NUMERICAL ALGORITHMS FOR THREE DIMENSIONAL COMPUTATIONAL FLUID DYNAMIC PROBLEMS

Josué Mora Acosta

TESIS DOCTORAL

presentada al
Departamento de Máquinas i Motores Térmicos
E.T.S.E.I.T
Universidad Politécnica de Catalunya

para la obtención del grado de

Doctor Ingeniero Industrial

Terrassa, 2001

NUMERICAL ALGORITHMS FOR THREE DIMENSIONAL COMPUTATIONAL FLUID DYNAMIC PROBLEMS

Josué Mora Acosta

Directores de la Tesis

Dr. Assensi Oliva LLena
Dr. Manel Soria Guerrero

Tribunal calificador

Dr. Carlos David Pérez Segarra
Universitat Politècnica de Catalunya

Dr. Eduard Egusquiza Estévez
Universitat Politècnica de Catalunya

Dr. Jess Labarta Mancho
Universitat Politècnica de Catalunya

Dr. José Julio Guerra Macho
Universitat de Sevilla

Dr. Antonio Pascau Benito
Universitat de Zaragoza

Abstract

NUMERICAL ALGORITHMS FOR THREE DIMENSIONAL COMPUTATIONAL FLUID DYNAMIC PROBLEMS

The target of this work is to contribute to the enhancement of numerical methods for the simulation of complex thermal systems. Frequently, the factor that limits the accuracy of the simulations is the computing power: accurate simulations of complex devices require fine three-dimensional discretizations and the solution of large linear equation systems.

Their efficient solution is one of the central aspects of this work. Low-cost parallel computers, for instance, PC clusters, are used to do so. The main bottle-neck of these computers is the network, that is too slow compared with their floating-point performance.

Before considering linear solution algorithms, an overview of the mathematical models used and discretization techniques in staggered cartesian and cylindrical meshes is provided. The governing Navier-Stokes equations are solved using an implicit finite control volume method. Pressure-velocity coupling is solved with segregated approaches such as SIMPLEC.

Different algorithms for the solution of the linear equation systems are reviewed: from incomplete factorizations such as MSIP, Krylov solvers such as BiCGSTAB and GMRESR to acceleration techniques such as the Algebraic Multi Grid and the Multi Resolution Analysis with wavelets. Special attention is paid to preconditioned Krylov solvers for their application to parallel CFD problems.

The fundamentals of parallel computing in distributed memory computers as well as implementation details of these algorithms in combination with the domain decomposition method are given. Two different distributed memory computers, a Cray T3E and a PC cluster are used for several performance measures, including network throughput, performance of algebraic subroutines that affect to the overall efficiency of algorithms, and the solver performance. These measures are addressed to show the capabilities and drawbacks of parallel solvers for several processors and their partitioning configurations for a problem model.

Finally, in order to illustrate the potential of the different techniques presented, a three-dimensional CFD problem is solved using a PC cluster. The numerical results obtained are validated by comparison with other authors. The speedup up to 12 processors is measured. An analysis of the computing time shows that, as expected, most of the computational effort is due to the pressure-correction equation, here solved with BiCGSTAB. The computing time of this algorithm, for different problem sizes, is compared with Schur-Complement and Multigrid.

Acknowledgements

During the development of this dissertation at Centre Tecnològic de Transfèrencia de Calor (CTTC) I have received help and encouragement from many people which I would like to acknowledge.

- I am indebted to Prof. A.Oliva, for giving me the opportunity to work on Computational Fluid Dynamics (CFD) and directing my dissertation.
- I gratefully acknowledge to Prof. M.Soria, co-director of this work. His knowledge and experience in algebraic multigrid and parallel computing as well as many helpful discussions with him have been invaluable in the development of many parts of this work.
- I would like to thank R.Alba, for providing me the technical support, in both hardware and software sense, in the code development of numerical algorithms for CFD.
- I am grateful to K.Claramunt for his collaboration at benchmarking stage of the CFD code.
- I am also grateful to S.Buxton for suggestions and corrections of my english style.
- Many thanks to G.Colomer for helping me with the edition of this document.
- Many thanks are also due to Profs. D.Pérez Segarra and M.Costa for reading, revising and proving helpful comments.

During my stay in Japan at Industrial Institute of Science (IIS) in Tokyo, I had the benefit of learning on parallel computing of Large Eddy Simulations (LES).

- I wish to thank to Profs. T.Kobayashi and N.Taniguchi for their generosity to allow me to use their computer facilities.
- I am grateful to K. Kobayashi and T. Kogaki for their advice on parallel computing for LES.

I am grateful to the computer centers (CC) who allowed to me to use their facilities

- CESCA and CEPBA, Origin2000, HP2250, IBMSP2
- CIEMAT, Cray T3E
- CC of University of Tokyo, Hitachi SR8000

I would like to thank among many friends, Prof. M.Ishikawa and his wife for their hospitality during my stay in Japan.
Finally, I am deeply grateful to my family and specially to Mari Carmen Hoyas, for their patient, support and love through these years.

This research has been financially supported by the Generalitat de Catalunya, under grant TDOC98-00, CTTC under grant I0341 and by the ”Comisión Interministerial de Ciencia y Tecnologı́a” Spain project TIC99-0770.

Contents

List of Algorithms

Acronyms

ACM	Additive Correction Multigrid
AMG	Algebraic Multi Grid
BiCGSTAB	Bi Conjugate Gradient STABilized
CFD	Computational Fluid Dynamics
CFL	Courant Friedrich Levy condition
CG	Conjugate Gradient
CGA	Coarse Grid Approximation
CTTC	Centre Tecnologic de Transferencia de Calor
DFT	Discrete Fourier Transform
DPC	Biblioteca per al Desenvolupament de Programes aplicats a la resolucio de problemes Combinats de transferencia de calor i massa
DDACM	Domain Decomposed ACM
FVM	Finite Volum Method
GMRESR	Generalized Minimal RESidual Restarted
GS	Gauss-Seidel
HPC	High Performance Computing
HMT	Heat and Mass Transfer
ILU	Incomplete Lower Upper
JFF	Joan Francesc Fernandez PC cluster
LU	Lower Upper decomposition
MIMD	Multiple Instruction Multiple Data
MG	Multi Grid
MPI	Message Passing Interface
MPMD	Multiple Program Multiple Data
MRA	Multi Resolution Analisys
MSIP	Modified Strogly Implicit Procedure
PC	Personal Computer
SC	Schur Complement method
SIMD	Single Instruction multiple Data
SIMPLE	Semi-Implicit Method for Pressure-Linked Equations
SIP	Strongly Implicit Procedure
SIS	Strongly Implicit Solver
SPAI	SParse Approximate Inverse
SPMD	Single Program Multiple Data

Chapter 1

Introduction

1.1 Motivation

Computational Fluid Dynamics (CFD), enable us to deal with a wide range of engineering problems related with heat and mass transfer (HMT) by means of the numerical solution of the governing equations, i.e. the conservation of mass, momentum and energy.

With the help of digital computers it has been possible to solve numerically these equations and simulate engineering problems including different phenomena (from pure diffusion to natural or forced convection), flow structures (laminar and turbulent), states of flows (single or two phase flows) and geometries (from simple to complex cavities, channels, bluff bodies, etc.).

The purpose of these simultations is that they may be a laboratory of accurate numerical experiments for predicting complex phenomenologies and a supply of information to less detailed solutions for engineering problems. However, the experimental research (not done in this work) cannot be underestimated in front of the numerical experiments. These experimental cases are chosen carefully to show either the limitations or the range of application of the models implemented. Once the code has been "checked", the simulations that it provide are a tool for the design and optimization of engineering solutions.

The target of this work is to contribute to the development of numerical techniques that will allow the detailed numerical simulation of complex heat and mass transfer phenomena inside thermal equipments, such as melting for heat storage (1), heat looses due to the natural convection in solar colectors (2) and ventilated facades (3), turbulence inside the chamber or through the valves of a reciprocating compressor for refrigeration (4), thermal stratification in tanks for heat storage (5), etc.. All of them, should be simulated in detail for an improvement of their performances.

For a given equipment, the different levels of complexity of modellization could be discussed from the accuracy and feasibility points of view of the results. If a relatively simple model is used with a global conservation of mass, momentum and energy balances over the parts of the equipment, and additionally, with the help of experimental coefficients (i.e. friction factor, heat transfer coefficient in convection, void fraction, pressure-loss coefficient, etc.) we can expect to obtain, with low computational resources (i.e. memory storage and cpu time), global results. Indeed, the accuracy of results depends on the accuracy of the experimental coefficients and their suitability for such model.

