART,NSTOP)
DO i=NSTART,NSTOP
   CPUID(i)=myid
ENDDO
!NJ is number of jobs for CPU(myid)
NJ(myid)=NSTOP-NSTART+1
DO i=0,myrank-1
   CALL MPI_BCAST(NJ(i),1,MPI_INTEGER,i,MPI_COMM_WORLD,ierr)
ENDDO
k=0
DO i=0,myrank-1
   DO j=1,NJ(i)
      k=k+1
      CPUID(k)=i
   ENDDO
ENDDO
```

where `NJOBs` is the number of jobs at this stage; `CPUID` is an array of integers which identity the jobs to processors; `NSTART` and `NSTOP` are the range of jobs for each processor; `myrank` is the number of processors.

Therefore, at this stage, each processor can compute its jobs simultaneously as demonstrated in Figure 3.6.

```
DO WORK=NSTART,NSTOP
   IF(WORK==1)THEN
      CALL subroutine job1
   ELSEIF(WORK==2)THEN
      CALL subroutine job2
```

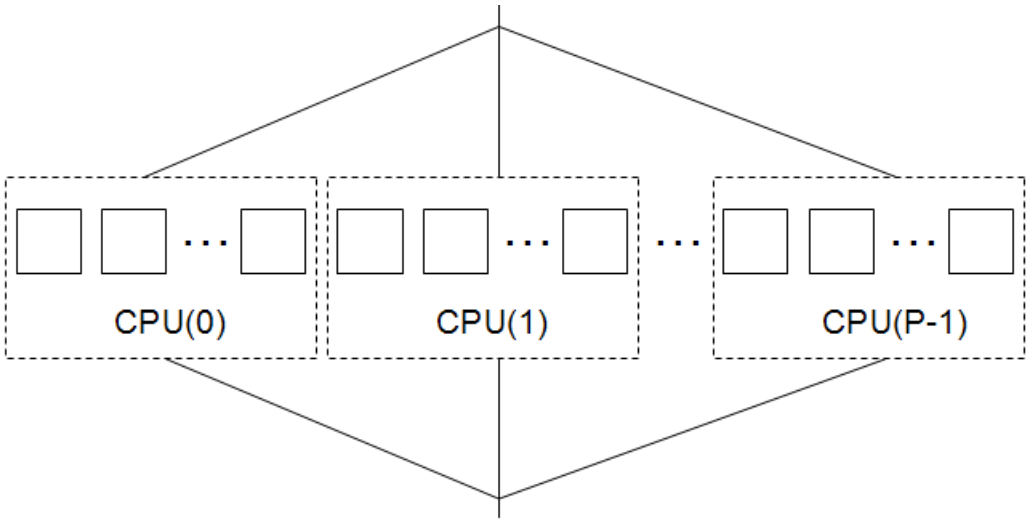

Figure 3.6 Sketch of parallel functional decomposition at the l^{th} stage

```
   ELSEIF(WORK==3)THEN
      .
      .
      .
   ELSEIF(WORK==m)THEN
      CALL subroutine job(m_l)
   ENDIF
ENDDO
```

After that we need to update the data between processors before going to the next stage by a broadcasting process as follows:

```
CALL MPI_BCAST(v_1,s_1,MPI_Type,CPUID(1),MPI_COMM_WORLD,ierr)
CALL MPI_BCAST(v_2,s_2,MPI_Type,CPUID(2),MPI_COMM_WORLD,ierr)
            .
            .
            .
```

```
CALL MPI_BCAST(v_l,s_l,MPI_Type,CPUID(P),MPI_COMM_WORLD,ierr)
```

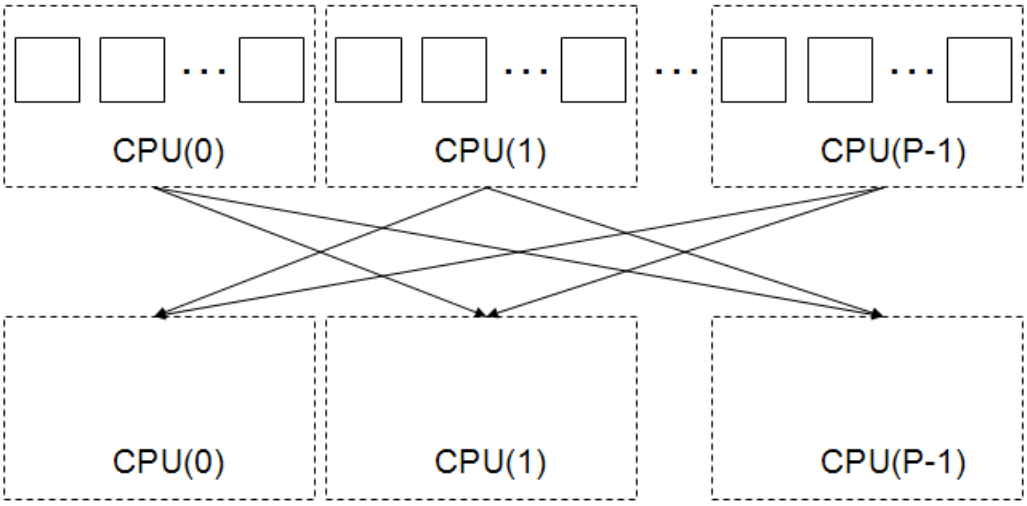

Figure 3.7 Data updating in parallel functional decomposition algorithm

Functional decomposition algorithm:

FA1. Initialization step for creating groups of communication for MPI.

FA2. The initial and boundary processes.

FA3. Divide jobs for each processor.

FA4. Compute U_d, V^* and W^* simultaneously.

FA5. Update data for U_d, V^*, and W^*.

FA6. Compute $\langle p_1 \rangle^{n+1}$ by all processors.

FA7. Compute V^{n+1} and W^{n+1} on two processors simultaneously.

FA8. Update data for V^{n+1} and W^{n+1}.

FA9. Compute $\langle \rho_1 \rangle^{n+1}$, ε^{n+1}, $\langle v'w' \rangle^{n+1}$, $\langle u'^2 \rangle^{n+1}$, $\langle v'^2 \rangle^{n+1}$, $\langle w'^2 \rangle^{n+1}$, and Θ^{n+1} simultaneously.

FA10. Update data for $\langle \rho_1 \rangle^{n+1}$, ε^{n+1}, $\langle v'w' \rangle^{n+1}$, $\langle u'^2 \rangle^{n+1}$, $\langle v'^2 \rangle^{n+1}$, $\langle w'^2 \rangle^{n+1}$, and Θ^{n+1}.

FA11. Compute e^{n+1}.

FA12. Check conditions for the exit loop, if not goto step FA4.

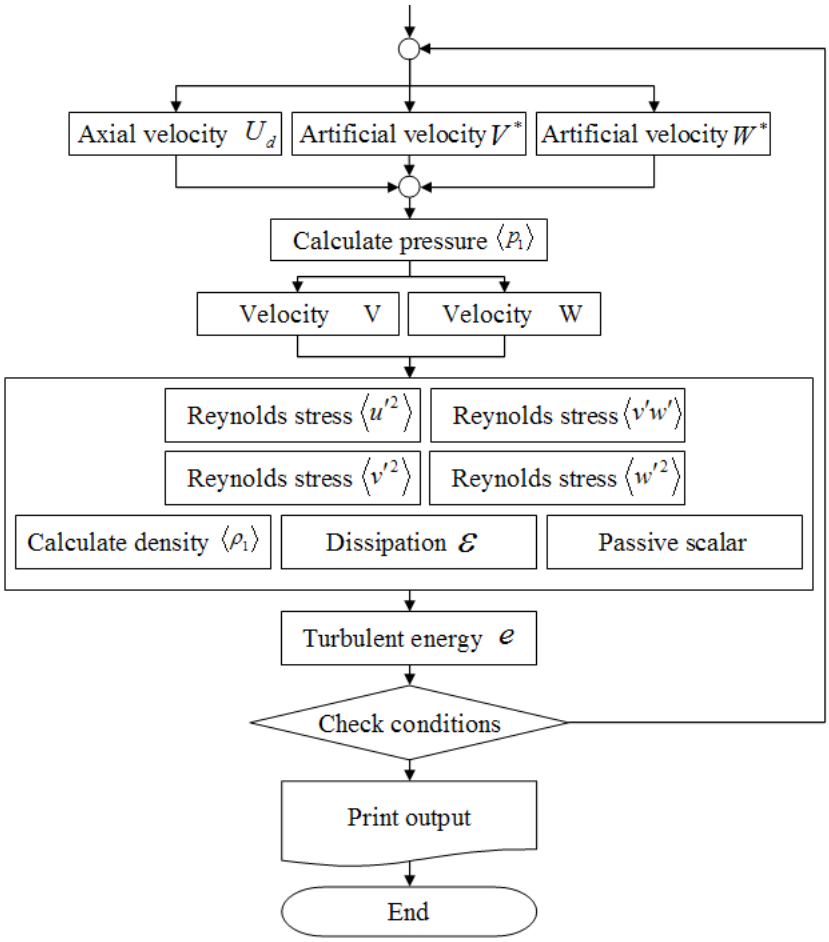

Figure 3.8 Flowchart of parallel functional decomposition algorithm

Figure 3.8 shows the sketch of the flowchart of the parallel functional decomposition algorithm. Figures 3.9 and 3.10 show the flowchart of parallel functional decomposition algorithm with the number of processors $P = 2$ and $P = 4$, respectively.

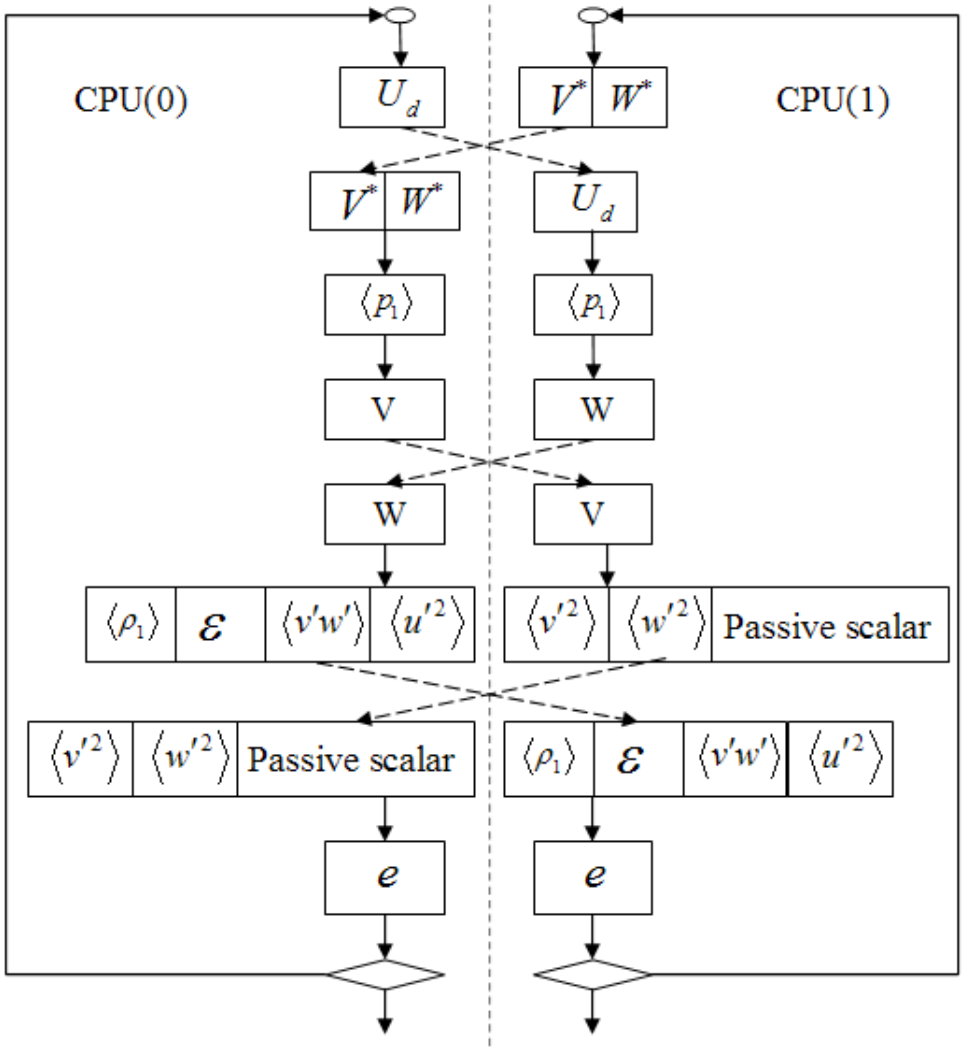

Figure 3.9 Flowchart of functional decomposition with $P = 2$

3.2.2 Parallel Domain Decomposition

In numerical partial differential equations, domain decomposition methods solve a boundary value problem by splitting the problem into smaller boundary value problems on the subdomains and iterate to coordinate the solutions between the subdomains. The problems on the subdomains are independent, which makes the domain decomposition methods suitable for parallel computing. However, in general, " *Domain decomposition*" refers to any method that divides the original problem domain into subdomains and solves locally on each subdomain. We have

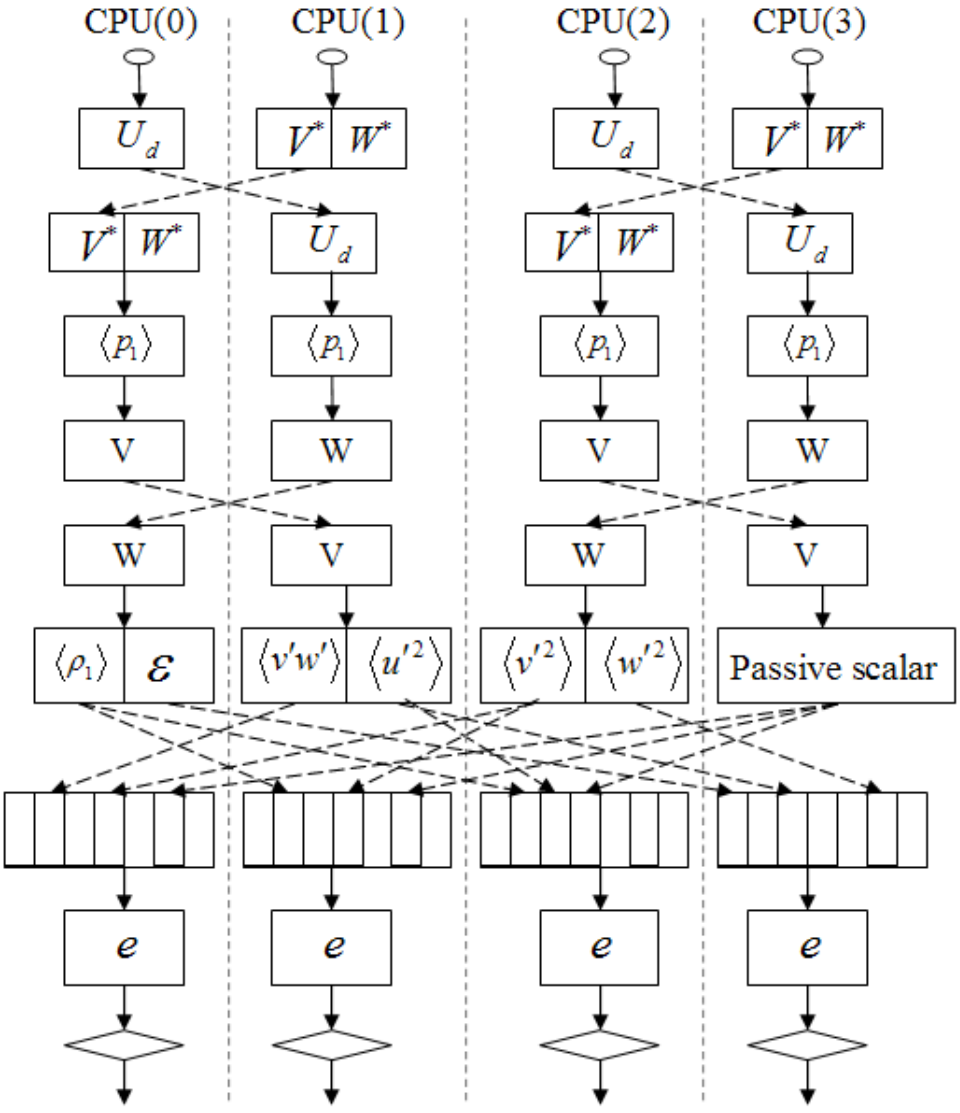

Figure 3.10 Flowchart of the functional decomposition with $P = 4$

developed parallel MPI codes to deal with general advection-diffusions partial differential equations, which are solved by using the splitting method (Yanenko (1971)). The splitting methods help reduce a multidimensional problem to series of one dimensional problems. A solution of each one dimensional problem requires an inversion of a tridiagonal matrix. It is difficult to beat sparse direct factorization methods for solving the linear systems. Taking into account the special structure of the transport equation and the particular form of splitting method, we have developed a parallel algorithm with a reasonable speedup.

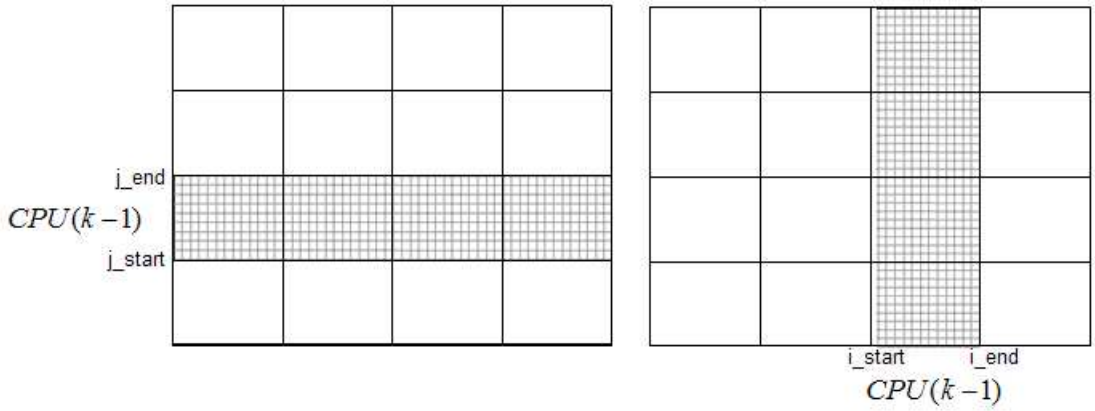

Figure 3.11 The range of strip on $CPU(K-1)$

In the scheme of the fractional steps, the transition from one time stage of the calculations to the next is divided into a series of intermediate steps, and it requires to satisfy the conditions of consistency and stability of the original problem at each stage. As a result, this method allows a choice of parameters which make it possible to construct the economical and exact schemes. The techniques of construction economical finite difference schemes are based on the fractional steps method developed by Yanenko (1971).

The key idea of splitting method is reduction of a multidimensional problem to a sequence of one dimensional problems. Only the variables of one space

direction are involved in computation at each fractional step. As a result, to find solution at the following time stage we have to solve many banded linear systems of equations (as a rule they are tridiagonal). It is evident that we can parallelize over the number of systems. Each processor gets roughly M/P systems if P is the number of processors and M is the number of systems.

Numerical models of turbulent wake dynamics in a stratified fluid involve solution of several transport equations which are solved by the implicit splitting scheme over spatial variables, Yanenko (1971). Let us consider as an example, a 2D problem in a rectangular domain Ω for a typical transport equation (this can be differential equation for transport turbulent energy, e, dissipation, ε, and so on).

$$\frac{\partial U}{\partial t} + V\frac{\partial U}{\partial y} + W\frac{\partial U}{\partial z} = \frac{\partial}{\partial y}\left(K_y\frac{\partial U}{\partial y}\right) + \frac{\partial}{\partial z}\left(K_z\frac{\partial U}{\partial z}\right) + F, \text{in } \ \Omega \times [0,T], \quad (3.17)$$

$$\begin{gathered} U(0,y,z) = U^0(0,y,z), \quad (y,z) \in \Omega, \\ U(t,y,z) = g(t,y,z), \quad (y,z) \in \partial\Omega \times [0,T], \end{gathered} \quad (3.18)$$

For the sake of simplicity, assume uniform tensor-product grid in the domain Ω

$$\Omega_h = \{(y_i, z_j) | y_i = i\triangle y, \ z_j = j\triangle z,$$

$$i = 0, \ldots, M+1; \ \ j = 0, \ldots, N+1; \ \triangle y = 1/M; \ \ \triangle z = 1/N; \}$$

The splitting scheme is written in two half time-steps as follows. During the first half step the following discretization is used

$$\begin{gathered} \frac{U_{i,j}^{n+1/2} - U_{i,j}^{n}}{\triangle t} + V_{i,j}\frac{U_{i+1,j}^{n+1/2} - U_{i-1,j}^{n+1/2}}{2\triangle y} = \\ \frac{(K_y)_{i+1/2,j}(U_{i+1,j}^{n+1/2} - U_{i,j}^{n+1/2}) - (K_y)_{i-1/2,j}(U_{i,j}^{n+1/2} - U_{i-1,j}^{n+1/2})}{\triangle y^2} + F_{i,j} \end{gathered} \quad (3.19)$$

and during the second half step

$$\frac{U_{i,j}^{n+1} - U_{i,j}^{n+1/2}}{\triangle t} + W_{i,j}\frac{U_{i,j+1}^{n+1} - U_{i,j-1}^{n+1}}{2\triangle z} = \frac{(K_z)_{i,j+1/2}(U_{i,j+1}^{n+1} - U_{i,j}^{n+1}) - (K_z)_{i,j-1/2}(U_{i,j}^{n+1} - U_{i,j-1}^{n+1})}{\triangle z^2}. \tag{3.20}$$

Equation (3.19) can be written in the tridiagonal format

$$\begin{aligned} U_{0,j}^{n+1/2} &= g(0, z_j) \\ -a_{ij}U_{i-1,j}^{n+1/2} + c_{ij}U_{i,j}^{n+1/2} - b_{ij}U_{i+1,j}^{n+1/2} &= f_{i,j}, \quad i = 1, 2, \ldots, M; j = 1, 2, \ldots, N, \\ U_{M+1,j}^{n+1/2} &= g(y_M, z_j) \end{aligned} \tag{3.21}$$

Therefore, at each fixed j one has a linear system of M equations with a tridiagonal matrix. So, we can simply parallelize the computation in (3.21) by assigning N/P systems of three-point equations to each processor. For processor K, namely $CPU(K-1)$, it will solve the system of equations for $j = \frac{(K-1)N}{P} + 1, \ldots, \frac{KN}{P}$. With this parallelization strategy, the coefficient matrix needs to be distributed row-wise as demonstrated in Figure 3.12.

We use the subroutine `PARA_RANGE` to find the range of row-strip and column-strip on `CPU(K-1)` as shown in Figure 3.11 while `j_start`, `j_end`, `i_start` and `i_end` can be determined by

```
CALL PARA_RANGE(1,M,nprocs,myid,j_start,j_end)
```

and

```
CALL PARA_RANGE(1,N,nprocs,myid,i_start,i_end).
```

After that, we can define the blocks of data updating for each processor as follows

```
SUBROUTINE block(imin,imax,jmin,ista,iend,jend,iotype,inewtype)
INCLUDE 'mpif.h'
```

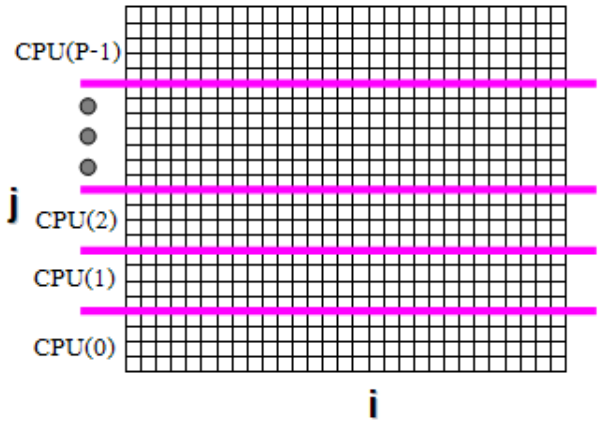

Figure 3.12 The sketch of domain decomposition process in direction of $Y-$axis

```
  INTEGER imin,imax,ista,iend,jsta,jend
  INTEGER iblock(2),idisp(2),itype(2)
  CALL MPI_TYPE_EXTENT(ioldtype,isize,ierr)
  ilen = iend - ista + 1 jlen = jend - jsta + 1
  CALL MPI_TYPE_VECTOR(jlen,ilen,imax-imin+1,ioldtype,itemp,ierr)
  iblock(1) = 1
  iblock(2) = 1
  idisp(1)=0
  idisp(2)=((imax-imin+1)*(jsta-jmin)+(ista-imin))*isize
  itype(1) = MPI_LB
  itype(2) = itemp
  !Construct the sub-block
  CALL MPI_TYPE_STRUCT(2,iblock,idisp,itype,inewtype,ierr) CALL
  MPI_TYPE_COMMIT(inewtype,ierr)
  return
END
```

If each processor computes all its grid points, then we need to update the data blocks before we go to the next computation by the following algorithm

```
!sending and receiving updated data
DO ir = 0,nprocs-1
 IF (ir /= myid) THEN
   CALL MPI_ISEND(U,1,itype2(ir,myid),ir,1,MPI_COMM_WORLD,ireq1(ir),
   ierr)
   CALL MPI_IRECV(U,1,itype2(myid,ir),ir,1,MPI_COMM_WORLD,ireq2(ir),
   ierr)
 ENDIF
ENDDO

!check for status of sending and receiving data
DO ir =0,nprocs-1
 IF (ir/=myid) THEN
   CALL MPI_WAIT(ireq1(ir),status,ierr)
   CALL MPI_WAIT(ireq2(ir),status,ierr)
 ENDIF
ENDDO
```

Equations (3.20) of the second half step can be written in a tridiagonal format

$$\begin{aligned} U_{i,0}^{n+1} &= g(y_i, 0) \\ -a_{i,j}U_{i,j-1}^{n+1} + c_{i,j}U_{i,j}^{n+1} - b_{i,j}U_{i,j+1}^{n+1} &= f_{i,j}, \;\; j = 1, 2, \ldots, N; i = 1, 2, \ldots, M, \\ U_{i,N+1}^{n+1} &= g(y_i, z_N) \end{aligned} \tag{3.22}$$

Therefore, at each fixed i one has a linear system of N equations with tridiagonal

matrix. So, on $CPU(K-1)$, it will solve a system of equations for $i = \frac{(K-1)M}{P} + 1, \ldots, \frac{KM}{P}$. Figure 3.13 shows the direction of computation of column-wise for the second step of the splitting scheme.

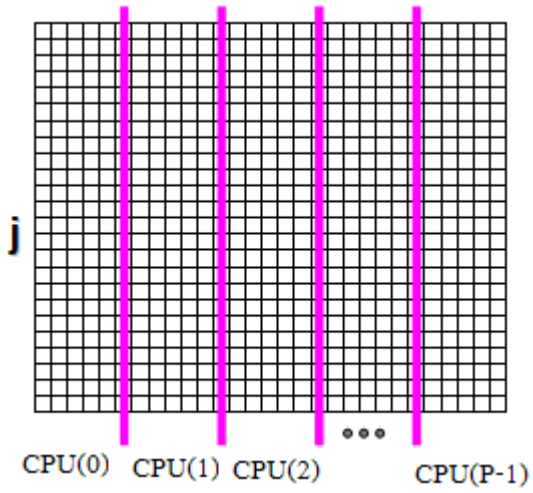

Figure 3.13 The sketch of domain decomposition process in direction of $X-$axis

We can compute in parallel on every strip in the direction of the $X-$axis for the first step and on every strip in the direction of the $Y-$axis for the second step. Consequently, we need to redistribute the updated data between the row-splitting and column-splitting or vice versa. The scheme of data updating is depicted in Figure 3.14

Let us consider a grid of size $M \times N$ and assume that one grid node requires t seconds for computation. So, for all nodes of the grid require $(M \times N) \times t$ seconds. Let P be the number of processors and each processor works with $(M \times N)/P$ nodes of grid. Hence, each processor requires $(M \times N \times t)/P$ seconds and $[P \times (P-1)]$ times for updating data blocks. Therefore, the speedup is

$$S = \frac{N \times M \times t}{\frac{N \times M \times t}{P} + P \times (P-1) \times \left(t_l + \frac{MessageSize}{BandWidth}\right)}. \tag{3.23}$$

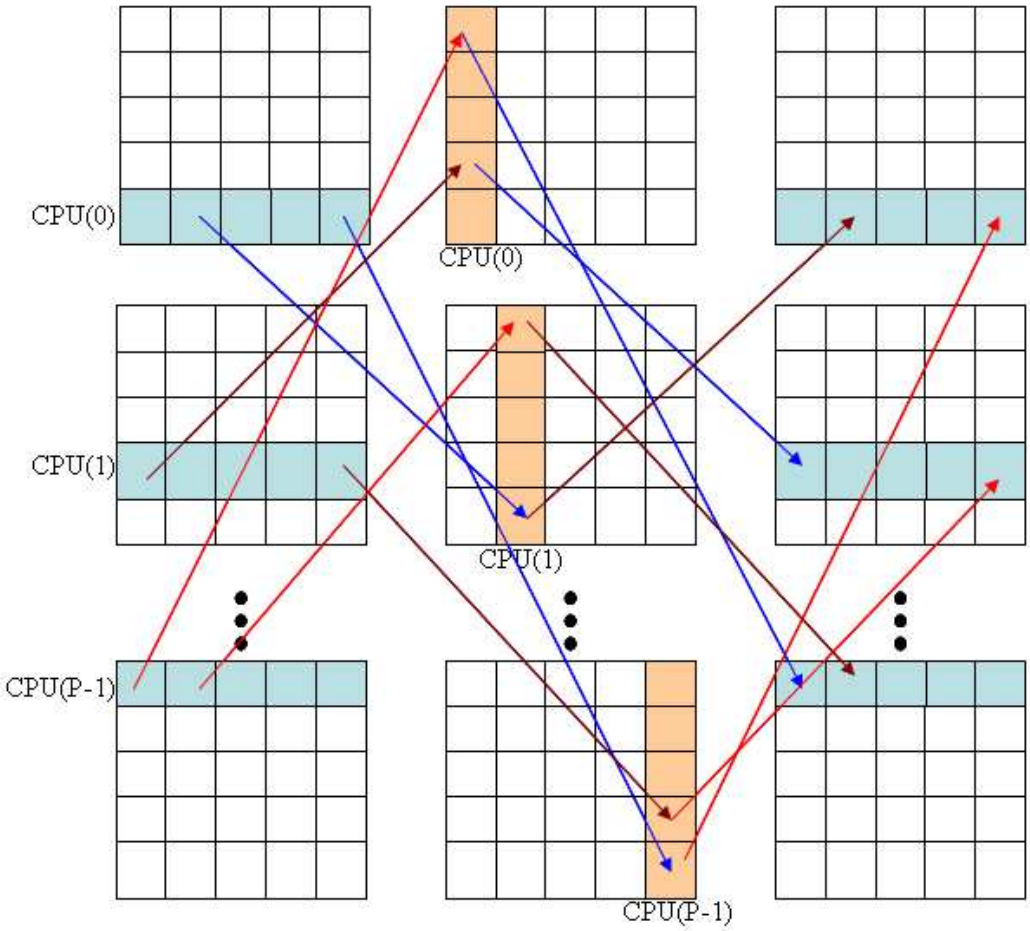

Figure 3.14 Updating data blocks on each processor

To consider the conditions of speedup, let us assume that T is the sequential run time, that is $T = N \times M \times t$. To get the speedup, we must have

$$\frac{T}{\frac{T}{P} + P \times (P-1) \times \left(t_l + \frac{MessageSize}{BandWidth}\right)} > 1 \tag{3.24}$$

or

$$\begin{aligned} T &> \tfrac{T}{P} + P(P-1)\left(t_l + \tfrac{MessageSize}{BandWidth}\right) \\ T\left(\frac{P-1}{P}\right) &> P(P-1)\left(t_l + \tfrac{MessageSize}{BandWidth}\right) \\ T &> P^2\left(t_l + \tfrac{MessageSize}{BandWidth}\right). \end{aligned} \tag{3.25}$$

Hence, to get the speedup, we must have

$$T > P^2 \left(t_l + \frac{MessageSize}{BandWidth} \right).$$

There is no speedup if

$$T \leq P^2 \left(t_l + \frac{MessageSize}{BandWidth} \right).$$

It means that we may have used too many processors or the message size of problem and the latency time of the parallel system is too high and the bandwidth is too small. So, the sequential time will be less than or equals to the parallel time in which case we will have a bad speedup.