Conversely, a complex and detailed model with less use of experimental coefficients would give very accurate and detailed results, but it demands large computational resources. For instance, the modellization of an element such as the reciprocating compressor is based on the integration of the governing equations in the whole compressor domain (compression chamber, valves, manifolds, mufflers, connecting tubes, parallel paths, shell, motor, etc.) featured by complex heat transfer and fluid flow phenomena: 3D turbulent, pulsatory and compressible flow, fast transient processes, complex geometries, moving surfaces, etc.

Hence, for an accurated solution of these phenomena, in order to be used in the design and optimization processes, a very fine spatial and temporal discretization of the governing equations is required, resulting into a large number of systems of equations with hundreds of thousands and even millions of unknowns. Moreover, the treatment of the high convective terms of the governing equations, the pressure-velocity coupling and the boundary conditions lead to non linear and ill-conditioned systems of equations which are more difficult to solve.

Therefore, the feasibility to solve in detail problems with sets of large systems of equations is constrained by the current technology of the computing science applied to CFD problems: large computational resources and powerful solvers.

1.2 Overview of parallel computing in CFD problems

In order to deal with a problem such as the presented above, the parallel computation on multiprocessor architectures and the numerical algorithms which exploit the capabilities of these machines offers the possibility to solve large CFD problems. To do so, current research efforts are being stressed in two areas: computational resources based on distributed memory architectures and numerical algorithms (i.e. the discretization and solution of the governing equations). It is worth noting that, the developement of high performance software follows the current software enginering philosofy: the layering or block-building technique.

1.2.1 Computer technology

The computer technology is continuously changing in order to afford the present and future of a wide number of disciplines. Few of them are work-office, entertainment (multimedia), data base and High Performance Computing (HPC). Here we are interested on the application of the emerging technologies to HPC, and more precisely, to the simulation of CFD problems.

Single-processor architecture machines, in spite of their increasing improvements in computational power, are incapable to afford the numerical simulations of our interest. For instance, to carry out a numerical simulation within a reasonable time it would require an speed of the processor over the physical limits of the current technology. Moreover, the size of the problem is too large and hence, its storage in RAM is another limiting factor. The development and progressive introduction of the operating system Linux in PCs gives the equivalent reliability and performance features to the work-stations with UNIX.

Conventional parallel computers with either shared or distributed memories and with high speeds of communication among processors, represent an important advance respect to the above machines. These machines use UNIX based operating systems. However, this computational power is tightly linked with an expensive cost.

Nowadays, it is being both a decreasing cost of personal computers (PCs) with fast processors and a rapid development of the communication technologies that improve data transfer rates in networks. As a result of this scenario, the PC clusters are an alternative to the supercomputation of distributed applications.

For similar computational power per processor, the PC clusters have a cost per processor lower than supercomputers. However, the main disadvantage of PC clusters relays on their limited capacity of communication, roughly 10 times slower than in the case of the supercomputers. Since the parallel performance is directly affected by the communication, it is very difficult to design the numerical algorithms, i.e. the parallel solvers for CFD problems.

The computer technology advances are also concerned with faster communications. Therefore, they have also to be considered for the improvement of our parallel implementations. These advances are summarized below from the hardware and software points of view.

- With respect to the hardware for networks, the advances are directed to increase the transfer rate of data per second, i.e. the bandwidth and to reduce the start-up time of the communication, i.e. the latency. Examples of network connections are Fastethernet(100Mb/sec) and Myrinet(1Gb/sec) (6). The connections are joined in the switch elements that accept transference of data from all the processors simultaneously and in both directions, i.e. in full duplex communication. Furthermore, these elements enable the clustering of several tens of processors, and hence, the increase of the potential scalability or computational power in distributed applications.

- With respect to the software for networks, the advances have been directed to the development of efficient, reliable and portable message passing libraries. For example, the Message Passing Interface (7) (MPI) is becoming an standard de facto.

1.2.2 Numerical algorithms

Current research efforts on numerical algorithms are being stressed in the development of accurate and fast solutions of CFD and HT problems by means of new discretization procedures of the governing equations and new solvers for both single and distributed memory multiprocessor architectures.

The accuracy of results is achieved by either a grid refinement on the spatial and temporal coordinates for a given discretization scheme or by an improved discretization scheme for a given grid. However, the grid refinement leads to larger systems of equations while the discretization with higher order schemes, following conservative and stability criteria, leads to more complex computational molecules, and hence, to more dense matrices. Moreover, the discretization procedure has to consider the coupling problem among the mass, momentum and energy systems of equations. Segregated and coupled discretization procedures and solvers are continuously being revised (8; 9).

Since the most computation-intensive part of a simulation is the solution of the algebraic systems of equations, the research is also focused on fast and robust solvers. These solvers are featured by an increase of efficiency and scalability to deal with difficult and large systems of equations. Furthermore, the research and development of new solvers is closely related with the sequential and parallel computers, and hence, leading to sequential and parallel solvers respectively.

Within the framework of solvers for CFD problems, current sequential solvers are based in iterative methods (e.g. the incomplete factorizations SIP (10), MSIP (11) and SIS (12; 13)) which exploit, better than direct methods (e.g. the complete factorization (14)), the sparsity of the large matrices that arise in the discretization of the governing equations.

Apart from the improvements of the iterative solvers on the reduction of both the number of floating point operations and memory storage requirements, the main improvements are obtained when combining any of them with a Multi Grid acceleration technique (15; 16) (MG) for the solution of large linear systems of equations. It is well known the degradation of the efficiency of these iterative solvers, so called smoothers in the context of MG, for these systems of equations. The major improvements are done in the development of algebraic restriction and prolongation transfer operators which lead to the Algebraic Multi Grid (AMG) (17; 18; 19) as well as in the implementation of different solvers at different grid levels (e.g. the use of a iterative solvers at finer levels plus a direct solver at the coarsest level). An emerging approach to the acceleration techniques with similar results to AMG is the Multi Resolution Analysis (MRA) (20; 21; 22) with wavelets (23). It offers a generalization of the transfer operators, and hence, a designing tool of them.

Nevertheless, the elliptic nature of the governing equations and the coupling among the fluid-flow variables constrain their fast solution with sequential solvers to not too large size problems (e.g. up to a half million of unknowns).

The strategy based on parallel solvers is the key to scale the problem size to several millions of unknowns at low computational cost time. Although the solvers mentioned above are powerful in sequential (i.e. execution in single processor architectures), the inherent sequential parts contained within the algorithm (e.g. incomplete LU factorizations) and the large number of communications disable their efficient implementation in distributed memory multiprocessor architectures. For instance, in a parallel implementation of the algebraic multigrid, several communication steps are required at each level leading to a low ratio of the time of computation with the time of communication for the coarser levels.

Furthermore, the efficient parallelization of CFD problems with large systems of equations not only depends on this ratio but also on the distribution of data (or load) among the processors (i.e. the load balancing). The addition of the domain decomposition (24) with the Krylov space based iterative methods (25) is the most widely adopted technique. However, the efficiency of this strategy is strongly dependent on the partitioning directions of the domain and on the parallel efficiency of the preconditioner for the Krylov solver. The most common preconditioners are based in incomplete factorizations (26), polynomial preconditioners (27) and approximate inverses (28; 29). However, due to the inherent sequential parts of the incomplete factorization, the efficiency decreases with the number of processors. The polynomial preconditioner parallelize well but it needs an accurate approximation of the singular values, task as difficult as to solve the system of equations. And, the approximate inverse preconditioner requires, in advance, the knowledge of the pattern of the inverse of the matrix problem.

1.2.3 Software engineering: layering

Software engineers often recommend the abstraction and encapsulation by layers in developing software components(30). They recommend the layering of new components on top of existing components, using only information about the functionality and interfaces provided by the existing components. This layering approach is in contrast to a direct implementation of new components, utilizing unencapsulated access to the representation data structures and code present in the existing components. By reusing the existing components, the layering approach intuitively result in reduced development costs, and in increased quality for the new components.

Following this idea a CFD code based on a structure of four layers may be proposed: the user layer, the solver layer, the algebra layer and the communication layer. These layers are linked and integrated to build an structure which looks like an iceberg (see Fig. 1.1).

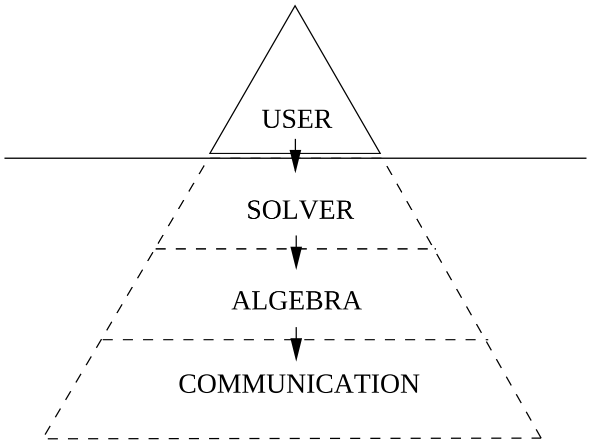

Figure 1.1: Iceberg built by layers: user layer, solver layer, algebra layer and communication layer.