In the next chapter, we show the validation of our parallel algorithms, the speedup, and the numerical results of the parallel program on two different clusters.

CHAPTER IV

VALIDATION AND NUMERICAL RESULTS

4.1 Comparison of Parallel and Sequential Algorithms

For the far turbulent wake behind a towed body in a linear stratified medium, a hierarchy of semi-empirical turbulent models was described in Chernykh *et al.* (1999, 2006, 2008) and Voropayeva (2000, 2002), etc. The most complex model comprises differential equations for transport of normal Reynolds stresses. In total, this model consists of 12 differential equations. These equations are solved by using the splitting techniques of Yanenko (1971). In the sequential algorithm, this model requires 14 jobs, which are executed sequentially. We chose this model to implement the ideas of the functional and domain decomposition developed in Chapter III.

The reliability of the five models is considered in the work of Chernykh (1999, 2006), Moshkin *et al.* (2001). It demonstrated that all models except Model 2 give reasonable results which correspond to the numerical and experimental data of Lin and Pao (1973, 1974, 1979). The numerical results using the parallel codes were compared with the results using the sequential code to guarantee the correctness of the parallel algorithms. Therefore, the performance of parallel algorithms is discussed in this chapter.

First, we utilize the idea of the functional decomposition. In the second approach, we use the idea of the domain decomposition to each of the 14 jobs solved in a sequential sequence. It means that each job with the splitting scheme is realized by parallel domain decomposition algorithm with all available processors.

The results of parallel functional decomposition algorithm and parallel domain decomposition algorithm are compared with Lin and Pao's experimental data and Hassid's computational results (Lin and Pao (1973, 1974, 1979), Hassid (1980)) and obtained by the execution of the sequential algorithm.

The comparison of results for the density Froude number $F_d = 31$ is presented in Figures 4.1-4.4. Figures 4.1 and 4.2 show the time variation of the dimensionless axial velocity defect $U_d = U_d(x, 0, 0)$ in the case of drag and momemtumless wakes, respectively. The symbols $\circ$ and $\bullet$ correspond to Lin ans Pao's experimental data. The dashed lines correspond to the results of Hassid's numerical experiments (Hassid (1980)). The symbols $\triangledown$ and $\blacktriangledown$ indicate the computational results by the sequential algorithm. The symbols $\triangleright$ and $\blacktriangleright$ indicate the computational results by the domain decomposition algorithm. The symbols $\square$ and $\blacksquare$ indicate the computational results by the functional decomposition algorithm. Analogous data for the dimensionless axial values of turbulent energy $e_0 = e_0(x) = e(x, 0, 0)$ are presented in Figures 4.3-4.4.

Tables 4.1-4.4 show the computational results obtained from the sequential, functional decomposition and domain decomposition algorithms for the density Froude number $F_d = 280$. Tables 4.1 and 4.2 compare the non-dimension values of axial turbulent energy for different distances. Table 4.1 corresponds to momentumless wake and Table 4.2 corresponds to drag wake. Tables 4.3 and 4.4 show the comparison of the values of non-dimensional axial velocity defect for the momentumless and drag wakes respectively. Tables 4.5 and 4.6 show the decay of dissipation ε in the momentumless and drag wakes, respectively.

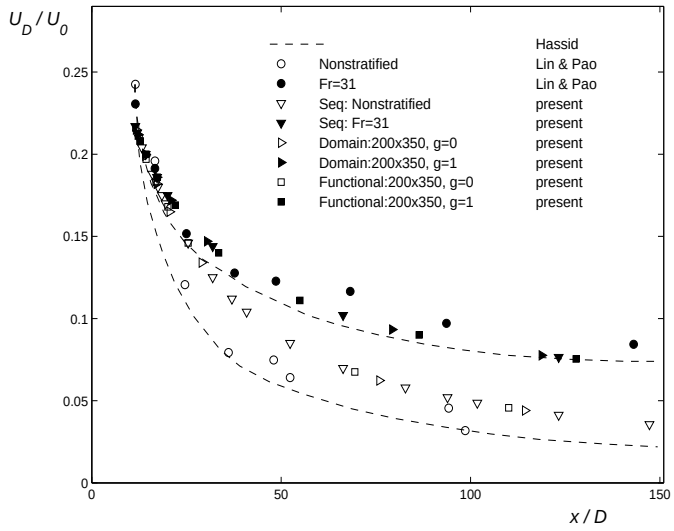

Figure 4.1 Comparison of the axial values of the longitudinal velocity component defect, $U_d(x)$ in the wake behind a towed body calculated by the sequential algorithm and parallel algorithms with Lin and Pao's experimental data and Hassid's computational results

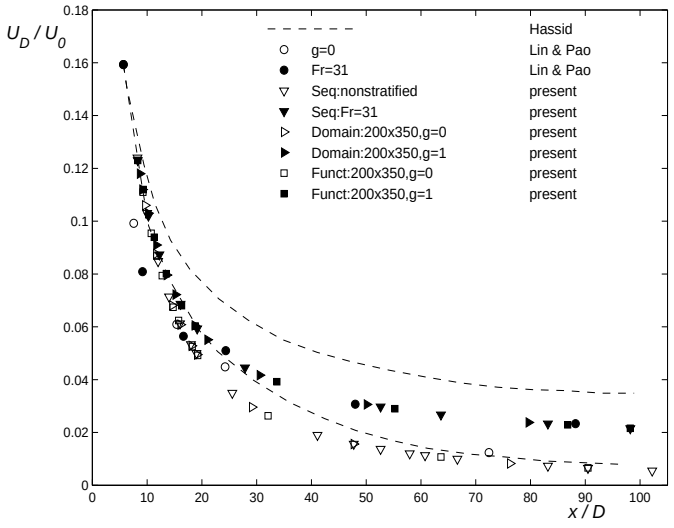

Figure 4.2 Comparison of the axial values of the longitudinal velocity component defect, $U_d(x)$ in the wake behind a self-propelled body calculated by the sequential algorithm and parallel algorithms with Lin and Pao's experimental data and Hassid's computational results

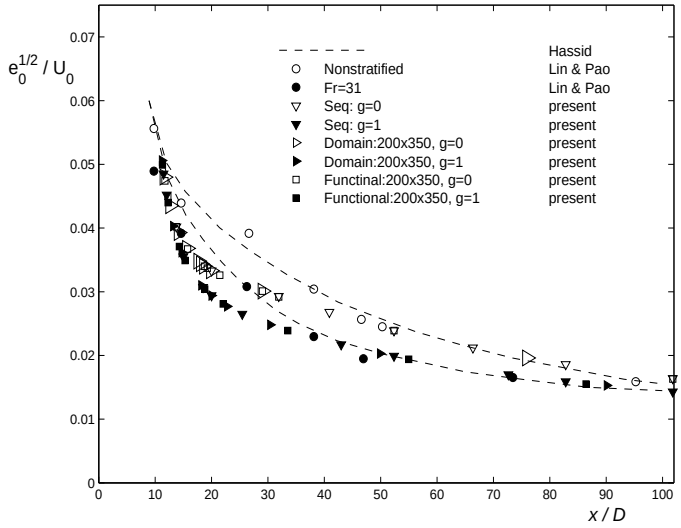

Figure 4.3 Comparison of the axial values of the turbulent energy, $e_0(x)$ in the wake behind a towed body calculated by the sequential algorithm and parallel algorithms with Lin and Pao's experimental data and Hassid's computational results

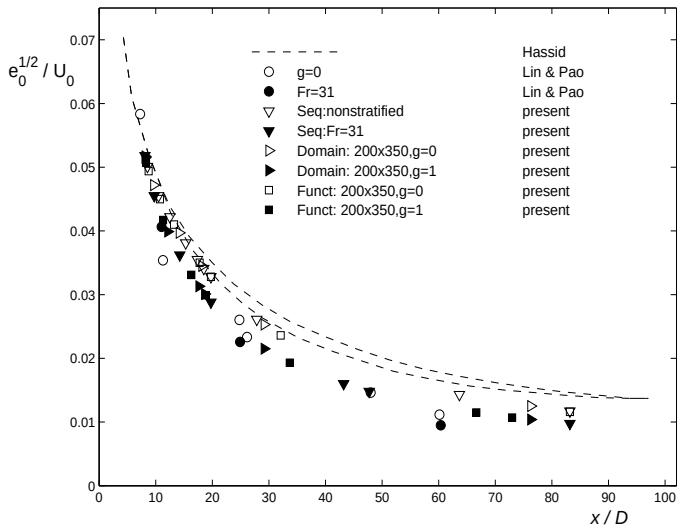

Figure 4.4 Comparison of the axial values of the turbulent energy, $e_0(x)$ in the wake behind a self-propelled body calculated by the sequential algorithm and parallel algorithms with Lin and Pao's experimental data and Hassid's computational results

x/D	e (sequential alg.)	e (functional decomp.)	e (domain decomp.)
12.00	0.42813E-01	0.42890E-01	0.42813E-01
19.00	0.33442E-01	0.33436E-01	0.33442E-01
63.65	0.13778E-01	0.13744E-01	0.13778E-01
119.22	0.80534E-02	0.80220E-02	0.80534E-02
252.74	0.41380E-02	0.41117E-02	0.41380E-02
947.74	0.13891E-02	0.13771E-02	0.13891E-02
1502.74	0.96091E-03	0.95273E-03	0.96091E-03

Table 4.1 Axial values of the dimensionless turbulent energy, e, depending on the distance in the momentumless wake, $F_d = 280$

x/D	e (sequential alg.)	e (functional decomp.)	e (domain decomp.)
12.00	0.46145E-01	0.46144E-01	0.46145E-01
19.00	0.33749E-01	0.33737E-01	0.33749E-01
63.35	0.21323E-01	0.21258E-01	0.21323E-01
118.78	0.14147E-01	0.14093E-01	0.14147E-01
252.24	0.79067E-02	0.78716E-02	0.79067E-02
947.24	0.29912E-02	0.29963E-02	0.29912E-02
1502.24	0.21874E-02	0.21867E-02	0.21874E-02

Table 4.2 Axial values of the dimensionless turbulent energy, e, depending on the distance in the drag wake, $F_d = 280$

x/D	U_d (sequential alg.)	U_d (functional decomp.)	U_d (domain decomp.)
12.00	0.84457E-01	0.84848E-01	0.84457E-01
19.00	0.50144E-01	0.50165E-01	0.50144E-01
63.65	0.11243E-01	0.11336E-01	0.11243E-01
119.22	0.54013E-02	0.54826E-02	0.54013E-02
252.74	0.27806E-02	0.28392E-02	0.27806E-02
947.74	0.14482E-02	0.14851E-02	0.14482E-02
1502.74	0.12296E-02	0.12611E-02	0.12296E-02

Table 4.3 Axial values of the dimensionless axial velocity defect, U_d, depending on the distance in the momentumless wake, $F_d = 280$

x/D	U_d (sequential alg.)	U_d (functional decomp.)	U_d (domain decomp.)
12.00	0.21287E-00	0.21288E-00	0.21287E-00
19.00	0.17274E-00	0.17276E-00	0.17274E-00
63.35	0.73696E-01	0.73956E-01	0.73696E-01
118.78	0.44832E-01	0.45083E-01	0.44832E-01
252.24	0.27355E-01	0.27544E-01	0.27355E-01
947.24	0.16515E-01	0.16626E-01	0.16515E-01
1502.24	0.14563E-01	0.14622E-01	0.14563E-01

Table 4.4 Axial values of the dimensionless axial velocity defect, U_d, depending on the distance in the drag wake, $F_d = 280$

x/D	ε (sequential alg.)	ε (functional decomp.)	ε (domain decomp.)
12.00	0.21002E-03	0.21176E-03	0.21002E-03
19.00	0.80929E-04	0.80946E-04	0.80929E-04
63.65	0.36783E-05	0.36947E-05	0.36783E-05
119.22	0.61573E-06	0.61726E-06	0.61573E-06
252.74	0.69568E-07	0.69330E-07	0.69568E-07
947.74	0.19673E-08	0.19434E-08	0.19673E-08
1502.74	0.59178E-09	0.58408E-09	0.59178E-09

Table 4.5 Axial values of the dimensionless turbulent dissipation, ε, depending on the distance in the momentumless wake, $F_d = 280$

x/D	ε (sequential alg.)	ε (functional decomp.)	ε (domain decomp.)
12.00	0.44275E-03	0.44291E-03	0.44275E-03
19.00	0.11068E-03	0.11087E-03	0.11068E-03
63.35	0.12594E-04	0.12645E-04	0.12594E-04
118.78	0.27430E-05	0.27560E-05	0.27430E-05
252.24	0.35925E-06	0.35989E-06	0.35925E-06
947.24	0.12583E-07	0.12619E-07	0.12583E-07
1502.24	0.42757E-08	0.42988E-08	0.42757E-08

Table 4.6 Axial values of the dimensionless turbulent dissipation, ε, depending on the distance in the drag wake, $F_d = 280$

It can be seen that the results obtained from the three algorithms agree well (see Figures 4.1-4.4) with Lin and Pao's experiments, Lin and Pao (1973, 1974, 1979). Tables 4.1-4.6 show the results of computations by the functional decomposition algorithm as well as the results by the domain decomposition algorithm which coincide with the results of sequential computation. The dynamics of a turbulent wake and internal wakes generated by the wake in a linearly stratified fluid are illustrated in Figures 4.5-4.6. The Froude number was assumed to be equal 280 which corresponds to the conditions of one of the laboratory experiments of Lin and Pao (1979). It is not surprising that the patterns of isolines $\langle\rho_1\rangle/(aD\rho_0 Fr_D^{1/4})$ drawing by the results from the functional and domain decomposition algorithms are the same. Figure 4.5 shows the isolines $\langle\rho_1\rangle/(aD\rho_0 Fr_D^{1/4})$ in momentumless wake for the moments of time $t/T = 1$, $t/T = 2$ and $t/T = 3$, respectively. The level of isolines varies from -0.015 to 0.015 with a step of 0.005. Analogous results for the drag wake are presented in Figure 4.6.

4.2 Results of Experimental Performance of Parallel Algorithms

For numerical experiments, the sequential and parallel algorithms are compiled and run by using two clusters of Solaris (Cluster-I) and Linux (SUT-HPCC) operating systems which are located at School of Mathematics and High Performance Computing Center of Suranaree University of Technology. Cluster-I is based on PC sever with 10 processors of Opteron (2 cores) 1.6 GHz, 8 GB of RAM, Intel (4 cores) 2.0 GHz, 8 GB of RAM, and two machines of AMD (2 cores) 2.4 GHz, 2 GB of RAM. SUT-HPCC is based on 7 machines of 2 Quad cores Xeon 2.33 GHz, 8 GB of RAM and 7 machines of 2 Dual cores Xeon 3.0 GHz, 4 GB

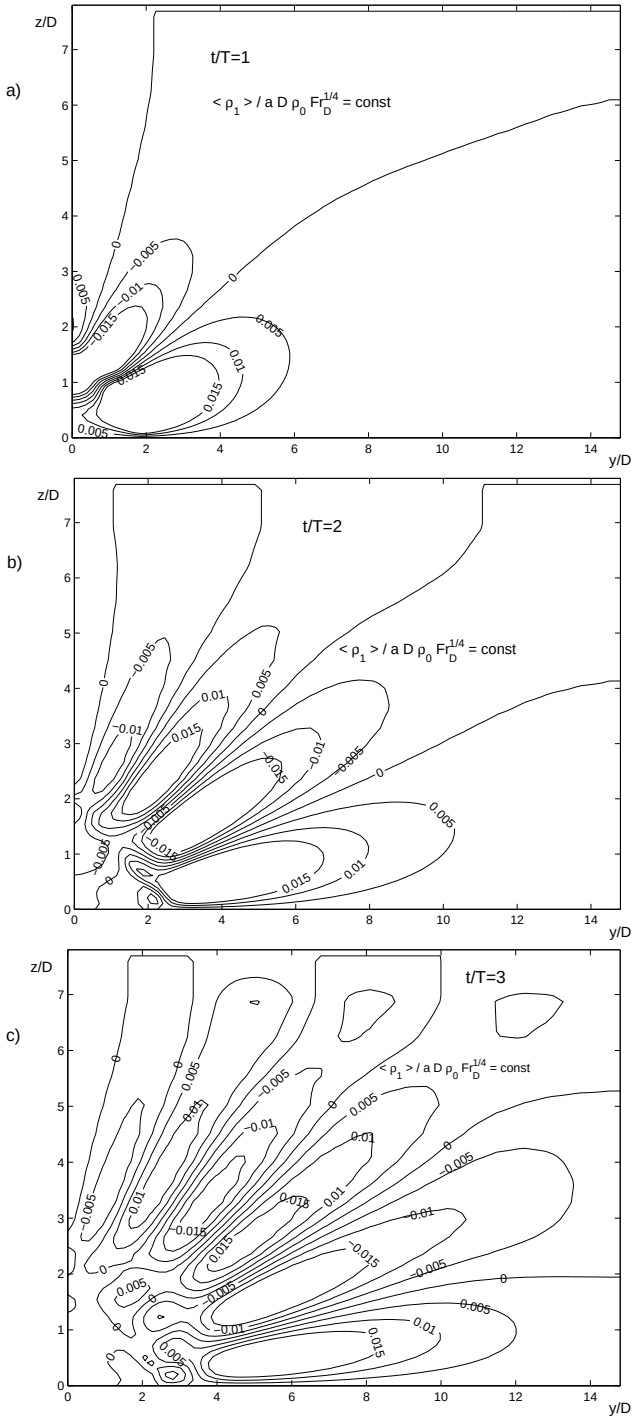

Figure 4.5 Isolines $\langle\rho_1\rangle/(aD\rho_0 Fr_D^{1/4})$ =const. Momentumless wake, $F_d = 280$. Functional and domain decomposition, (a) $t/T = 1$, (b) $t/T = 2$ and (c) $t/T = 3$

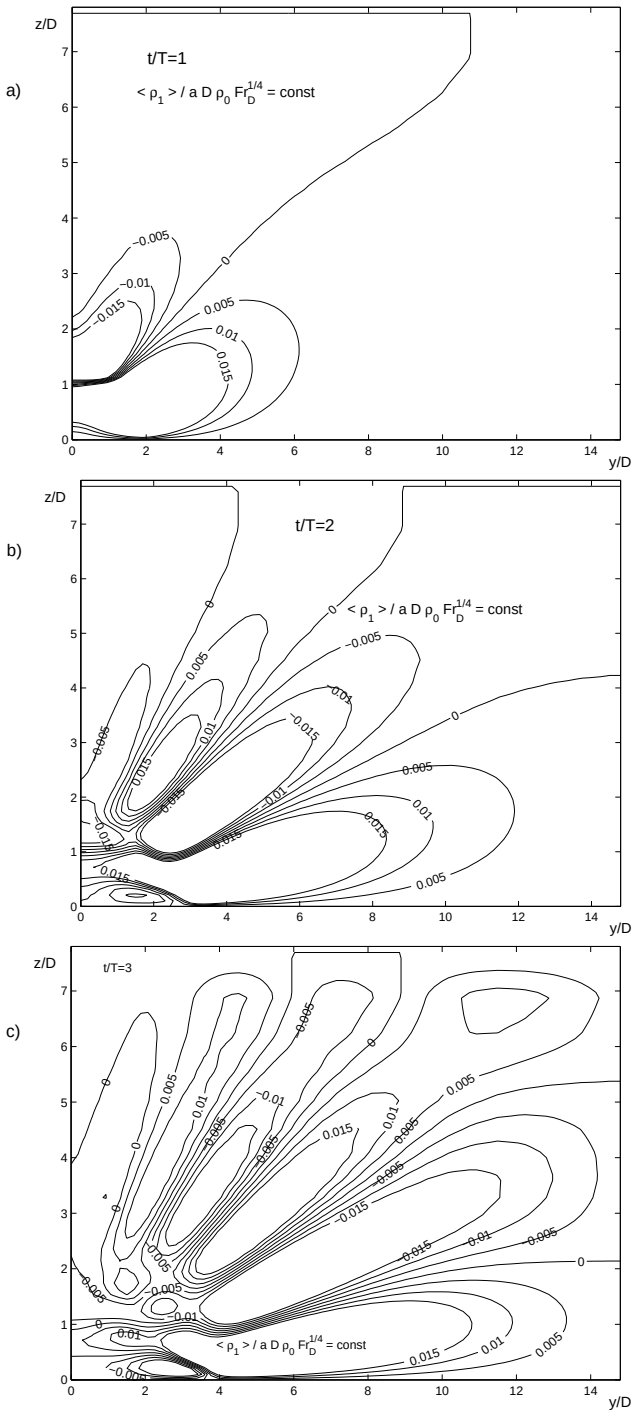

Figure 4.6 Isolines $\langle\rho_1\rangle/(aD\rho_0 Fr_D^{1/4})$ =const. Drag wake, $F_d = 280$. Functional and domain decomposition, (a) $t/T = 1$, (b) $t/T = 2$ and (c) $t/T = 3$

of RAM. On both clusters, we used the Sun FORTRAN compiler of Sun Studio 11 with MPI 2.0 library. Latency time and bandwidth are estimated by using the program of Nielsen (2003) and was $5.2E-5$ sec, 118MB/sec for Cluster-I, and $5.9E-5$ sec, 260MB/sec for SUT-HPCC.

The run time of the sequential code is used as a measure of the run time on one processor. In this study, the run time starts after generating an initial state. Wall-clock time is used to record the run time. The wall clock time (Schönauer (2000), Kumar *et al.* (1994)) is used to represent the total run time since it includes the idle time and communication time.

To perform experimental estimation of speedup and efficiency, we utilize Model 4. For parallel functional decomposition, the analysis of the numerical model shows that it can be represented in the form of 5 independent groups of tasks. In the notations of Figure 3.5, we have $m_1 = 3$, $m_2 = 1$, $m_3 = 2$, $m_4 = 7$ and $m_5 = 1$. The jobs in each group can be executed independently.

The execution times are shown in Tables 4.7 and 4.8 for Cluster-I and in Tables 4.9 and 4.10 for SUT-HPCC. The performance results for both clusters in terms of speedup characteristics are shown in Figures 4.7 and 4.8. These figures give the detailed statistics of the parallel code on 1, 2, 4, 7 and 8 processors for Cluster-I and SUT-HPCC. We estimate the speedup of parallel functional and domain decomposition on the following grids 400×750, 600×1150 and 800×1550 nodes in z and y directions, respectively. Tables 4.11-4.14 show the numerical efficiency of parallel functional and domain decompositions on Cluster-I and SUT-HPCC, respectively.

The performance results for parallel runs for both clusters depend on grid size. The speedup of parallel functional decomposition increases almost linearly up to four processors, after that, it starts to deviate away from the perfect speedup,

while the speedup of parallel domain decomposition increases linearly up to eight processors.

Grid size	400x750		600x1150		800x1550	
NCPUs	Time	Speedup	Time	Speedup	Time	Speedup
1	6824.52	-	14058.94	-	27420.92	-
2	5593.86	1.22	11247.15	1.25	19727.28	1.39
4	3554.43	1.92	6994.49	2.01	13183.13	2.08
7	3174.19	2.15	6361.51	2.21	11718.34	2.34

Table 4.7 Speedup results of the functional decomposition algorithm on Cluster-I

Grid size	400x750		600x1150		800x1550	
NCPUs	Time	Speedup	Time	Speedup	Time	Speedup
1	6824.52	-	14058.94	-	27420.92	-
2	4382.75	1.56	8824.99	1.59	16876.53	1.62
4	2293.28	2.98	4608.72	3.05	8874.08	3.09
7	1683.57	4.05	3426.01	4.10	6368.64	4.31
8	1564.32	4.36	3171.43	4.43	6039.14	4.54

Table 4.8 Speedup results of the domain decomposition algorithm on Cluster-I

Grid size	400x750		600x1150		800x1550	
NCPUs	Time	Speedup	Time	Speedup	Time	Speedup
1	4913.07	-	11952.72	-	25420.44	-
2	3584.59	1.37	8417.48	1.42	17614.25	1.44
4	2516.09	1.95	5683.31	2.10	12058.83	2.11
7	2134.25	2.30	4927.48	2.43	10291.68	2.47

Table 4.9 Speedup results of the functional decomposition algorithm on SUT-HPCC

Grid size	400x750		600x1150		800x1550	
NCPUs	Time	Speedup	Time	Speedup	Time	Speedup
1	4913.07	-	11952.72	-	25420.44	-
2	3125.54	1.57	7431.28	1.61	14869.83	1.71
4	1628.22	3.02	3850.87	3.10	8173.77	3.11
7	1186.73	4.14	2851.70	4.19	5846.71	4.35
8	1120.38	4.39	2686.21	4.45	5487.65	4.63

Table 4.10 Speedup results of the domain decomposition algorithm on SUT-HPCC

Grid size	400x750		600x1150		800x1550	
NCPUs	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
2	1.22	0.6100	1.25	0.6250	1.39	0.6950
4	1.92	0.4800	2.01	0.5025	2.08	0.5200
7	2.15	0.3071	2.21	0.3157	2.34	0.3343

Table 4.11 Efficiency results of the functional decomposition algorithm on Cluster-I

Grid size	400x750		600x1150		800x1550	
NCPUs	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
2	1.56	0.7800	1.59	0.7950	1.62	0.8100
4	2.98	0.7450	3.05	0.7625	3.09	0.7725
7	4.05	0.5786	4.10	0.5857	4.31	0.6157
8	4.36	0.5450	4.43	0.5538	4.54	0.5675

Table 4.12 Efficiency results of the domain decomposition algorithm on Cluster-I

Grid size	400x750		600x1150		800x1550	
NCPUs	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
2	1.37	0.6850	1.42	0.7100	1.44	0.7200
4	1.95	0.4875	2.10	0.5250	2.11	0.5275
7	2.30	0.3286	2.43	0.3471	2.47	0.3529

Table 4.13 Efficiency results of the functional decomposition algorithm on SUT-HPCC

Grid size	400x750		600x1150		800x1550	
NCPUs	Speedup	Efficiency	Speedup	Efficiency	Speedup	Efficiency
2	1.57	0.7850	1.61	0.8050	1.71	0.8550
4	3.02	0.7550	3.10	0.7750	3.11	0.7775
7	4.14	0.5914	4.19	0.5986	4.35	0.6214
8	4.39	0.5488	4.45	0.5563	4.63	0.5788

Table 4.14 Efficiency results of the domain decomposition algorithm on SUT-HPCC

Figures 4.7 and 4.8 show the speedup results of parallel functional and domain decompositions on the different grid size, 400×750, 600×1150 and 800×1550 on Cluster-I and SUT-HPCC.

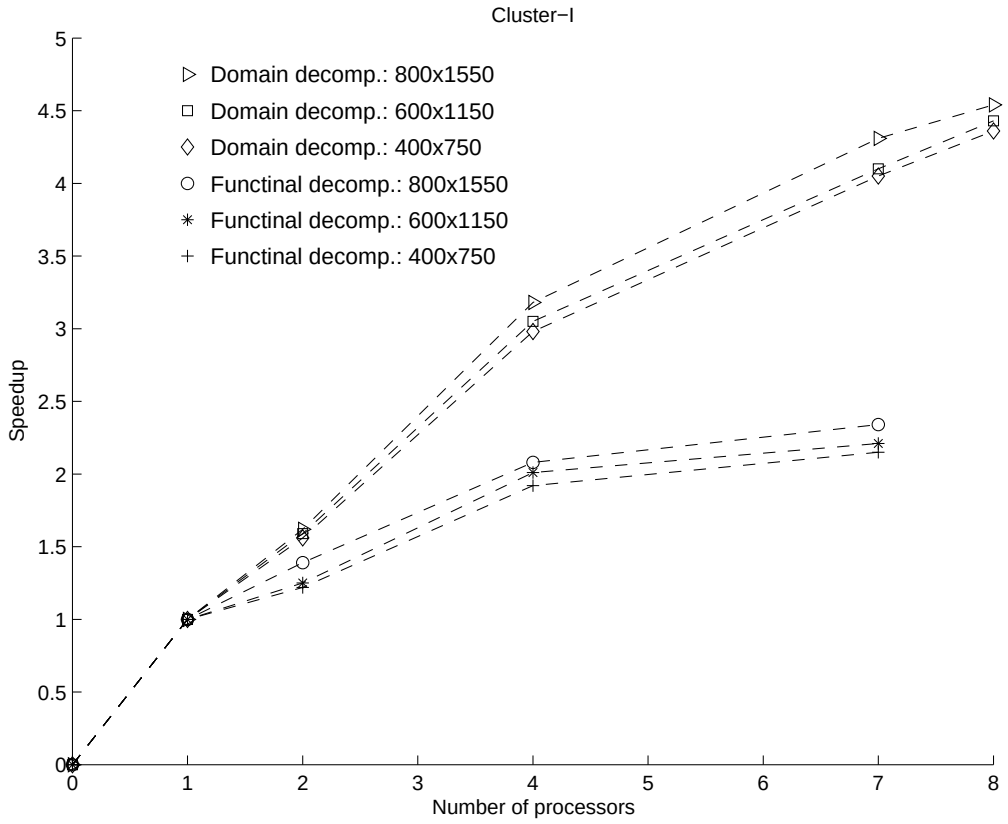

Figure 4.7 Speedup results of grid size 400×750, 600×1150 and 800×1550 on Cluster-I

Figures 4.9-4.14 show the comparison between the theoretical speedup and the numerical speedup on both clusters of parallel functional and domain decomposition algorithms, respectively. The speedups obtained by both the parallel functional and domain decomposition algorithms are in reasonably well agreement with the theoretical estimate. Theoretical speedup is higher than experimental speedup due to not perfect balance of computational and communication time of different jobs.