The concept of the iceberg comes from the idea that only the top of the iceberg or the last layer is visible while the rest of the iceberg or layers are hidden below the water. This top end is called the user layer, the only layer visible to the end user of the CFD code. The rest of layers are hidden from the user, i.e. the user does not access to the remaining layers.

In order to build the CFD code in layers it is very important to specify clearly which are the subroutines of each layer and which layer is supported by another, i.e, to define the dependencies between layers in only one direction. This dependency goes from the top to the base as shown with the arrows in figure 1.1. The detailled description of these layers is given in appendix.

1.3 Scope of the work

This work is aimed to contribute to the development of numerical algorithms for CFD problems using both sequential and parallel computers based on clusters of commodity computers. The purpose of this work is fourfold; it is to provide with:

- The derivation of the systems of equations of a CFD model problem.
- A review of the sequential and parallel solvers for these systems.
- A parallel implementation of the domain decomposition and Krylov solvers.
- A performance study of such implementation for a benchmark problem.

These items are described in the forthcoming chapters as follows. In chapter 2, the fundamentals of fluid dynamics and heat transfer in 3D phenomena are reviewed through the governing equations (i.e. the conservation of mass, momentum and energy) and the boundary conditions. The discretization of the governing equations for both cartesian and cylindrical coordinates is based on the Finite Volume Method (FVM) on staggered grids. Numerical techniques such the discretization schemes, implementation of boundary conditions and solution procedures for incompressible and unsteady flow problems are reviewed.

The chapter 3 is concerned with the analysis and solution of these systems of equations. This analysis is focused on the matrix properties of systems of equations such the sparsity of matrices, the stability criteria associated with the matrix coefficients and the convergence behavior for different boundary conditions. On the other hand, this chapter is a review of the implementation details of several suitable solvers for these systems of equations. This review covers the basic iterative methods based on incomplete factorizations, the algebraic multigrid, the multiresolution analysis with wavelets and the Krylov space based solvers with incomplete LU factorizations as preconditioners. These implementation descriptions are completed with a set of performance tests by means of the comparison of the time of computation and the memory requirements for a sequential execution.

In chapter 4, the concepts of parallel computing in distributed memory machines and performance parameters are defined and measured for two well differentiated architectures: the Cray T3E and a PC cluster (see Appendix). In this work the MPI library has been adopted for the implementation of the exchange of data among processors. This MPI implementation is submitted to a set of performance tests where the communication and computational efficiencies (i.e. the speed-up and the scalability) of both the algebraic operations and the Krylov solvers are measured. Since these tests have been executed in both architectures, the comparison of the communication and computation performances has reported valuable information about the range of application of these solvers (i.e. the scalability and the speed-up) and the cost-effectiveness of each machine.

This is the base of the parallel implementation of the systems of equations and solvers under an algebraic implementation of the domain decomposition technique. The idea is to decompose the algebraic operations involved in all procedures by means of the distribution of vector and matrix data among the processors. The operations over data and the data follow the so called Single Program and Multiple Data (SPMD) paradigm. This paradigm suits well for distributed memory machines such as the PC clusters.

In chapter 5, several of the above techniques are applied to the solution of a CFD problem: the domain decomposition technique, the incomplete factorization solver for the solution of momentum equations and BiCGSTAB preconditioned for the solution of the pressure correction equation. The well known 3D lid driven cavity problem is chosen to firstly benchmark the accuracy of results and secondly to analyze the speed-up and the scalability of the parallel implementation for several number of processors, partitioning configurations and problem sizes. Bottlenecks within the algorithm and solution alternatives to the current implementation are also outlined.

The concluding remarks of the solution of large systems of equations in PC clusters and further trends on parallel computing of CFD problems are discussed in chapter 6.

Chapter 2

Modelization of CFD problems

2.1 Description of the governing equations

The fluid flow and heat transfer phenomena assumed throughout this work can be summarized in the following hypotheses and restrictions:

- Two or three dimensional flow structure
- Steady or unsteady flow
- Laminar flow
- Newtonian and incompressible fluid
- Constant physical properties
- Buoyancy effect modeled by the Boussinesq's hypothesis
- Neglected viscous dissipation
- Neglected radiation
- Single phase fluid
- Single component

A detailed explanation of these hypothesis can be found in any book of fundamentals of CFD (31; 32; 33). Under these hypotheses it is possible to cover a wide range of engineering applications.

The governing equations, i.e. the conservation of mass, momentum and energy equations give us the tools necessary to deal mathematically with these applications. These equations are written in differential form as:

- Conservation of mass (continuity equation)

$$\nabla \circ \vec{V} = 0$$

- Conservation of momentum (Navier-Stokes equations)

$$\frac{\partial \vec{V}}{\partial \tau} + \vec{V}\left(\nabla \vec{V}\right) = -\frac{1}{\rho}\nabla P + \nu \nabla^2 \vec{V} + \vec{g}\beta\left(T - T_0\right)$$

- Conservation of energy

$$\frac{\partial T}{\partial \tau} + \vec{V}(\nabla T) = \frac{\kappa}{\rho c_p}\nabla^2 T + \frac{S_T}{\rho c_p}$$

The solution of these equations reports a detailed information of the fluid-flow variables involved in the phenomena of study: the velocity $\vec{V} = \{u, v, w\}$, the pressure P and the temperature T.

Since the geometry of the domain of study has a strong influence in the pattern flow, it is suitable to choose the coordinate system which represent this pattern better. By doing so, generality is gained for the development of successive sections.

2.1.1 Cartesian and cylindrical coordinate systems

The governing equations are represented in two orthogonal coordinate systems: the cartesian $\{x, y, z\}$ and the cylindrical $\{r, \theta, z\}$ (see Fig. 2.1).

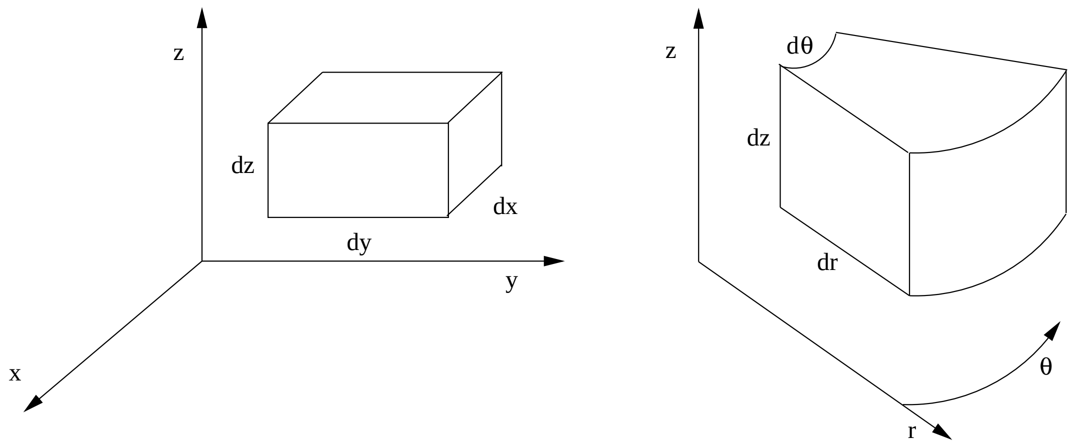

Figure 2.1: Coordinate systems: cartesian (left) and cylindrical (right).

Hence, the representation of the governing equations in the cartesian coordinate system leads to:

- Conservation of mass or continuity equation

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0$$

- Conservation of momentum in x direction

$$\frac{\partial u}{\partial \tau}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}=-\frac{1}{\rho}\frac{\partial P}{\partial x}+\nu\left[\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}\right]+g_x\beta(T-T_0)$$

- Conservation of momentum in y direction

$$\frac{\partial v}{\partial \tau}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}=-\frac{1}{\rho}\frac{\partial P}{\partial y}+\nu\left[\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 v}{\partial z^2}\right]+g_y\beta(T-T_0)$$

- Conservation of momentum in z direction

$$\frac{\partial w}{\partial \tau}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}=-\frac{1}{\rho}\frac{\partial P}{\partial z}+\nu\left[\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}+\frac{\partial^2 w}{\partial z^2}\right]+g_z\beta(T-T_0)$$

- Conservation of energy

$$\frac{\partial T}{\partial \tau}+u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y}+w\frac{\partial T}{\partial z}=\frac{\kappa}{\rho c_p}\left[\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}+\frac{\partial^2 T}{\partial z^2}\right]+S_T$$

They are all summarized in the so called convection and diffusion equation (32) for the cartesian coordinate system as:

$$\frac{\partial \phi}{\partial \tau}+\frac{\partial u\phi}{\partial x}+\frac{\partial v\phi}{\partial y}+\frac{\partial w\phi}{\partial z}=\Gamma_\phi\left[\frac{\partial^2 \phi}{\partial x^2}+\frac{\partial^2 \phi}{\partial \theta^2}+\frac{\partial^2 \phi}{\partial z^2}\right]+S_\phi$$

Where the values of ϕ, Γ_ϕ and S_ϕ are summarized in table 2.1.