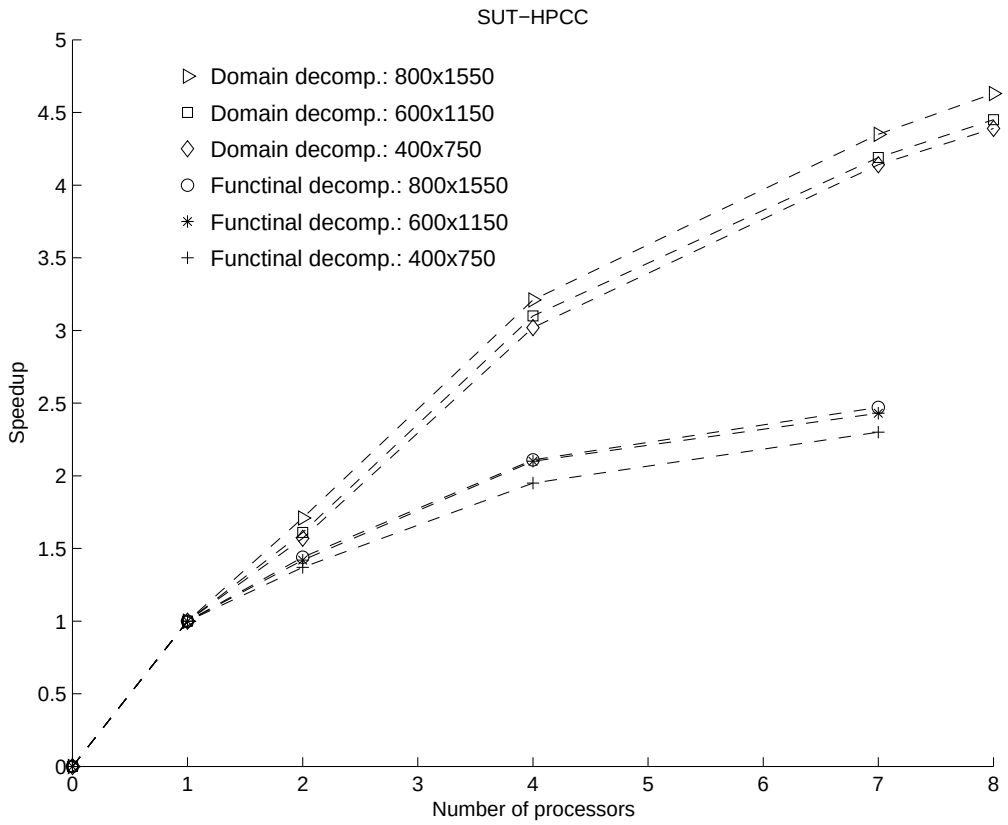

Figure 4.8 Speedup results of grid size 400×750, 600×1150 and 800×1550 on SUT-HPCC

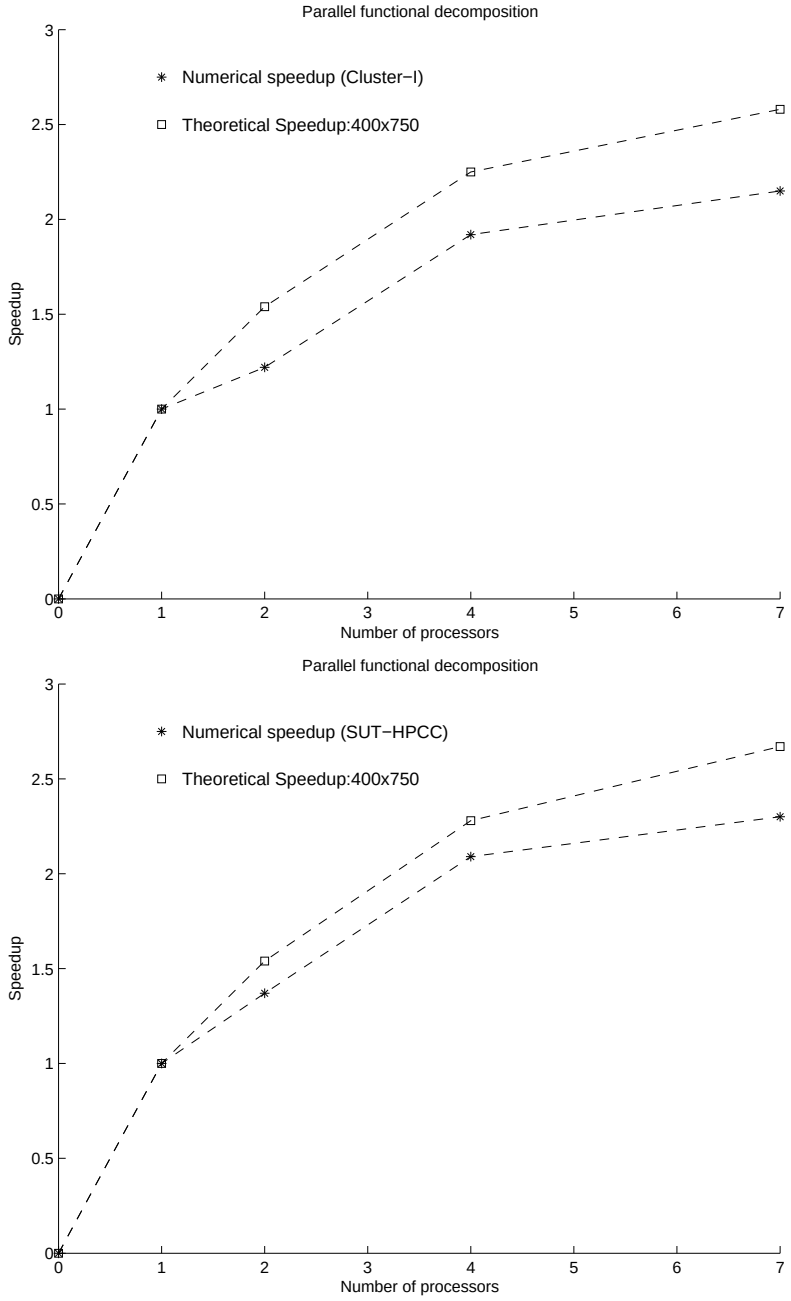

Figure 4.9 Comparison between theoretical speedup and numerical speedups of parallel functional decomposition on Cluster-I and SUT-HPCC, grid size=400×750

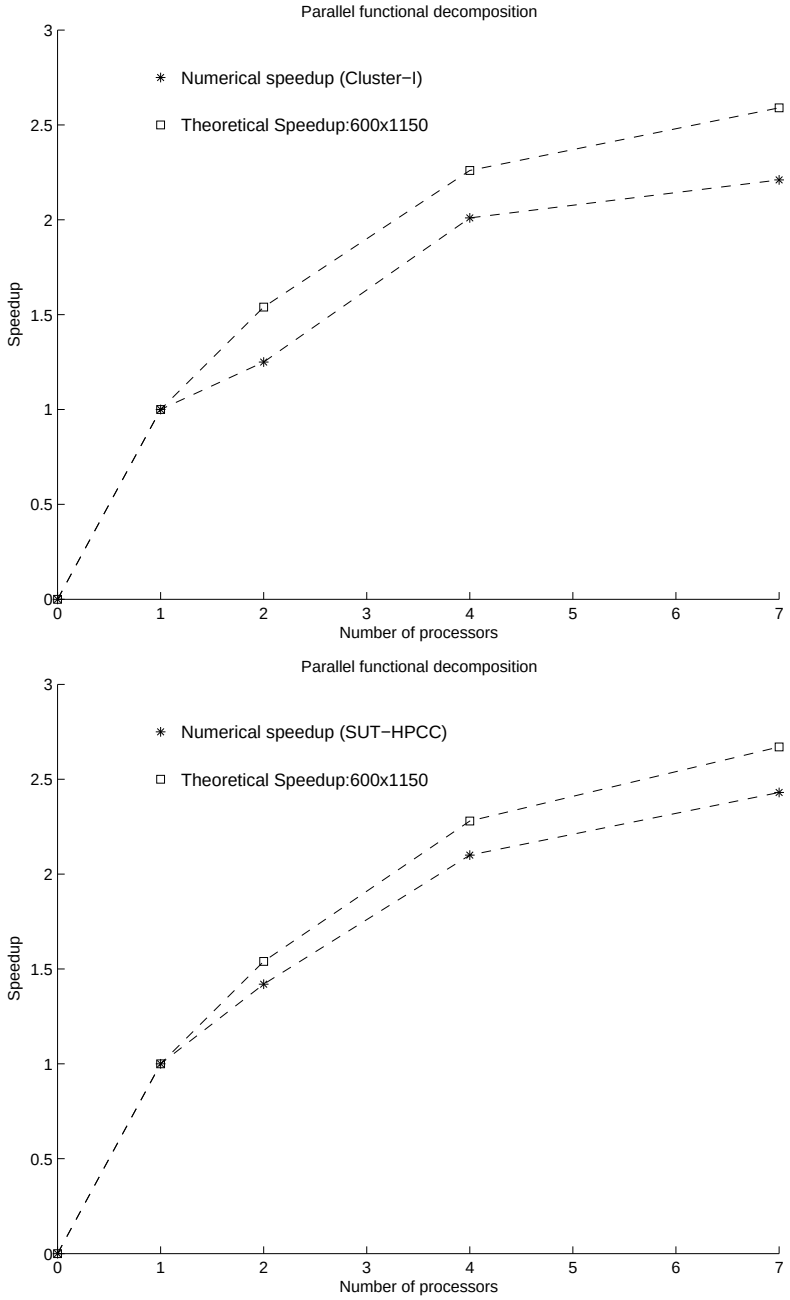

Figure 4.10 Comparison between theoretical speedup and numerical speedups of parallel functional decomposition on Cluster-I and SUT-HPCC, grid size=600 × 1150

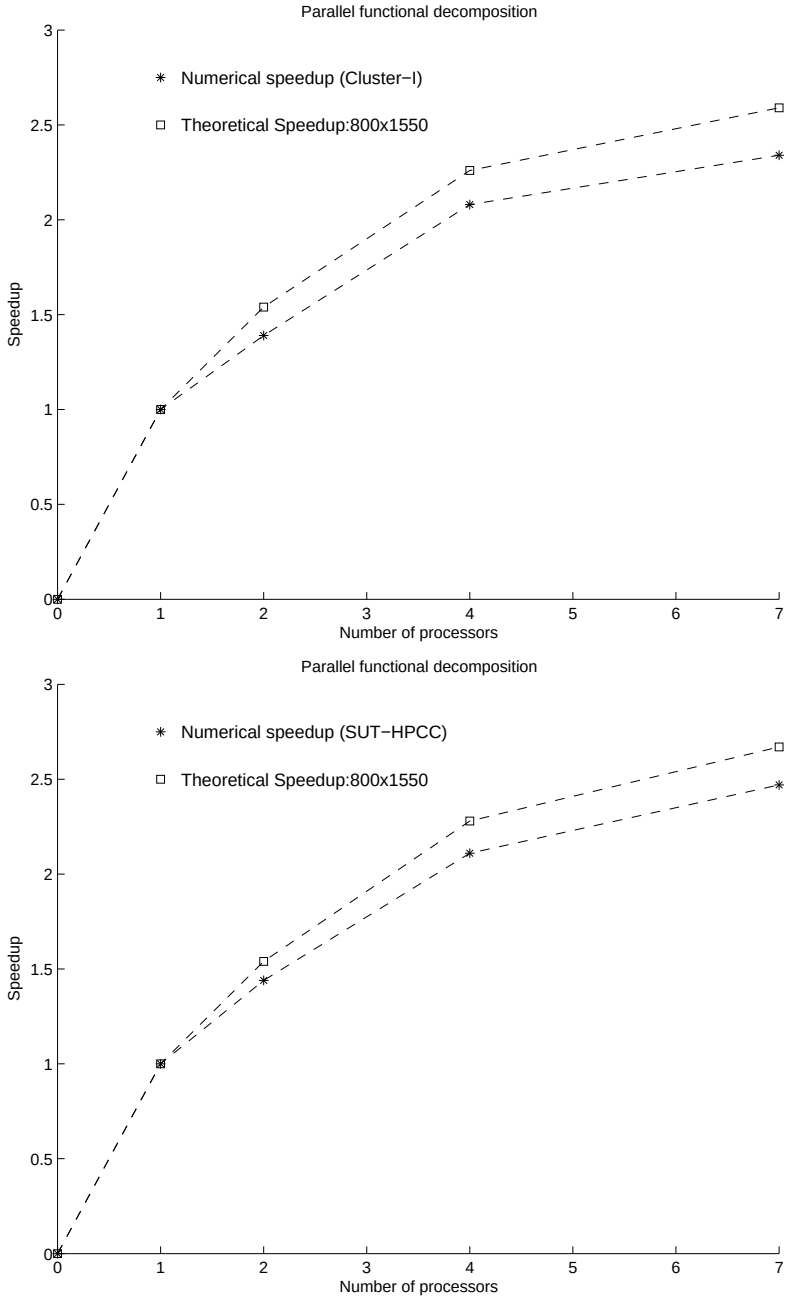

Figure 4.11 Comparison between theoretical speedup and numerical speedups of parallel functional decomposition on Cluster-I and SUT-HPCC, grid size=800 × 1550

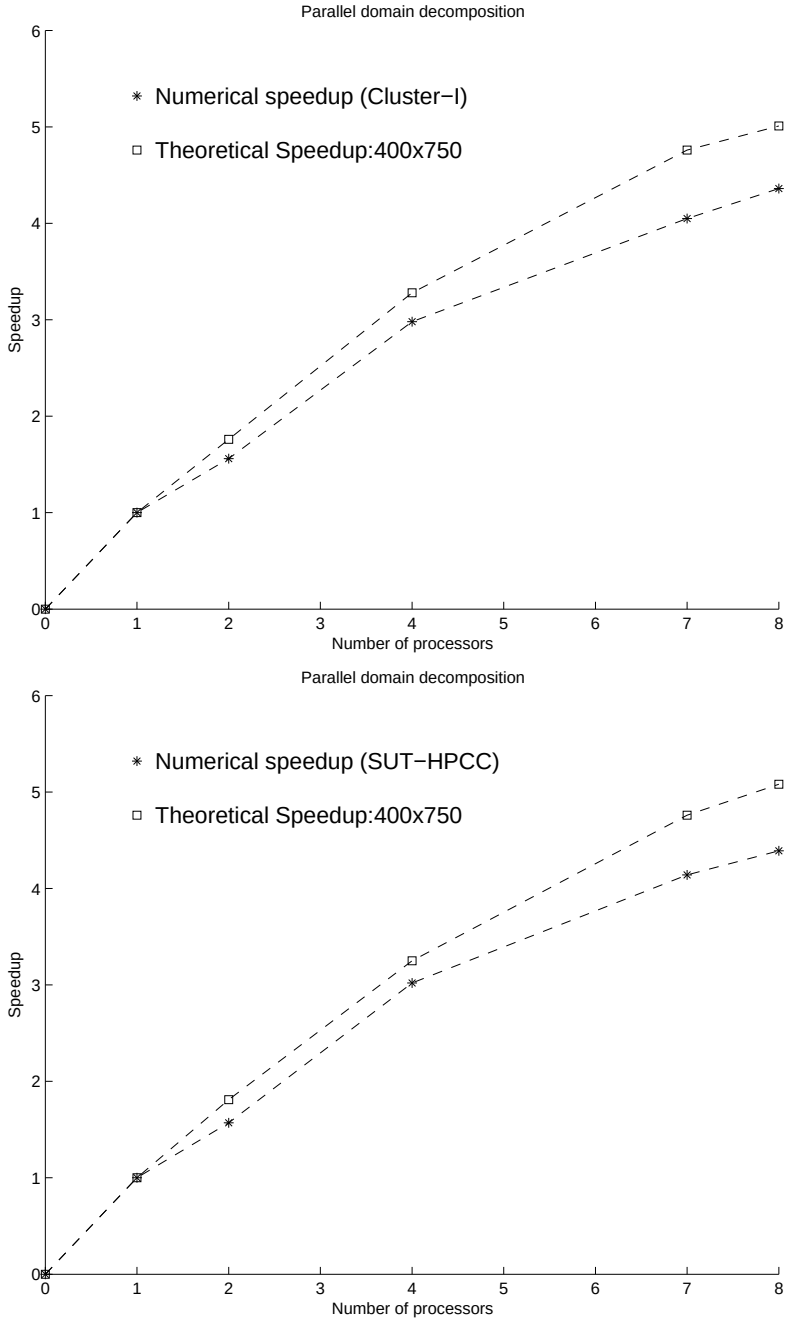

Figure 4.12 Comparison between theoretical speedup and numerical speedups of parallel domain decomposition on Cluste-I and SUT-HPCC, grid size=400×750

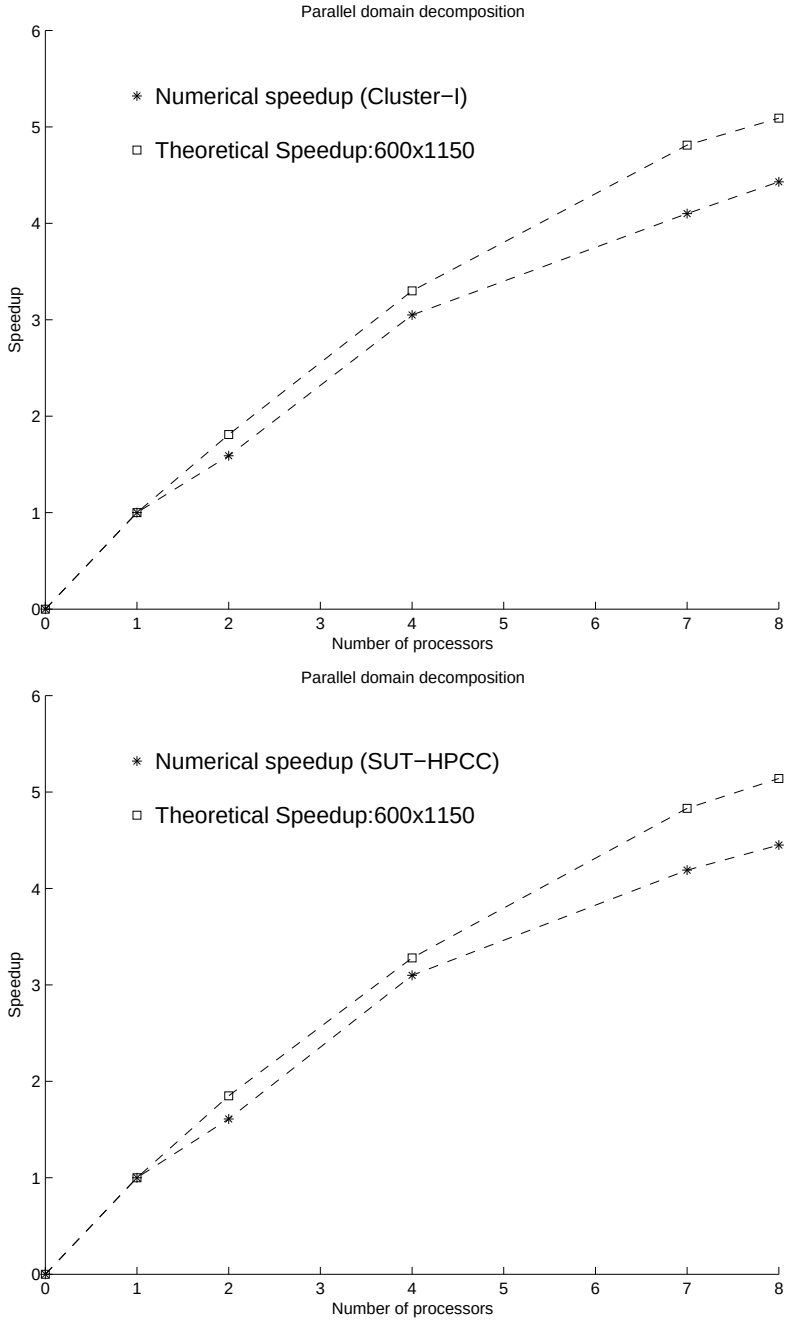

Figure 4.13 Comparison between theoretical speedup and numerical speedups of parallel domain decomposition on Cluster-I and SUT-HPCC, grid size=600×1150

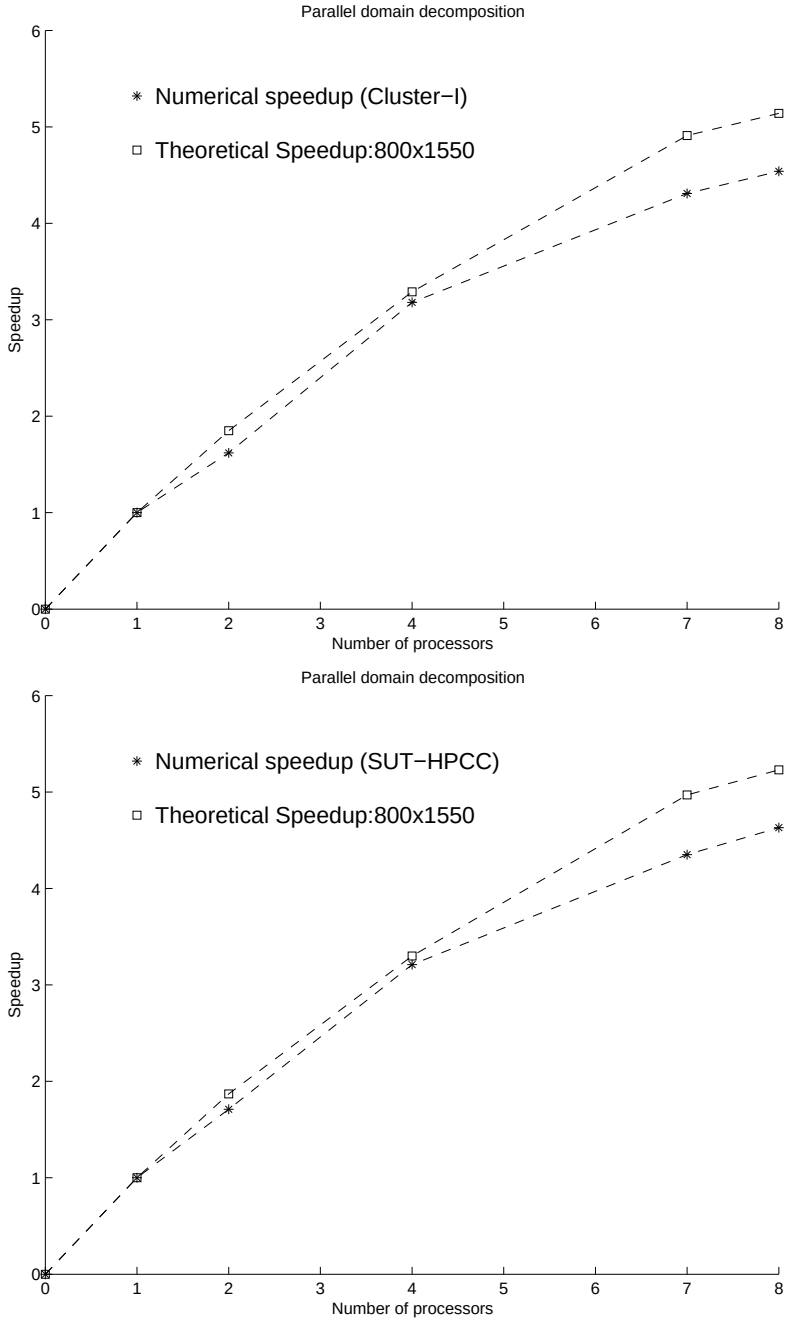

Figure 4.14 Comparison between theoretical speedup and numerical speedups of parallel domain decomposition on Cluster-I and SUT-HPCC, grid size=800×1550

CHAPTER V

CONCLUSIONS

In this final chapter, the major contributions made in this research are summarized, and conclusions are made from the presented results. Finally, recommendations are presented for future research efforts.

5.1 Contributions

The following contributions made in this research can be summarized as follows:

1. Two parallel algorithms and parallel computer codes for numerical models of a turbulent wake dynamics in a stratified fluid have been developed and validated. The first is based on the functional decomposition approach. The second one is based on the domain decomposition techniques. The solution of the transport differential equations are obtained by the fractional step methods.

2. Theoretical estimates of speedups for both functional and domain decomposition techniques are derived.

3. The speedups of both algorithms are compared with the theoretical estimates.

4. Both parallel algorithms are successfully applied to the model of turbulent wake behind axisymmetric towed and self-propelled bodies in a linearly strat-

ified fluid. The model comprises differential equations for transport of normal Reynolds stresses and passive scalar.

The computation in this study was conducted on the Sun Solaris PC cluster and Linux PC cluster in School of Mathematics and High Performance Computing Cluster of the Suranaree University of Technology (Nakhon Ratchasima, Thailand). Numerical simulation have been performed on a variety of grids. The parallel codes were tested on 8 CPUs. The computer codes are developed by FROTRAN-90 language and MPI library.

5.2 Conclusions

The following conclusions can be made from the present work.

1. The theoretical estimates and numerical experiments demonstrate the significant speedup of both parallel algorithms in comparison with the sequential one. The results of numerical simulations of turbulent wake dynamics in a stratified fluid using parallel algorithms are in good agreement with the experimental data and results of numerical simulation by the sequential code.

2. The run time of the functional decomposition algorithm is faster than the run time of sequential code about two times while the run time of the domain decomposition algorithm is faster than that of the sequential code for more than 4 times when we use seven CPUs.

3. To distribute the tasks between CPUs and organize the communications of the parallel functional decomposition algorithm is simpler when compared with the parallel domain decomposition algorithm. But the functional decomposition algorithm has a limitation on the number of processors used and has the lower limit of maximal speedup.

4. The effect of latency time, bandwidth and message size on the speedup have also been analyzed.

Finally, the present research shows that both developed parallel algorithms can be the viable tools to numerical simulations of turbulent wake dynamics in a stratified fluid and can serve as a basis for numerical experiments for more complicated models of turbulence.

5.3 Recommendations for Future Research

Future work should include the mixed techniques of the functional and domain decompositions to solve the mathematical models of a turbulent wakes dynamics in a stratified fluid. Also, the developed methods for new mathematical models of turbulence can be applied to numerical simulation with more complicated flow problems for instance, the turbulent wake dynamics in linearly and nonlinearly stratified fluids, as well as the numerical simulation of swirling turbulent wake dynamics in a stratified fluid.

REFERENCES

REFERENCES

Aoyama Y. and Nakano J., (1999). **RS/6000 SP:Practical MPI programming**, IBM, USA, 221.

Batchelor, G.K., (2000). **Introduction to Fluid Dynamics**, Cambridge University Press, Cambridge, UK, 615.

Chernykh G.G. and Voropayeva O.F. (1999). Numerical modeling of momentumless turbulent wake dynamics in a linearly stratified medium, **Computers and Fluids**, 28(3):281-306.

Chernykh G.G, Fomina A.V. and Moshkin N.P., (2006). Numerical models of turbulent wake dynamics behind a towed body in a linearly stratified medium, **Russ. J. Numer. Anal. Math. Modelling**, 21(5):395-424.