Equation	ϕ	Γ_ϕ	S_ϕ
Continuity	1	0	0
Momentum in x direction	u	ν	$-\frac{1}{\rho}\frac{\partial P}{\partial x}+g_x\beta(T-T_0)$
Momentum in y direction	v	ν	$-\frac{1}{\rho}\frac{\partial P}{\partial y}+g_y\beta(T-T_0)$
Momentum in z direction	w	ν	$-\frac{1}{\rho}\frac{\partial P}{\partial z}+g_z\beta(T-T_0)$
energy	T	$\frac{\kappa}{\rho c_p}$	$\frac{S_T}{\rho c_p}$

Table 2.1: Values of ϕ, Γ_ϕ and S_ϕ for the convection and diffusion equation in a cartesian coordinate system

The representation of the governing equations for the cylindrical coordinate system leads to:

- Conservation of mass or continuity equation

$$\frac{1}{r}\frac{\partial ru}{\partial r} + \frac{1}{r}\frac{\partial v}{\partial \theta} + \frac{\partial w}{\partial z} = 0$$

- Conservation of momentum in r direction

$$\frac{\partial u}{\partial \tau} + u\frac{\partial u}{\partial r} + \frac{v}{r}\frac{\partial u}{\partial \theta} - \frac{v^2}{r} + w\frac{\partial u}{\partial z} =$$
$$-\frac{1}{\rho}\frac{\partial P}{\partial r} + \nu\left[\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial ru}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} - \frac{2}{r^2}\frac{\partial v}{\partial \theta} + \frac{\partial^2 u}{\partial z^2}\right] + g_r\beta(T - T_0)$$

- Conservation momentum in θ direction

$$\frac{\partial v}{\partial \tau} + u\frac{\partial v}{\partial r} + \frac{v}{r}\frac{\partial v}{\partial \theta} + \frac{uv}{r} + w\frac{\partial v}{\partial z} =$$
$$-\frac{1}{\rho r}\frac{\partial P}{\partial \theta} + \nu\left[\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial rv}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 v}{\partial \theta^2} + \frac{2}{r^2}\frac{\partial u}{\partial \theta} + \frac{\partial^2 v}{\partial z^2}\right] + g_\theta\beta(T - T_0)$$

- Conservation of momentum in z direction

$$\frac{\partial v}{\partial \tau} + u\frac{\partial v}{\partial r} + \frac{v}{r}\frac{\partial v}{\partial \theta} + \frac{uv}{r} + w\frac{\partial v}{\partial z} =$$
$$-\frac{1}{\rho r}\frac{\partial P}{\partial \theta} + \nu\left[\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial rv}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 v}{\partial \theta^2} + \frac{2}{r^2}\frac{\partial u}{\partial \theta} + \frac{\partial^2 v}{\partial z^2}\right] + g_z\beta(T - T_0)$$

- Conservation of energy

$$\frac{\partial T}{\partial \tau} + u\frac{\partial T}{\partial x} + \frac{v}{r}\frac{\partial T}{\partial \theta} + w\frac{\partial T}{\partial z} = \frac{\kappa}{\rho c_p}\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} + \frac{\partial^2 T}{\partial z^2}\right] + S_T$$

Analogously as in the cartesian coordinate system, they are written in the convection and diffusion equation for the cylindrical coordinate system.

$$\frac{\partial \phi}{\partial \tau} + \frac{1}{r}\frac{\partial ru\phi}{\partial r} + \frac{\partial v\phi}{\partial \theta} + \frac{\partial w\phi}{\partial z} = \Gamma_\phi\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial \phi}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 \phi}{\partial \theta^2} + \frac{\partial^2 \phi}{\partial z^2}\right] + S_\phi$$

with ϕ, Γ_ϕ and S_ϕ values summarized in table 2.2

Equation	ϕ	Γ_ϕ	S_ϕ
Continuity	1	0	0
Momentum in r direction	u	ν	$-\frac{1}{\rho}\frac{\partial P}{\partial r} + g_r\beta(T-T_0) + \frac{v^2}{r} - \nu\frac{2}{r^2}\frac{\partial v}{\partial \theta} - \nu\frac{u}{r^2}$
Momentum in θ direction	v	ν	$-\frac{1}{\rho}\frac{\partial P}{\partial y} + g_r\beta(T-T_0) - \frac{uv}{r} + \nu\frac{2}{r^2}\frac{\partial u}{\partial \theta} - \nu\frac{v}{r^2}$
Momentum in z direction	w	ν	$-\frac{1}{\rho}\frac{\partial P}{\partial z} + g_z\beta(T-T_0)$
Energy	T	$\frac{\kappa}{\rho c_p}$	$\frac{S_T}{\rho c_p}$

Table 2.2: Values of ϕ, Γ_ϕ and S_ϕ for the convection and diffusion equation in a cylindrical coordinate system

These equations are written in differential form and all together, they represent a system of partial differential equations with non linear terms and coupling between variables. It is clear that no analytical solution is feasible in a general case. Conversely, numerical methods such as finite differences, finite elements or finite volumes can handle this problem.

2.2 Discretization by finite volume method

The finite volume method is used here because it is easy to understand from the physical point of view. In this sense, the work done by Patankar (32) for the discretization of the governing equations and the linking procedure between systems of equations was adopted and followed.

Firstly, the domain is discretized into a finite number of volumes covering all domain. The grid defined by this discretization is called the centered grid. At the center of these volumes, the scalar variables P and T are evaluated. In addition three staggered grids in each direction of the coordinate system, i.e. the $\{x, y, z\}$ directions for the cartesian and the $\{r, \theta, z\}$ directions for the cylindrical, are also used for the evaluation of each component of the velocity vector $\vec{V} = \{u, v, w\}$.

The integration of these partial differential equations in the respective grid can be briefly described by means of the integration of the convection and diffusion equation for a general grid whether it is centered or staggered.

The convection and diffusion equation composed by four terms, i.e. the convection, the diffusion, the source term and the unsteady term, is integrated in a generic volume $\mathcal{V}$. Fig. 2.2 shows the geometry of the volume, as well as the faces and distances between centers of the neighbour volumes.

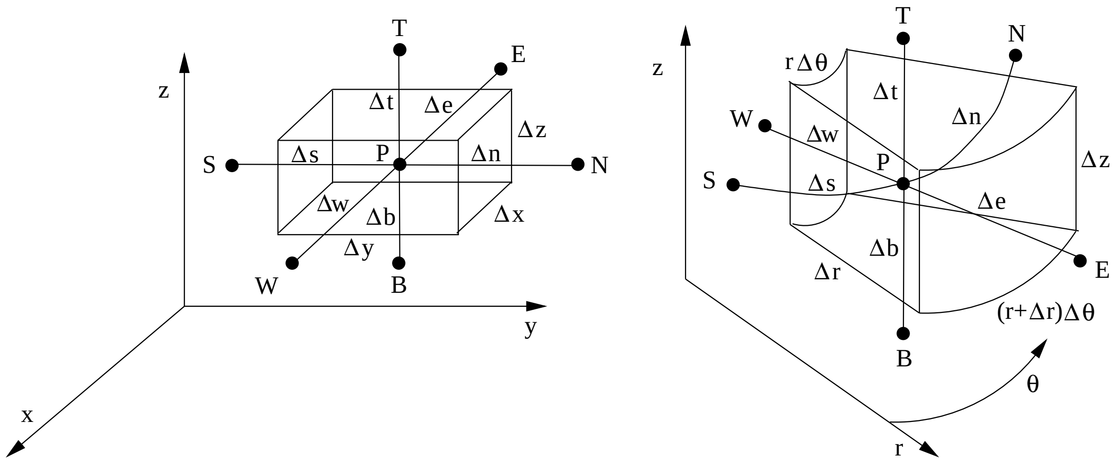

Figure 2.2: Coordinate systems: left, cartesian $\{y, y, z\}$ and right, cylindrical $\{r, \theta, z\}$.

In both pictures, it is followed the same notation. The uppercase letters $\{P, W, E, S, N, B, T\}$ represent the values of the variable ϕ at the center of volumes whilst the lowercase letters represent the values at faces $f = \{w, e, s, n, b, t\}$.

Therefore, the resulting integrated convection and diffusion equation is expressed in algebraic form as

$$\begin{array}{rcl} \frac{\rho\mathcal{V}}{\Delta\tau}(\phi_P - \phi_P^0) & + & \\ F_e\phi_e - F_w\phi_w & + & \\ F_n\phi_n - F_s\phi_s & + & \\ F_t\phi_t - F_b\phi_b & = & \\ D_e(\phi_E - \phi_P) - D_w(\phi_P - \phi_W) & + & \\ D_n(\phi_N - \phi_P) - D_s(\phi_P - \phi_S) & + & \\ D_t(\phi_T - \phi_P) - D_b(\phi_P - \phi_B) & + & S_\phi\mathcal{V} \end{array}$$

where F_f and D_f are the convective and diffusive transport coefficients respectively of the variable ϕ at face f with surface $\mathcal{S}_f$

$$F_f = (\rho u \mathcal{S})_f, \quad D_f = \left(\Gamma\frac{\mathcal{S}}{\Delta}\right)_f$$

The algebraic equation for a given volume P can be solved to find the variable ϕ at its center ϕ_P.

It is worth noting that this equation contains also the variables of its neighbour volumes ϕ_{NGB}. Values of variables at faces ϕ_f are evaluated by interpolations of second order of accuracy or using higher order schemes that consider the closest values at centered points and the Peclet number at face f.

$$Pe_f = \frac{F_f}{D_f}$$

The deferred correction (34) is one of these approaches.

The set of coefficients of the resulting algebraic equation has the following structure

$$a_{P,\phi}\phi_P + \sum_{NGB} a_{NGB,\phi}\phi_{NGB} = b_{P,phi}$$

where

$$\begin{aligned}
a_{W,\phi} &= D_w A(Pe_w) + F_w max(F_w, 0) \\
a_{E,\phi} &= D_e A(Pe_e) + F_e max(-F_e, 0) \\
a_{S,\phi} &= D_s A(Pe_s) + F_s max(F_s, 0) \\
a_{N,\phi} &= D_n A(Pe_n) + F_n max(-F_n, 0) \\
a_{B,\phi} &= D_b A(Pe_b) + F_b max(F_b, 0) \\
a_{T,\phi} &= D_t A(Pe_t) + F_t max(-F_t, 0) \\
a_{P0} &= \frac{\rho(V)}{\Delta\tau} \\
a_{P,\phi} &= -(a_{W,\phi} + a_{E,\phi} + a_{S,\phi} + a_{N,\phi} + a_{B,\phi} + a_{T,\phi}) + \rho\frac{\mathcal{V}}{\Delta\tau} \\
b_{P,\phi} &= \rho\frac{\mathcal{V}}{\Delta\tau}\phi_P^0 + S_\phi\mathcal{V}
\end{aligned}$$

where $A(Pe_f)$ is a scheme function whose argument is the Peclet number. Further details of these implementations may be found in Patankar (32) and others (33).

The integration of the convection and diffusion equation for all volumes at all domain leads to an algebraic system of equations for a single variable ϕ.

By doing so for the momentum equations at each direction and in the respective staggered grid and the energy equation in the centered grid, an amount of four algebraic systems of equations is obtained.

$$a_{P,u}u_P + \sum_{NGB} a_{NGB,u}u_{NGB} = b_{P,u}$$

$$a_{P,v}v_P + \sum_{NGB} a_{NGB,v}v_{NGB} = b_{P,v}$$

$$a_{P,w}w_P + \sum_{NGB} a_{NGB,w}w_{NGB} = b_{P,w}$$

$$a_{P,T}T_P + \sum_{NGB} a_{NGB,T}T_{NGB} = b_{P,T}$$

Where the subscript P represent a generic point of the domain but into their respective grids.

Regarding these systems, the right hand side of the first three systems of equations contains the pressure gradient, the temperature effects over the density, i.e. the Boussinesq's hypothesis, and the source term derived from the cylindrical coordinate system on their respective directions.

$$b_{P,u} = -\frac{P_E - P_P}{\Delta_e}\mathcal{V}_u + S_u\mathcal{V}_u = -c_u\left(P_E - P_P\right) + S_u\mathcal{V}_u$$

$$b_{P,v} = -\frac{P_N - P_P}{\Delta_n}\mathcal{V}_v + S_u\mathcal{V}_v = -c_v\left(P_N - P_P\right) + S_v\mathcal{V}_v$$
$$b_{P,w} = -\frac{P_T - P_P}{\Delta_t}\mathcal{V}_w + S_u\mathcal{V}_w = -c_w\left(P_T - P_P\right) + S_w\mathcal{V}_w$$

Where $\mathcal{V}_u$, $\mathcal{V}_v$ and $\mathcal{V}_w$ are the volumes for each staggered direction.

For the resulting four systems of equations, it is said to have a coupling between the fluid-flow variables $\vec{V} = \{u, v, w\}$, T and P. The first three systems give the velocity components under the assumption of a pressure field, whilst the last one gives the temperature field. Therefore, the complete solution of the problem, i.e. the solution of the pressure P, would require an additional system of equations.

2.2.1 The SIMPLE-like algorithms

For the above three systems of equations that represent the momentum equations in each direction, a field of pressures P^* has to be guessed in order to solve the velocities say $\vec{V}^*$. This is written as:

$$a_{P,u}u_P^* + \sum_{NGB} a_{NGB,u}u_{NGB}^* = -c_u\left(P_E^* - P_P^*\right) + S_u\mathcal{V}_u$$
$$a_{P,v}v_P^* + \sum_{NGB} a_{NGB,v}v_{NGB}^* = -c_v\left(P_N^* - P_P^*\right) + S_u\mathcal{V}_v$$
$$a_{P,w}w_P^* + \sum_{NGB} a_{NGB,w}w_{NGB}^* = -c_w\left(P_T^* - P_P^*\right) + S_u\mathcal{V}_w$$

Since the velocities evaluated with momentum equations satisfy only the momentum, there is no guaranty about continuity. In this sense, the continuity equation serves to introduce the correction values of velocities $\vec{V}'$.

$$\vec{V} = \vec{V}^* + \vec{V}'$$

Moreover, it is necessary to evaluate the right value of the pressure P. This is carried out by the addition of a pressure correction P'.

$$P = P^* + P'$$

If the correction of the velocities $\vec{V}'$ is set in function of a pressure correction rate $\Delta P'$ such that

$$u_P = u_P^* + u_P' = u_P^* - d_u\left(P_E' - P_P'\right)$$
$$u_P = u_P^* + u_P' = v_P^* - d_v\left(P_N' - P_P'\right)$$
$$w_P = w_P^* + w_P' = w_P^* - d_w\left(P_T' - P_P'\right)$$

The substitution of these expressions in the respective algebraic systems of equations under the assumption that the $\vec{V}^*$ values satisfy each equation, the following systems of equations in $\vec{V}'$ and P' are obtained.

$$a_{P,u}u_P^{'} + \sum_{NGB} a_{NGB,u}u_{NGB}^{'} = -d_u\left(P_E^{'} - P_P^{'}\right)$$

$$a_{P,v}v_P^{'} + \sum_{NGB} a_{NGB,v}v_{NGB}^{'} = -d_v\left(P_N^{'} - P_P^{'}\right)$$
$$a_{P,w}w_P^{'} + \sum_{NGB} a_{NGB,w}w_{NGB}^{'} = -d_w\left(P_T^{'} - P_P^{'}\right)$$

For a first approach of d_u, d_v and d_w the neighbour summation of each system can be neglected yielding to

$$d_u = -\frac{c_u}{a_{P,u}}, \quad d_v = -\frac{c_v}{a_{P,v}}, \quad d_w = -\frac{c_w}{a_{P,w}}$$

This approach has been named SIMPLE (32). A much better improvement is SIMPLEC (35) wich considers the neighbour summation.

$$d_u = -\frac{c_u}{a_{P,u} - \sum_{NGB} a_{NGB,u}}, \quad d_v = -\frac{c_v}{a_{P,v} - \sum_{NGB} a_{NGB,v}}, \quad d_w = -\frac{c_w}{a_{P,w} - \sum_{NGB} a_{NGB,w}}$$

Finally, the values of pressure corrections P' are obtained from the continuity equation by substitution of the velocities $\vec{V}$ by the guessed velocities $\vec{V}^*$ plus the corrections $\vec{V}'$ expressed in terms of the pressure corrections.