Chernykh G.G., Moshkin N.P. and Fomina A.V., (2008). Dynamics of turbulent wake with small excess momentum in stratifed media, **Commun Nonlinear Sci Numer Simul** ,(in press), doi:10.1016/j.cnsns.2008.01.014.

Damaj I.W. (2006). Parallel algorithms development for programmable logic devices, **Advances in Engineering**, 37(9):561-582.

Dommermuth D.G., Rottman J.W., Innis G.E., and Novikov E.A., (2002). Numerical simulation of the wake of a towed sphere in a weakly stratified fluid, **Fluid Mechanics**, 473:83-101.

Flynn M., (1966). Very high-speed computing systems, **Proc. IEEE**, 54:1901-1909.

Gourlay M.J., Arendt S.C., Fritts D.C., and Werne J., (2001). Numerical modeling of initially turbulent wakes with net momentum, **Physics of Fluids**, 13(12):3783-3802.

Hassid S., (1980). Collapse of turbulent wakes in stable stratified media, **J. Hydronautics**, 14(1):25-32.

Hwang K., (1993). **Advanced computer architecture: Parallelism, scalability, programmabilty**, McGraw-Hill, New York, USA.

Kumar V., GramaA., Gupta A. and Karypis G., (1994). **Introduction to parallel computing**. The Benjamin/Cummings Publishing Company, Inc.

Lin J.T. and Pao Y.H., (1973). Turbulent wakes of a self-propelled slender body in stratified and non-stratified fluids: analysis and flow visualizations, *Bull. Am. Phys. Soc.*, 18:1484.

Lin J.T. and Pao Y.H., (1974). Velocity and Density measurements in the Turbulent Wake of a Propeller- Driven Body in a Startified Fluid, *Flow Research Inc., Rept.*, 36.

Pao Y.H. and Lin J.T., (1974). Turbulent wake of a towed slender body in stratified and nonstratified fluids., *Bull. Am. Phys. Soc.*, 19:1164-1165.

Lin J.T. and Pao Y.H., (1979). Wakes in stratified fluids, **Annu. Rev. Fluid Mech.**, 11:317-336.

Meunier P. and Spedding G.R., (2006). Stratified propelled wakes, **Fluid Mechanics**, 552:229-256.

Moshkin N.P., Chernykh G.G. and Voropayeva O.F., (2001). **Numerical modeling of internal waves generated by turbulent wakes behind**

self-propelled and towed bodies in stratified media, Proc. of 1st Int. Conf. on CFD. Kyoto, Japan, Springer-Verlag, New York-Heidelberg-Berlin, 455-460.

Moshkin N.P., Fomina A.V., Chernykh G.G., (2004). **Dynamics of passive scalar in turbulent wakes in density stratified fluids**, Proc. of the 8-th Annual National Symposium on Computational Science and Engineering, (ANSCSE- 8), 21-23 July 2004, Thailand, Suranaree University of Technology, 307-310.

Nielsen O.M. (2003), Graduate Course in Scientific Computing,**http://datamining.anu.edu.au/ole/**.

Pao Y.J. and Lin J.T., (1973). Turbulent Wake of a Towed Slender Body in stratified and Nonstratified Fluids: Analysis and Flow Visualizations, *Flow Research Inc., Rept.*, 10.

Pao Y.H. and Lin J.T., (1973). Velocity and Density measurements in the Turbulent Wake of a Towed Slender Body in Stratified and Nonstratified Fluids, *Flow Research Inc., Rept.* 12.

Rodi W., (1987). Examples of calculation methods for flow and mixing in stratified fluids, **J. Geophys. Res.**, 92(C5):5305-5328.

Samarskij and Aleksandr A., (1989). **Numerical Methods for Grid Equations**, Birkhauser Verlag Basel, 101.

Spedding G.R., (2001). Anisotropy in turbulence profiles of stratified wakes, **Physics of Fluids**, 13(8):2361-2372.

Schönauer W., (2000). **Scientific supercomputing architecture and use of**

shared and distributed memory parallel computing. Self-edition by Willi Schönauer, Germany.

Voropaeva O.F., Ilyushin B.B., and Chernykh G.G., (2002) **Numerical Simulation of the Far Momentumless Turbulent Wake in a Linearly Stratified Medium**, Doklady Physics, 47(10):762-766.

Voropayeva O.F.,Moshkin N.P. and Chernykh G.G., (2000). Internal waves generated by turbulent wakes behind towed and self-propelled bodies in linearly stratified medium, **Mathematical Modeling**, 12(10):77-94 (In Russian).

Yanenko, N.N., (1971). **Method of Fractional Steps**, Gordan and Breach, NY, USA.

APPENDICES

APPENDIX A

THE COMPUTATION OF THE THEORETICAL SPEEDUP

In this Appendix, we show by the example how to estimate the theoretical speedup. In this example, we chose the largest grid of grid size 800×1550, and computed the speedpup for the SUT-HPCC cluster.

A.1 The Theoretical Speedup of the Domain Decomposition Algorithms

To estimate the theoretical speedup of the domain decomposition algorithm on SUT-HPCC, we use the measured latency time, $t_l = 5.9E{-}5$ sec, 260 MB/sec of the bandwidth and the sequential run time of all subroutines for the computation of one iteration.

Table A.1 shows the sequential run time for each component, the communication times, and the speedup. The message size for a grid of size 800×1550 is determined by

$$MessageSize = \frac{800 \times 1550 \times 8}{1024^2 \times P^2} \text{ MB.}$$

The communication time (T_{comm}) can be approximated by

$$\begin{aligned} T_{comm} &= P(P-1)\left(t_l + \frac{MessageSize}{Bandwidth}\right) \\ &= P(P-1)\left(0.000059 + \frac{800 \times 1550 \times 8}{1024^2 \times P^2 \times 260}\right). \end{aligned} \tag{A.1}$$

In the implicit splitting method, we need to update data twice for 10 components

and once for 4 components. So, in one iteration, we require $(10 \times 2 + 4)\, T_{comm}$ of the communication time. By formula (A.1), this amounts to $0.00146P(P-1) + 0.8733(P-1)/P$.

A.2 The Theoretical Speedup of the Functional Decomposition Algorithms

This an example of estimation for 4 CPUs (CPU(0), CPU(1), CPU(2) and CPU(3)). In the algorithm, we use the `MPI_BCAST` command to update the data of message size 800×1550 which was measured to require about 0.014392 sec of the communication time.

Stage 1: For U_d, V^* and W^*, we can compute U_d on CPU(0) and CPU(2). Since the run time of V^* and W^* are small when compared with U_d, hence, we can compute these two components on the same processors of CPU(1) and CPU(3). Then we can update the data simultaneously between CPU(0)-CPU(1) and CPU(2)-CPU(3).

Stage 2: All processors can compute for $\langle p_1 \rangle^{n+1}$ simultaneously.

Stage 3: For V and W, we assign CPU(0) and CPU(2) to compute for V while CPU(1) and CPU(3) compute for W. Then we can update the data simultaneously between CPU(0)-CPU(1) and CPU(2)-CPU(3).

Stage 4: In this stage, there are 7 components of $\langle \rho_1 \rangle$, ε, $\langle v'w' \rangle$, $\langle u'^2 \rangle$, $\langle v'^2 \rangle$, $\langle w'^2 \rangle$ and Θ. So we can compute simultaneously for $\langle \rho_1 \rangle$ on CPU(0), ε on CPU(1), $\langle v'w' \rangle$ and $\langle u'^2 \rangle$ on CPU(2) and $\langle v'^2 \rangle$ and $\langle w'^2 \rangle$ on CPU(3). We need to update the data for all processors (except for the passive scalar Θ).

Stage 5: For e, all processors can compute simultaneously.

Grid size	800x1550				
NCPUs	P=1	P=2	P=4	P=8	P=16
U_d	0.927328	0.463664	0.231832	0.115916	0.057958
V^*	0.424598	0.212299	0.106149	0.053075	0.026537
W^*	0.413166	0.206583	0.103292	0.051646	0.025823
$\langle p_1 \rangle$	2.498945	1.249473	0.624736	0.312368	0.156184
V	0.431262	0.215631	0.107816	0.053908	0.026954
W	0.408874	0.204437	0.102219	0.051109	0.025555
$\langle \rho_1 \rangle$	1.335131	0.667566	0.333783	0.166891	0.083446
ε	1.003255	0.501628	0.250814	0.125407	0.062703
$\langle v'w' \rangle$	0.976604	0.488302	0.244151	0.122075	0.061038
$\langle u'^2 \rangle$	0.973988	0.486994	0.243497	0.121749	0.060874
$\langle v'^2 \rangle$	0.991730	0.495865	0.247933	0.123966	0.061983
$\langle w'^2 \rangle$	0.980651	0.490326	0.245163	0.122581	0.061291
Θ	0.991672	0.495836	0.247918	0.123959	0.061979
e	0.360719	0.180359	0.090180	0.045090	0.022545
T_{comm}	-	0.439468	0.671946	0.843409	1.158533
T_P	12.717923	6.798430	3.851427	2.433150	1.953403
Speedup	-	1.870715	3.302133	5.226938	6.510650

Table A.1 The theoretical speedup of the domain decomposition algorithm for grid size=800 $\times$ 1550 on SUT-HPCC

NCPUs		P=1	P=4
stage j=1	U_d	0.927328	0.927328
	V^*	0.424598	
	W^*	0.413166	
	T_{comm}	-	0.043176 (3 messages)
stage j=2	$\langle p_1 \rangle^{n+1}$	2.498945	2.498945
	T_{comm}	-	-
stage j=3	V	0.431262	0.431262
	W	0.408874	
	T_{comm}	-	0.028784(2 messages)
stage j=4	$\langle \rho_1 \rangle$	1.335131	1.335131
	ε	1.003255	
	$\langle v'w' \rangle$	0.976604	
	$\langle u'^2 \rangle$	0.973988	
	$\langle v'^2 \rangle$	0.991730	
	$\langle w'^2 \rangle$	0.980651	
	Θ	0.991672	
	T_{comm}	-	0.086352 (6 messages)
stage j=5	e	0.360719	0.386185
	T_{comm}	-	-
	Total	12.717923	5.711697
	Speedup	-	2.226645

Table A.2 The theoretical speedup of the functional decomposition algorithm for grid size=800×1550 on SUT-HPCC

APPENDIX B

TERMINOLOGY

B.1 Finite Difference Method

The principle of the finite-difference method is to replace the differential operators by combinations of algebraic finite-difference quotients that can be received from truncations of Taylor series. When all the partial derivatives in a given partial differential equation are replaced by finite-difference quotients, the resulting algebraic equation is called **difference equation**, which is an algebraic representation of the partial differential equation.

B.2 The Method of Stabilizing Corrections

In this research we use the iterative method of stabilizing corrections (Yanenko (1971)) for computing the finite difference equation (3.10) for pressure. The method of stabilizing corrections, which was introduced by Douglas and Rachford (1956) and formulated in its general form by Douglas and Gunn (1964), is a very general and effective method for the construction of schemes with fractional steps. We present here the general iterative scheme of stabilizing corrections for elliptic equations. In our explanation, we follow Yanenko (1970).

For the elliptic equation

$$Lu + f = \sum_{i,j=1}^{m} a_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} + f = 0. \tag{B.1}$$

the parallel between the iterative schemes and integration schemes of the corresponding parabolic equation

$$\frac{\partial u}{\partial t} = \sum_{i,j=1}^{m} a_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} + f. \tag{B.2}$$

is always valid, i.e., the solution of the unsteady problem $(B.2)$ approaches the solution of the steady problem with the same boundary conditions, regardless of the choice of initial data. The scheme of stabilizing corrections is

$$\begin{aligned} \frac{u^{n+1/m} - u^n}{\tau} &= \Lambda_{11} u^{n+1/m} + (\Omega - \Lambda_{11}) u^n, \\ \frac{u^{n+2/m} - u^{n+1/m}}{\tau} &= \Lambda_{22}(u^{n+2/m} - u^n), \\ \cdots &= \cdots, \\ \frac{u^{n+1} - u^{n+(m-1)/m}}{\tau} &= \Lambda_{mm}(u^{n+1} - u^n), \end{aligned} \tag{B.3}$$

where $\Omega = \sum_{i,j=1}^{n} \Lambda_{ij}$. After eliminating fractional steps, the equivalent scheme in whole steps is

$$\begin{aligned} \frac{u^{n+1} - u^n}{\tau} &= \Lambda u^{n+1} + (\Omega - \Lambda) u^n - \tau \sum_{i<j} \Lambda_{ii}\Lambda_{jj}(u^{n+1} - u^n) + \\ &+ \tau^2 \sum_{i<j<k} \Lambda_{ii}\Lambda_{jj}\Lambda_{kk}(u^{n+1} - u^n) + \cdots \\ &+ (-1)^{m-1} \Lambda_{11} \ldots \Lambda_{mm} \tau^{m-1} (u^{n+1} - u^n), \\ \Lambda &= \sum_{i=1}^{m} \Lambda_{ii}, \quad i, j, k = 1, \ldots, m. \end{aligned} \tag{B.4}$$

From $(B.4)$ complete consistency follows at any m. Scheme $(B.3)$ is strongly stable. The main idea of the scheme of stabilizing correction is to solve at each fractional step the system of algebraic equations only with tridiagonal matrix. Next we give a short description of the "sweep" method of solution of three-point equations and the cyclic elimination method for three-point equations. The three-point equations arise from three-point difference schemes designed to find periodic solutions of second-order ordinary differential equation and also when

approximating the solutions of equations with partial derivatives in cylindrical bipolar coordinates.

B.3 The Elimination Method for Three-Point Equations (Samarskij (1989))

Suppose we must solve the following system of three-point equations

$$\begin{aligned} c_0 y_0 - b_0 y_1 &= f_0, \qquad i = 0, \\ -a_i y_{i-1} + c_i y_i - b_i y_{i+1} &= f_i, \qquad 1 \le i \le N-1, \\ -a_N y_{N-1} + c_N y_N &= f_N, \qquad i = N, \end{aligned} \tag{B.5}$$

or, in vector form,

$$\mathbf{A}Y = F \tag{B.6}$$

where $Y = (y_0, y_1, \ldots, y_N)^T$ is the vector of unknowns, $F = (f_0, f_1, \ldots, f_N)^T$ is the right hand side vector, and A is the square $(N+1) \times (N+1)$ matrix with real or complex coefficients.

$$\mathbf{A} = \begin{pmatrix} c_0 & -b_0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ -a_1 & c_1 & -b_1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -a_2 & c_2 & -b_2 & \cdots & 0 & 0 & 0 & 0 \\ . & . & . & . & \cdots & . & . & . & . \\ 0 & 0 & 0 & 0 & \cdots & -a_{N-2} & c_{N-2} & -bN-2 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -a_{N-1} & c_{N-1} & -b_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -a_N & c_N \end{pmatrix}$$

Systems of the form $(B.5)$ arise from a three-point approximation to a boundary-value problem for second-order ordinary differential equations with constant and variable coefficients, and also when realizing difference schemes for equations with partial derivatives.

Following the idea of Gauss' method, we carry out the elimination of the unknown in $(B.5)$. We introduce the notation $\alpha_1 = b_0/c_0, \beta_1 = f_0/c_0$ and write $(B.5)$ in the following form

$$\begin{aligned} y_0 - \alpha_1 y_1 &= \beta_1, \qquad i = 0, \\ -a_i y_{i-1} + c_i y_i - b_i y_{i+1} &= f_i, \qquad 1 \le i \le N-1, \\ -a_N y_{N-1} + c_N y_N &= f_N, \qquad i = N, \end{aligned} \tag{B.7}$$

Take the first two equations of the system $(B.7)$

$$y_0 - \alpha_1 y_1 = \beta_1, \quad -a_1 y_0 + c_1 y_1 - b_1 y_2 = f_1.$$

Multiply the first equation by a_1 and add it to the second equation. We get $(c_1 - a_1\alpha_1)y_1 - b_1 y_2 = f_1 + \alpha_1\beta_1$ or, after dividing by $c_1 - a_1\alpha_1$

$$y_1 - \alpha_2 y_2 = \beta_2, \quad \alpha_2 = \frac{b_1}{c_1 - \alpha_1 a_1}, \quad \beta_2 = \frac{f_1 + a_1\beta_1}{_1 - \alpha_1 a_1}.$$

All the remaining equations of the system $(B.7)$ do not contain y_0, therefore this stage of the elimination process is completed. As a result we obtain a new "reduced" system

$$\begin{aligned} y_1 - \alpha_2 y_2 &= \beta_2, \qquad i = 1, \\ -a_i y_{i-1} + c_i y_i - b_i y_{i+1} &= f_i, \qquad 2 \le i \le N-1, \\ -a_N y_{N-1} + c_N y_N &= f_N, \qquad i = N, \end{aligned} \tag{B.8}$$

which does not contain the unknown y_0 and which has a structure analogous to $(B.7)$. When this system has been solved, the unknown y_0 is found from the formula $y_0 = \alpha_1 y_1 + \beta_1$. We can apply the above described elimination procedure to system $(B.8)$. At the second stage, the unknown y_1 is eliminated, at the third y_2, and so forth. At the end of the l^{th} stage we obtain a system for the unknowns

$y_l, y_l + 1, \cdots, y_N$

$$\begin{aligned} y_l - \alpha_{l+1} y_{l+1} &= \beta_{l+1}, \qquad i = 1, \\ -a_i y_{i-1} + c_i y_i - b_i y_{i+1} &= f_i, \qquad l+1 \le i \le N-1, \\ -a_N y_{N-1} + c_N y_N &= f_N, \qquad i = N, \end{aligned} \tag{B.9}$$

and formulas for finding y_i for $i \le l - 1$

$$y_i = \alpha_{i+1} y_{i+1} + \beta_{i+1}, \quad i = l-1, l-2, \ldots, 0. \tag{B.10}$$

The coefficients α_i and β_i, clearly, are found from the formulas

$$\alpha_{i+1} = \frac{b_i}{c_i - \alpha_i a_i}; \quad \beta_{i+1} = \frac{f_i + a_i \beta_i}{c_i - \alpha_i a_i};\ i = 1, 2, \ldots, ;\ \alpha_1 = \frac{b_0}{c_0},\ \beta_1 = \frac{f_0}{c_0}.$$

Substituting $l = N - 1$ in $(B.9)$, we obtain a system for y_N and y_{N-1}

$$y_{N-1} - \alpha_N y_N = \beta_N, -a_N y_{N-1} + c_N y_N = f_N \tag{B.11}$$

from which we find $y_N = \beta_{N+1},\ y_{N-1} = \alpha_N y_N + \beta_N$.

Combining these equations with $(B.10)$ $(l = N - 1)$, we obtain the final formulas for finding the unknowns

$$\begin{aligned} y_i &= \alpha_{i+1} y_{i+1} + \beta_{i+1}, \qquad i = N-1, N-2, \ldots, 0, \\ y_N &= \beta_N + 1, \end{aligned} \tag{B.12}$$

where α_i and β_i are found from the recurrence formulas

$$\begin{aligned} \alpha_{i+1} &= \frac{b_i}{c_i - a_i \alpha_i}, \qquad i = 1, 2, \ldots, N-1, \qquad \alpha_i = \frac{b_0}{c_0}, \\ \beta_{i+1} &= \frac{f_i + a_i \beta_i}{c_i - a_i \alpha_i}, \qquad i = 1, 2, \ldots, N, \qquad \beta_i = \frac{f_0}{c_0}. \end{aligned} \tag{B.13}$$

Thus, the formulas $(B.12)$ and $(B.13)$ describe Gauss'method which, when applied to the system $(B.5)$, is given a special name - the *elimination method.* The coefficients α_i and β_i are called the *elimination coefficients*, formulas $(B.13)$ describe the *forward elimination pate*, and $(B.12)$ the *backward path.* Since the values y_i are

found sequentially in reverse order, the formulas ($B.12$) and ($B.13$) are sometimes called the *right-elimination formulas*.

An elementary count of the arithmetic operations in ($B.12$) and ($B.13$) shows that realizing the elimination method using these formulas requires $3N$ multiplications, $2N+1$ divisions and $3N$ additions and subtractions. If there is no difference between arithmetic operations, the total number of operations required for the elimination method is $Q = 8N + 1$. Of this total, $3N - 2$ operations are used for computing α_i, and $5N + 3$ operations for computing β_i and y_i.

Notice that the coefficients α_i do not depend on the right-hand side of the system ($B.5$), but are determined solely by the coefficients a_i, b_i, c_i of the difference equations. Therefore, if we must solve a series of problems ($B.5$) with different right-hand sides, but with the same matrix $\mathbf{A}$, then the elimination coefficients α_i are only computed for the first problem of the series. Thus solving the first problem in the series costs $Q = 8N + 1$ operations, but solving each of the remaining problems only costs $5N + 3$ operations.

In conclusion we indicate the order of the computations for the formulas of the elimination method. Beginning with α_1 and β_1, we calculate and store α_i and β_i using ($B.13$). Then the solutions y_i are found using ($B.12$).

B.4 Run Time and Speedup

Run time: The serial run time (T_S) of a program is the time elapsed between the beginning and the end of its execution on a sequential computer. The parallel run time (T_P) is the time that elapsed from the moment that a parallel computational starts to the moment that the last processor finishes execution.

Speedup: Speedup is a measure that captures the relative benefit of

solving a program in parallel. It is defined as the ratio of the time taken to solve a program on a single processor to the time required to solve the same problem on a parallel computer. We denote speedup by the symbol S (i.e. $S = T_S/T_P$).