Integrating the convection and diffusion equation for $\phi = 0$ there is the continuity equation:

$$\rho u_e \mathcal{S}_e - \rho u_w \mathcal{S}_w + \rho v_n \mathcal{S}_n - \rho v_s \mathcal{S}_s + \rho w_t \mathcal{S}_t - \rho w_b \mathcal{S}_b = 0$$

where the velocities at faces are

$$u_e = u_P^*|_e - d_u|_e\left(P_E' - P_P'\right) = u_e^* - d_u|_e\left(P_E' - P_P'\right)$$
$$v_n = v_P^*|_n - d_v|_n\left(P_N' - P_P'\right) = v_n^* - d_v|_n\left(P_N' - P_P'\right)$$
$$w_t = w_P^*|_t - d_w|_t\left(P_T' - P_P'\right) = w_t^* - d_w|_t\left(P_T' - P_P'\right)$$

Substituting these velocities in the continuity equation leads to

$$\begin{array}{rcl}
\rho\left(u_e^* - d_u|_e\left(P_E' - P_P'\right)\right)\mathcal{S}_e & - & \rho\left(u_w^* - d_u|_w\left(P_P' - P_W'\right)\right)\mathcal{S}_w + \\
\rho\left(v_n^* - d_v|_n\left(P_N' - P_P'\right)\right)\mathcal{S}_n & - & \rho\left(v_s^* - d_v|_s\left(P_P' - P_S'\right)\right)\mathcal{S}_s + \\
\rho\left(w_t^* - d_w|_t\left(P_T' - P_P'\right)\right)\mathcal{S}_t & - & \rho\left(w_b^* - d_w|_b\left(P_P' - P_B'\right)\right)\mathcal{S}_b = 0
\end{array}$$

The arrangement of terms of $\vec{V}^*$ to the right hand side gives the so called pressure correction equation.

$$a_{P,P'}P_P' + \sum_{NGB} a_{NGB,P'}P_{NGB}' = b_{P'}$$

Where

$$\begin{array}{rcl}
a_{W,P'} & = & D_w A(Pe_w) + max(F_w, 0) \\
a_{E,P'} & = & D_e A(Pe_e) + max(-F_e, 0) \\
a_{S,P'} & = & D_s A(Pe_s) + max(F_s, 0) \\
a_{N,P'} & = & D_n A(Pe_n) + max(-F_n, 0) \\
a_{B,P'} & = & D_b A(Pe_b) + max(F_b, 0) \\
a_{T,P'} & = & D_t A(Pe_t) + max(-F_t, 0) \\
a_{P,P'} & = & -(a_{W,P'} + a_{E,P'} + a_{S,P'} + a_{N,P'} + a_{B,P'} + a_{T,P'}) \\
b_{P'} & = & \rho u_w^* \mathcal{S}_w - \rho u_e^* \mathcal{S}_e + \rho v_s^* \mathcal{S}_s - \rho v_n^* \mathcal{S}_n + \rho w_b^* \mathcal{S}_b - \rho w_t^* \mathcal{S}_t
\end{array}$$

Solving this system of algebraic equations the desired map of pressure corrections P' is obtained. With it, the pressure and velocities are corrected satisfying both criteria momentum and continuity.

Finally, the system of equations for the temperature is solved with the corrected velocities. By doing so iteratively, the coupling between all variables $\vec{V}$, P and T is achieved. The coupling of the segregated systems is summarized in the so called SIMPLE-like algorithm (see Alg. 1).

Algorithm 1: SIMPLE-like

```
start with $k = 0$
    $\vec{V}^{(k)}, P^{(k)}, T^{(k)}$
do
    new iteration $k = k + 1$
    guess fluid flow variables
        $\vec{V}^* = \vec{V}^{(k-1)},\ P^* = P^{(k-1)},\ T^* = T^{(k-1)}$
    evaluate coefficients of Navier-Stokes equation using: $\vec{V}^*,\ P^*,\ T^*$
    solve $a_{P,u}u_P^{(k)} + \sum_{NGB} a_{NGB,u}u_{NGB}^{(k)} = b_{P,u}$
    solve $a_{P,v}v_P^{(k)} + \sum_{NGB} a_{NGB,v}v_{NGB}^{(k)} = b_{P,v}$
    solve $a_{P,w}w_P^{(k)} + \sum_{NGB} a_{NGB,w}w_{NGB}^{(k)} = b_{P,w}$
    evaluate coefficients of continuity equation using: $\vec{V}^{(k)}$
    solve $a_{P,P'}P_P^{'(k)} + \sum_{NGB} a_{NGB,P'}P_{NGB}^{'(k)} = b_{P,P'}$
    correct the velocity and the pressure
        $\vec{V}^{(k)} = \vec{V}^{(k)} + \alpha_V \vec{V}'^{(k)}$
        $P^{(k)} = P^{(k)} + \alpha_P P'^{(k)}$
    evaluate coefficients of energy equation using: $\vec{V}^{(k)},\ T^*$
    solve $a_{P,T}T_P^{(k)} + \sum_{NGB} a_{NGB,T}T_{NGB}^{(k)} = b_{P,T}$
until (mass, momentum and energy conservation)
```

Notice that the velocities and pressures are underrelaxed with α_V, α_P for convergence at time of the correction step. Typical values of these parameters are $\alpha_V = 0.5$ and $\alpha_P = 0.8$.

2.2.2 Time marching algorithm

The algorithm outlined below enable us to evaluate all variables for a given instant of time τ, say $\vec{V}^\tau$, P^τ and T^τ under the initial conditions at previous time step $\tau - \Delta\tau$. This algorithm so called time marching (32) is carried out advancing in time until arriving at the steady state of all variables.

This algorithm completes the full simulation from an initial state which originates an unsteady state and the evolution to a final steady state. The full algorithm (see Alg. 2) including the time marching and coupling of systems is written down.

Algorithm 2: SIMPLE-like plus time marching

```
start with $\tau = 0$
    $\vec{V}^{\tau},\ P^{\tau},\ T^{\tau}$
do
    new time step $\tau = \tau + \Delta\tau$
    start with $k = 0$
        $\vec{V}^{(k)} = \vec{V}^{\tau-\Delta\tau},\ P^{(k)} = P^{\tau-\Delta\tau},\ T^{(k)} = T^{\tau-\Delta\tau}$
    do
        new iteration $k = k + 1$
        guess fluid flow variables
            $\vec{V}^{*} = \vec{V}^{(k-1)},\ P^{*} = P^{(k-1)},\ T^{*} = T^{(k-1)}$
        evaluate coefficients of Navier-Stokes equations using: $\vec{V}^{*},\ P^{*},\ T^{*}$
        solve $a_{P,u}u_P^{(k)} + \sum_{NGB} a_{NGB,u}u_{NGB}^{(k)} = b_{P,u}$
        solve $a_{P,v}v_P^{(k)} + \sum_{NGB} a_{NGB,v}v_{NGB}^{(k)} = b_{P,v}$
        solve $a_{P,w}w_P^{(k)} + \sum_{NGB} a_{NGB,w}w_{NGB}^{(k)} = b_{P,w}$
        evaluate coefficients of continuity equation using: $\vec{V}^{(k)}$
        solve $a_{P,P'}P_P^{'(k)} + \sum_{NGB} a_{NGB,P'}^{(k)}P_{NGB}^{'(k)} = b_{P,P'}$
        correct the velocity and the pressure
            $\vec{V}^{(k)} = \vec{V}^{(k)} + \alpha_V \vec{V}'^{(k)}$
            $P^{(k)} = P^{(k)} + \alpha_P P'^{(k)}$
        evaluate coefficients of energy equation using: $\vec{V}^{(k)},\ T^{*}$
        solve $a_{P,T}^{(k)}T_P^{(k)} + \sum_{NGB} a_{NGB,T}^{(k)}T_{NGB}^{(k)} = b_{P,T}^{(k)}$
    until (mass, momentum and energy conservation)
    $\vec{V}^{\tau} = \vec{V}^{(k)},\ P^{\tau} = P^{(k)},\ T^{\tau} = T^{(k)}$
until (steady state of all variables)
```

2.2.3 Calculation of the time step

In previous sections a spatial $\Delta x, \Delta y, \Delta z$ and temporal $\Delta\tau$ discretizations of the CFD problem were supposed. Since the formulation is fully implicit, there is no stability criterion that needs to be met in determining the time step $\Delta\tau$. However, in order to model the transient phenomena properly, it is necessary to set $\Delta\tau$ at least one order of magnitude smaller than the smallest time constant in the system being modeled. A good way to judge the choice of $\Delta\tau$ is to observe the number of iterations SIMPLE-like algorithm needs to converge at each time step. The ideal number of iterations per time step is 10-20. If SIMPLE-like algorithm needs substantially more, the time step is too large. If SIMPLE-like algorithm needs only a few iterations per time step, $\Delta\tau$ may be increased. Frequently a time-dependent problem has a very fast startup transient that decays rapidly. It is thus often wise to choose a conservatively small $\Delta\tau$ for the first 5-10 time steps. $\Delta\tau$ may then be gradually increased as the calculation proceeds.

At glance, a conservative approach (36) to $\Delta\tau$ for implicit methods is obtained from an explicit criteria based on the Courant-Friedrichs-Levy condition (CFL) and applied to all ijk point of the domain:

$$\Delta_t \leq min_{ijk}\left(\left[\frac{|u|}{\Delta x}+\frac{|v|}{\Delta y}+\frac{|w|}{\Delta z}+2\nu\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}+\frac{1}{\Delta z^2}\right)\right]^{-1}\right)$$

The physical meaning underlying this expression is that, the time step $\Delta\tau$ has to be small enough to take account of variations of fluid flow due to the convection and diffusion effects produced in small regions of dimension $\mathcal{V} = \Delta x \times \Delta y \times \Delta z$.

2.3 Boundary conditions

In previous section, the time marching algorithm and the set of initial conditions were described and they closed the problem in the temporal direction. With respect to the spatial directions, the problem is constrained by the boundary conditions. Due to the importance of the discretization of the boundary conditions this section is reserved.

2.3.1 Dirichlet and Neumann conditions

The first kind of boundary conditions to be treated is the Dirichlet boundary condition. It sets a prescribed value $\bar{\phi}$ to the variable ϕ at fixed points P of the domain.

$$\phi|_P = \bar{\phi}$$

An example of this condition for viscous flows is the non-slip condition $\vec{V}_{wall} = 0$ that appears in wall bounded viscous flows (see Fig. 2.3).

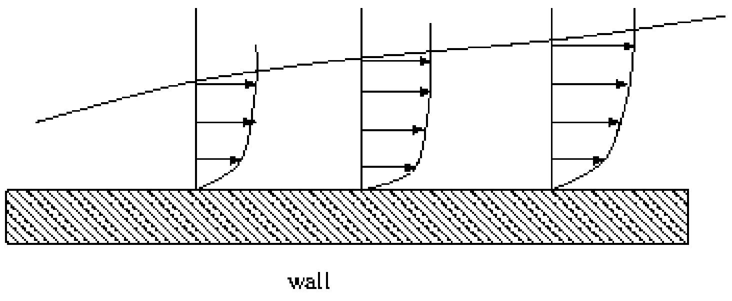

Figure 2.3: Velocity profiles and non-slip boundary condition at wall due to the viscous effects.

This condition is added to the rest of equations of momentum in their respective directions as

$$u|_{wall} = 0, \quad v|_{wall} = 0, \quad w|_{wall} = 0$$

If the problem has inlets the value of the velocity at these entrances is prescribed with either a fixed value or a velocity profile. For thermal problems, some regions of the domain can be fixed at a prescribed distribution of temperatures.

$$T|_{wall} = \bar{T}$$

In general, the Dirichlet condition is translated into an algebraic expression and included within the rest of the algebraic equations in order to close the system. The set of coefficients that represents this condition are

$$a_{P,\phi} = 1, \quad b_{P,\phi} = \bar{\phi}, \quad a_{NGB,\phi} = 0$$

The second kind of boundary condition is the Neumann boundary condition. It is associated to the numerical treatment of the phenomena where the derivative of the variable ϕ is prescribed. Let P be a point of the boundary of the domain, the prescription with a value $\bar{q}_\phi$ is written as

$$\left.\frac{\partial \phi}{\partial x}\right|_P = \bar{q}_\phi$$

For example, the free-slip condition or the null derivative of velocities should be fixed at outlets, where the structure of the flow remains constant downstream. For instance, a fully developed channel flow should consider this condition written as

$$\left.\frac{\partial \vec{V}}{\partial x}\right|_{outlet} = 0$$

For wall bounded domains and due to the non-slip condition, a zero pressure gradient at the pressure correction equation is set.

$$\left.\frac{\partial P}{\partial x}\right|_W = 0 \ , \ \left.\frac{\partial P}{\partial x}\right|_E = 0$$

$$\left.\frac{\partial P}{\partial y}\right|_S = 0 \ , \ \left.\frac{\partial P}{\partial y}\right|_N = 0$$

$$\left.\frac{\partial P}{\partial z}\right|_B = 0 \ , \ \left.\frac{\partial P}{\partial z}\right|_T = 0$$

For thermal problems, Neumann conditions describe heat fluxes (e.g. adiabatic walls).

$$\left.\frac{\partial T}{\partial x}\right|_W = \bar{q_T}$$

See Fig. 2.4 for a schematic representation of the points involved in the discretization of the adiabatic wall condition.

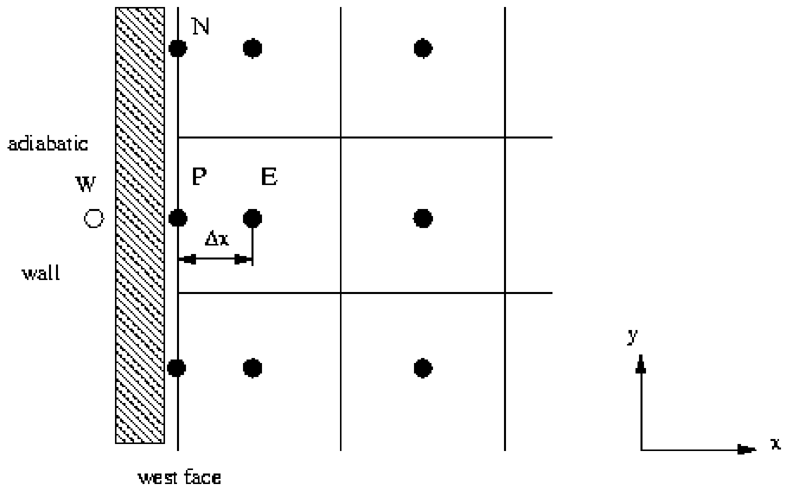

Figure 2.4: Adiabatic wall at west face of a domain.

Analogously to the Dirichlet condition, the Neumann condition is translated to an algebraic expression and included within the rest of algebraic equations and closing the system. For instance, let us to specify an adiabatic wall condition in the west face of a square box domain. The adiabatic wall condition is mathematically expressed as:

$$-\Gamma \frac{\partial T}{\partial x}\bigg|_w = 0$$

Using the Taylor's series the derivative is expanded and truncated to the first term.

$$-\Gamma \frac{\partial T}{\partial x}\bigg|_w = -\Gamma \frac{T_E - T_P}{\Delta x} = 0$$

And rearranging terms it yields

$$a_{P,T} = 1, \quad a_{E,T} = -1, \quad b_{P,T} = 0, \quad a_{NGB,T} = 0$$

Discretization of these boundary conditions with higher accuracies (37) are out of scope of this work.

2.3.2 Periodic condition

For those problems where the discretization is done in the cylindrical coordinate system, there appears a spatial periodicity condition in the angular direction θ. That implies a connection between the variables at the beginning and end of the angular coordinate. Fig. 2.5 shows graphically this connection.

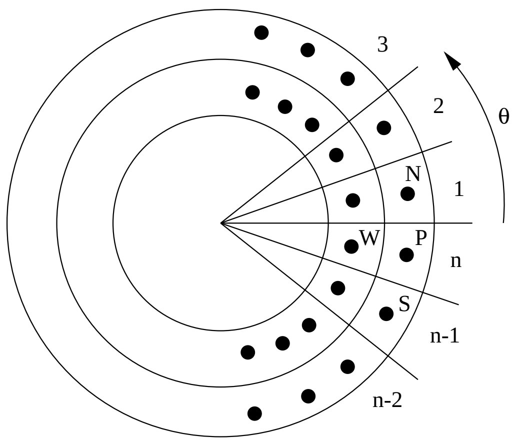

Figure 2.5: Periodicity in the angular direction θ.

Where

$$\phi_{N,n} = \phi_{P,n+1} = \phi_{P,1}$$

$$\phi_{S,1} = \phi_{P,0} = \phi_{P,n}$$

There are two ways to implement these conditions and to close the system of equations. The first one is simply adding explicitly the periodicity by including the two previous expressions in the system as follows:

$$\theta = n+1, \quad a_{P,\phi} = 1, \quad b_{P,\phi} = \phi_{P,1}, \quad a_{NGB,\phi} = 0$$

$$\theta = 0, \quad a_{P,\phi} = 1, \quad b_{P,\phi} = \phi_{P,n}, \quad a_{NGB,\phi} = 0$$

A similar treatment (38) is also used in cartesian coordinate systems for periodic and antiperiodic boundary conditions where an inlet flow is given by the outlet flow.

The second one is considering implicitly the periodicity in the solver and then no additional algebraic equations are needed. This implicit treatment of the periodicity will be discussed in the next chapter.