Efficiency: Only an ideal parallel system containing P processors can deliver a speedup equal to P since, parallel computing have the time of communications. Efficiency is defined as the ratio of speedup to the number of processors (i.e. $E = S/P$). In an ideal parallel system, efficiency is equal to 1. In practice, speedup is less than P and efficiency is between zero and one.

APPENDIX C

FREQUENTLY USED MPI SUBROUTINES

In this Appendix, we review the main subroutines of using MPI library (Aoyama and Nakano (1999)).

C.1 Environmental Subroutines

1.1 MPI_INIT

Purpose : Initializes MPI.

Usage: `CALL MPI_INIT(ierr)`

Parameters:

INTEGER ierr: The Fortran return code.

Description: This subroutine initializes MPI. All MPI programs must call this subroutine once and only once before any other MPI subroutine.

1.2 MPI_COMM_SIZE

Purpose: Returns the number of processes in the group associated with a communicator.

Usage: `CALL MPI_COMM_SIZE(comm, size, ierror)`

Parameters:

INTEGER comm: The communicator.

INTEGER size: An integer specifying the number of processes in

the group comm.

INTEGER ierror: The Fortran return code.

Description: This routine returns the size of the group associated with a communicator.

1.3 MPI_COMM_RANK

Purpose: Returns the rank of the local process in the group associated with a communicator.

Usage: `CALL MPI_COMM_RANK(comm, rank, ierror)`

Parameters:

INTEGER comm: The communicator.

INTEGER rank: An integer specifying the rank of the calling process in group comm.

INTEGER ierror: The Fortran return code.

Description: This routine returns the rank of the local process in the group associated with a communicator. MPI_COMM_RANK indicates the rank of the process that calls it in the range from 0..size - 1, where size is the return value of MPI_COMM_SIZE.

1.4 MPI_FINALIZE

Purpose: Terminates all MPI processing.

Usage: `CALL MPI_FINALIZE(ierror)`

Parameters:

INTEGER ierror: The Fortran return code.

Description: Make sure this routine is the last MPI call. Any MPI calls made after MPI_FINALIZE raise an error.

1.5 MPI_ ABORT

Purpose: Forces all processes of an MPI job to terminate.

Usage: `CALL MPI_ABORT(comm, errorcode, ierror)`

Parameters:

INTEGER comm: The communicator of the processes to abort.

INTEGER errorcode: The error code returned to the invoking environment.

INTEGER ierror: The Fortran return code.

Description: If any process calls this routine, all processes in the job are forced to terminate.

C.2 Collective Communication Subroutines

2.1 MPI_BCAST

Purpose : Broadcasts a message from root to all processes in comm.

Usage: `CALL MPI_BCAST(buffer, count, datatype, root, comm, ierror)`

Parameters:

buffer : The starting address of the buffer.

INTEGER count : The number of elements in the buffer.

INTEGER datatype: The data type of the buffer elements.

INTEGER root : The rank of the root process.

INTEGER comm : The communicator.

INTEGER ierror : The Fortran return code.

Description : This routine broadcasts a message from root to all processes in comm. The contents of roots communication buffer is copied to all processes on return. The type signature of count, datatype on any process must be equal to the type signature of count, datatype at the root. This means the amount of data sent must be equal to the amount of data received, pairwise between each process and the root. Distinct type maps between sender and receiver are allowed. All processes in comm need to call this routine.

2.2 MPI_REDUCE

Purpose : Applies a reduction operation to the vector sendbuf

over the set of processes specified by comm and places

the result in recvbuf on root.

Usage : `CALL MPI_REDUCE(sendbuf, recvbuf, count, datatype, op, root, comm, ierror)`

Parameters :

sendbuf : The address of the send buffer (IN).

recvbuf : The address of the receive buffer.

INTEGER count : The number of elements in the send buffer.

INTEGER datatype : The data type of elements of the send buffer.

INTEGER op : The reduction operation.

INTEGER root : The rank of the root process.

INTEGER comm : The communicator.

INTEGER ierror : The Fortran return code.

Description : This routine applies a reduction operation to the vector sendbuf over the set of processes specified by comm and places the result in recvbuf on root. Both the input and output buffers have the same number of elements with the same type. The arguments sendbuf, count, and datatype define the send or input buffer and recvbuf, count and datatype define the output buffer.

2.3 MPI_ALLREDUCE

Purpose : Applies a reduction operation to the vector sendbuf over the set of processes specified by comm and places the result in recvbuf on all of the processes in comm.

Usage : `CALL MPI_ALLREDUCE(sendbuf, recvbuf, count, datatype, op,comm, ierror)`

Parameters :

sendbuf : The starting address of the send buffer (IN).

recvbuf : The starting address of the receive buffer. sendbuf and recvbuf cannot overlap in memory (OUT)

INTEGER count : The number of elements in the send buffer (IN)

INTEGER datatype : The data type of elements of the send buffer (handle) (IN)

INTEGER op : The reduction operation (handle) (IN)

INTEGER comm : The communicator (handle) (IN)

INTEGER ierror : The Fortran return code

Description : This routine applies a reduction operation to the vector sendbuf over the set of processes specified by comm and places the result in recvbuf on all of the processes.

2.4 MPI_BARRIER

Purpose : Blocks each process in comm until all processes have called it.

Usage : `CALL MPI_BARRIER(comm, ierror)`

Parameters :

INTEGER comm : The communicator (handle) (IN)

INTEGER ierror : The Fortran return code

Description : This routine blocks until all processes have called it. Processes cannot exit the operation until all group members have entered. All processes in comm need to call this routine.

C.3 Point-to-Point Communication Subroutines

3.1 MPI_SEND

Purpose : Performs a blocking standard mode send operation.

Usage : `CALL MPI_SEND(buf, count, datatype, dest, tag, comm, ierror)`

Parameters :

buf : The initial address of the send buffer (IN).

INTEGER count : The number of elements in the send buffer (IN).

INTEGER datatype : The data type of each send buffer element (handle) (IN).

INTEGER dest : The rank of the destination process in comm.

INTEGER tag : The message tag.

INTEGER comm : The communicator (handle) (IN).

INTEGER ierror : The Fortran return code.

Description : This routine is a blocking standard mode send. MPI_SEND causes count elements of type datatype to be sent from buf to the process specified by dest. dest is a process rank which can be any value from 0 to n-1, where n is the number of processes in comm. The message sent by MPI_SEND can be received by either MPI_RECV or MPI_IRECV.

3.2 MPI_RECV

Purpose : Performs a blocking receive operation.

Usage :`CALL MPI_RECV(buf, count, datatype, source, tag, comm, status, ierror)`

Parameters :

buf : The initial address of the receive buffer (OUT).

INTEGER count : The number of elements to be received (IN).

INTEGER datatype : The data type of each receive buffer element (handle) (IN).

INTEGER source : The rank of the source process in comm, MPI_ANY_SOURCE.

INTEGER tag : The message tag or MPI_ANY_TAG (IN).

INTEGER comm : The communicator (handle) (IN).

INTEGER status : The status object (OUT).

INTEGER ierror : The Fortran return code.

Description : MPI_RECV is a blocking receive. The receive buffer is storage containing room for count consecutive elements of the type specified by datatype, starting at address buf. The message received must be less than or equal to the length of the receive buffer. If all incoming messages do not fit without truncation, an overflow error occurs. If a message arrives that is shorter than the receive buffer, then only those locations corresponding to the actual message are changed. MPI_RECV can receive a message sent by either MPI_SEND or MPI_ISEND.

3.3 MPI_ISEND

Purpose : Performs a nonblocking standard mode send operation.

Usage : `CALL MPI_ISEND(buf, count, datatype, dest, tag, comm, request, ierror)`

Parameters :

buf : The initial address of the send buffer (IN).

INTEGER count : The number of elements in the send buffer (IN).

INTEGER datatype : The data type of each send buffer element (handle) (IN).

INTEGER dest : The rank of the destination process in comm.

INTEGER tag : The message tag.

INTEGER comm : The communicator (handle) (IN).

INTEGER request : The communication request (handle) (OUT).

INTEGER ierror : The Fortran return code.

Description : This routine starts a nonblocking standard mode send. The send buffer may not be modified until the request has been completed by MPI_WAIT. The message sent by MPI_ISEND can be received by either MPI_RECV or MPI_IRECV.

3.4 MPI_IRECV

Purpose : Performs a nonblocking receive operation.

Usage : `CALL MPI_IRECV(buf, count, datatype, source, tag, comm, request, ierror)`

Parameters :

buf : The initial address of the receive buffer (OUT).

INTEGER count : The number of elements in the receive buffer (IN).

INTEGER datatype : The data type of each receive buffer element (handle) (IN).

INTEGER source : The rank of source, MPI_ANY_SOURCE.

INTEGER tag : The message tag or MPI_ANY_TAG (IN).

INTEGER comm : The communicator (handle) (IN).

INTEGER request : The communication request (handle) (OUT).

INTEGER ierror : The Fortran return code.

Description : This routine starts a nonblocking receive and returns a handle to a request object. A nonblocking receive call means the system may start writing data into the receive buffer. Once the nonblocking receive operation

is called, do not access any part of the receive buffer until the receive is complete. The message received must be less than or equal to the length of the receive buffer. If all incoming messages do not fit without truncation, an overflow error occurs. If a message arrives that is shorter than the receive buffer, then only those locations corresponding to the actual message are changed. If an overflow occurs, it is flagged at the MPI_WAIT or MPI_TEST. MPI_IRECV can receive a message sent by either MPI_SEND or MPI_ISEND.

3.5 MPI_WAIT

Purpose : Waits for a nonblocking operation to complete.

Usage : `CALL MPI_WAIT(request, status, ierror)`

Parameters :

INTEGER request : The request to wait for (handle) (INOUT).

INTEGER status : The status object (OUT).

INTEGER ierror : The Fortran return code.

Description : MPI_WAIT returns after the operation identified by request completes. If the object associated with request was created by a nonblocking operation, the object is deallocated and request is set to MPI_REQUEST_NULL.

C.4 Derived Data Types

4.1 MPI_TYPE_CONTIGUOUS

Purpose : Returns a new data type that represents the concatenation of count instances of oldtype.

Usage : `CALL MPI_TYPE_CONTIGUOUS(count, oldtype, newtype, ierror)`

Parameters :

INTEGER count : The replication count (IN).

INTEGER oldtype : The old data type (handle) (IN).

INTEGER newtype : The new data type (handle) (OUT).

INTEGER ierror : The Fortran return code.

Description : This routine returns a new data type that represents the concatenation of count instances of oldtype. MPI_TYPE_CONTIGUOUS allows replication of a data type into contiguous locations. newtype must be committed using MPI_TYPE_COMMIT before being used for communication.

4.2 MPI_TYPE_VECTOR

Purpose : Returns a new data type that represents equally spaced blocks. The spacing between the start of each block is given in units of extent (oldtype).

Usage : `CALL MPI_TYPE_VECTOR(count, blocklength, stride, oldtype, newtype, ierror)`

Parameters :

INTEGER count : The number of blocks (IN).

INTEGER blocklength : The number of oldtype instances in each block (IN).

INTEGER stride : The number of units between the start of each block (IN).

INTEGER oldtype : The old data type (handle) (IN).

INTEGER newtype : The new data type (handle) (OUT).

INTEGER ierror : The Fortran return code.

Description : This function returns a new data type that represents count equally spaced blocks. Each block is a concatenation of blocklength instances of oldtype. The origins of the blocks are spaced stride units apart where the counting unit is extent(oldtype). That is, from one origin to the next in bytes = stride * extent (oldtype). newtype must be committed using MPI_TYPE_COMMIT before being used for communication.

4.3 MPI_TYPE_COMMIT

Purpose : Makes a data type ready for use in communication.

Usage : `CALL MPI_TYPE_COMMIT(datatype, ierror)`

Parameters :

INTEGER datatype : The data type that is to be committed (handle) (INOUT).

INTEGER ierror : The Fortran return code.

Description : A data type object must be committed before you can use it in communication. You can use an uncommitted data type as an argument in data type constructors. This routine makes a data type ready for use in communication. The data type is the formal description of a communication buffer. It is not the content of the buffer. Once the data type is committed it can be repeatedly reused to communicate the changing contents of a buffer or buffers with different starting addresses.

4.4 MPI_TYPE_EXTENT

Purpose : Returns the extent of any defined data type.

Usage : `CALL MPI_TYPE_EXTENT(datatype, extent, ierror)`

Parameters :

INTEGER datatype : The data type (handle) (IN).

INTEGER extent : The data type extent (integer) (OUT).

INTEGER ierror : The Fortran return code.

Description : This routine returns the extent of a data type. The extent of a data type is the span from the first byte to the last byte occupied by entries in this data type and rounded up to satisfy alignment requirements. Rounding for alignment is not done when MPI_UB is used to define the data type. Types defined with MPI_LB, MP_UB or with any type that contains MPI_LB or MPI_UB may return an extent which is not directly related to the layout of data in memory. Refer to MPI_TYPE_STRUCT for more information on MPI_LB and MPI_UB.

4.5 MPI_COMM_SPLIT

Purpose : Splits a communicator into multiple communicators based on `color` and `key`.

Usage : `CALL MPI_COMM_SPLIT(comm, color, key, newcomm, ierror)`

Parameters :

INTEGER comm : The communicator (handle) (IN).

INTEGER color : An integer specifying control of subset assignment (IN).

INTEGER key : An integer specifying control of rank assignment (IN).

INTEGER newcomm : The new communicator (handle) (OUT).

INTEGER ierror : The Fortran return code.

Description : MPI_COMM_SPLIT is a collective function that partitions the group associated with comm into disjoint subgroups, one for each value of color. Each subgroup contains all processes of the same color. Within each subgroup, the processes are ranked in the order defined by the value of the argument key. Ties are broken according to their rank in the old group. A new communicator is created for each subgroup and returned in newcomm. If a process supplies the color value MPI_UNDEFINED, newcomm returns MPI_COMM_NULL. Even though this is a collective call, each process is allowed to provide different values for color and key.

CURRICULUM VITAE

NAME: Sa-at Muangchan. **GENDER:** Male. **NATIONALITY:** Thai.

DATE OF BIRTH: November 6, 1976. **MARITAL STATUS:** Single.

EDUCATIONAL BACKGROUND:

- B.Sc. in Mathematics, Khonkaen University, Khonkaen, Thailand, 1999.

- M.Sc. in Computational science, Chulalongkorn University, Bangkok, Thailand, 2002.

WORK EXPERIENCE:

- Lecturer in Mathematics and Statistics Department, Sakon Nakhon Rajabhat University, Sakon Nakhon, Thailand since 2002.

- Special lecturer, Department of Mathematics and Statistics, Nakhon Ratchasima Rajabhat University, Nakhon Ratchasima, Thailand, June 2004 - September 2004.

PUBLICATIONS:

- D. Yambangwai and S. Muangchan, "Parallel algorithm based on explicit and implicit splitting method for Laplace equation", The 8^{th} Annual National Symposium on Computational Science and Engineering, 2004, Suranaree University of Technology, Nakhon Ratchasima, Thailand.

- S. Muangchan and N.P. Moshkin, "Parallel numerical simulation of turbulent wake dynamics in linear stratified fluid", The 12^{th} Annual National Symposium on Computational Science and Engineering, 2008, Ubon Rajathanee University, Ubon Rajathanee, Thailand.

SCHOLARSHIPS:

- University Staff Development Program (Sakon Nakhon Rajabhat University), 2003-2006.