2.4 Stopping criteria for a simulation

An important issue in a simulation is the stopping criteria. The stopping criteria have mainly to cover two issues: the coupling and the transition to the steady state.

The first one is the coupling between the velocity field $\vec{V}$, the pressure P and the temperature T at each time step. For this purpose, after the correction of velocities (mass balance satisfied), the balance of momentum of each component of the velocity field and the balance of energy must be guaranteed.

The measure of the conservation of the momentum and energy is summarized into a global value by joining all balances previously normalized and non dimensionalized. The normalization adopted in this work is based on the 2-norm $||\cdot||_2$ of the residual $r^{(k)}$ at a given iteration k of each algebraic system.

Rewriting each algebraic systems of equations for their respective variable $\phi = \{u, v, w, T\}$ for a given k iteration, there are:

$$
\begin{aligned}
A_u^{(k)} u^{(k)} = b_u^{(k)} &\longrightarrow r_u^{(k)} = b_u^{(k)} - A_u^{(k)} u^{(k)} \\
A_v^{(k)} v^{(k)} = b_v^{(k)} &\longrightarrow r_v^{(k)} = b_v^{(k)} - A_v^{(k)} v^{(k)} \\
A_w^{(k)} w^{(k)} = b_w^{(k)} &\longrightarrow r_w^{(k)} = b_w^{(k)} - A_w^{(k)} w^{(k)} \\
A_T^{(k)} T^{(k)} = b_T^{(k)} &\longrightarrow r_T^{(k)} = b_T^{(k)} - A_T^{(k)} T^{(k)}
\end{aligned}
$$

where $A_\phi^{(k)}$ is the matrix of coefficients, $\phi^{(k)}$ the variable expressed as a vector and the right hand side $b_\phi^{(k)}$ another vector, all of them at iteration k.

The normalization and nondimesionalization, say $coupling_\phi^{(k)}$, is evaluated by

$$
coupling_\phi^{(k)} = \frac{||r_\phi^{(k)}||_2}{||b_\phi^{(k)}||_2}
$$

The addition of all of these quantities is the overall criterion of coupling of the systems for a given time step t.

$$
coupling^{(k)} = coupling_u^{(k)} + coupling_v^{(k)} + coupling_w^{(k)} + coupling_T^{(k)}
$$

Another coupling criterion based on the mass balance may be used rather than the described above. The reason is that the coupling among systems is strongly affected by the velocities and pressure. Hence the guarantee of the continuity (after the correction step) for each volume and in overall is an enough criterion to ensure the coupling among all variables.

The normalized mass balance based coupling criterion may be written as

$$
coupling^{(k)} = \frac{\sqrt{\sum_{ijk} (\rho u_w^{(k)} \mathcal{S}_w - \rho u_e^{(k)} \mathcal{S}_e + \rho v_s^{(k)} \mathcal{S}_s - \rho v_n^{(k)} \mathcal{S}_n + \rho w_b^{(k)} \mathcal{S}_b - \rho w_t^{(k)} \mathcal{S}_t)^2}}{\dfrac{\sum_{ijk} \rho \mathcal{V}}{\Delta \tau}}
$$

The second issue is the transition of the fluid flow from the unsteady state to the steady state. After the steady state there is no need to continue evaluating the variables at new time steps. Therefore a simply comparison of each variable in two close time steps is enough to decide if the steady state is achieved. For a given variable ϕ evaluated at time step τ the steady state $steady_\phi^t$ is measured as follows:

$$
steady_\phi^\tau = \frac{\sum_{ijk} |\phi^\tau - \phi^{\tau - \Delta\tau}|}{\sum_{ijk} |\phi^\tau|}
$$

where $|\cdot|$ is the absolute value of the quantity between bars.

Since the steady state of the fluid flow is achieved when all variables have arrived at the steady state, all measures in a single value are joined:

$$
steady^\tau = steady_u^\tau + steady_v^\tau + steady_w^\tau + steady_T^\tau
$$

Once evaluate both measures $coupling^{(k)}$ and $steady^{\tau}$ the stopping criteria for numerical simulations is fixed in the following values:

$$coupling^{(k)} \leq 10^{-6}, \quad steady^{\tau} \leq 10^{-3}$$

These values have been chosen experimentally in order to guarantee the numerical coupling of variables at each time step and an estimated situation of the steady state. Higher values of the coupling criterion may not ensure convergence of the pressure correction system or produce inaccurate field solutions. And higher values of the steady criterion may stop a slow fluid flow transition, for example in natural convection phenomena. Conversely, smaller values of the steady criterion may never stop the algorithm. The numerical perturbations of the algorithm and solvers masquerade the steady fluid flow state, and hence, the sensibility of the criterion cannot detect that the simulation has already reached the steady state.

2.5 Nomenclature

A	discretization matrix
a	coeff. in A
b	discretization right hand side
CV	control volume
c_p	specific heat
D	diffusion coefficient
d	coeff. of the pressure diff. term
F	flow rate through the CV face
f	general face
$\vec{g}$	gravitational force
P	pressure
Pe	Peclet number
q	net flux
r	residual, radial coordinate
S	general source term
T	temperature
$\vec{V}$	fluid flow velocity vector
$\{u, v, w\}$	fluid flow velocity components
$\{x, y, z\}$	cartesian coordinates
$\{r, \theta, z\}$	cylindrical coordinates

Greek symbols

β	thermal volumetric expansion coeff.
Δ	general width of a CV
Δr	r-direction width of CV
Δx	x-direction width of CV
Δy	similar to Δx
Δz	similar to Δx
$\Delta \tau$	time step
$\Delta \theta$	similar to Δx
Γ	general diffusion coeff.
κ	thermal conductivity
ϵ	precission
μ	dinamic viscosity
ν	kinematic viscosity
ρ	density
ϕ	general dependent variable
τ	time
θ	angular coordinate

Other symbols

$\mathcal{S}$	surface of CV
$\mathcal{V}$	volume of CV

Subscripts

B	neighbour at the bottom
b	CV face between P and B
E	neighbour at the east side
e	CV face between P and E
N	neighbour on the north side
NGB	general neighbour grid point
n	CV face between P and N
P	central grid point
S	neighbour on the south side
s	CV face between P and S
T	neighbour at the top
t	CV face between P and T
W	neighbour at the west side
w	CV face between P and W

Superscripts

(k)	k-th iteration
τ	time value
$*$	guessed value
$'$	correction value
-	prescribed value

Chapter 3

Linear solvers

3.1 Features of equation systems

In previous chapter, the algebraic equation systems arising from both Navier-Stokes and mass equations were derived. It is to be stressed that each of these systems, in spite of their coupling, is solved separately with the SIMPLE-like algorithm (32; 35). The Navier-Stokes system of equations solves the velocities $\vec{V} = \{u, v, w\}$ with the assumption of the known pressure P^* and temperature T fields. Then, the pressure correction system of equations solves P' with the guessed velocity field just evaluated in the previous step. Once the velocities and pressures are corrected, it is the turn of the energy equation, where the temperature T is obtained. Following this iterative procedure it is possible to uncouple the equation systems and focus efforts on each of these systems.

Algorithms (8; 39; 9) which consider directly the coupling between velocity and pressure fields within a single system of equations are not considered in this work, but most of the ideas that will be exposed throughout this chapter can be extended to the coupled systems.

It is well known that Navier-Stokes equations, once discretized and linearized, give a set of algebraic equation systems with non symmetric coefficients in their matrices. Meanwhile the continuity equation leads to a linear system of equations with symmetric coefficient matrix. This fact should be in mind when a solver is used which one suits for symmetric matrices or suits for non symmetric matrices. It is to be mentioned that a symmetric matrix can be considered as a special case of non symmetric matrix from the solver point of view. Thus a solver which treats with non symmetric matrices deals also with the symmetric matrices. Nevertheless, although such kind of solvers are robust and feasible for any linear system of equations, the efficiency in symmetric matrices decreases. For instance, for symmetric matrices the CG solver is faster than the robust BiCGSTAB solver, which performs double number of operations per iteration. Therefore, it is suggested to use a solver for non symmetric matrices and another one for symmetric matrices instead of a general and robust solver which is less efficient for symmetric matrices. Further guidelines on suitable solvers for symmetric and non symmetric matrices may be found in (40).

In addition, any algebraic system includes the set of equations derived from the discretization of the boundary and initial conditions. Typical boundary conditions are non-slip conditions, free-slip conditions, convective conditions and periodic conditions for Navier-Stokes equations (see details in chapter 2). And the pressure gradient is fixed to a value in the pressure correction equation . The initial conditions give the starting point of all variables at time $\tau = 0$. They all close the problem in order to supply only one solution at each successive time step $\tau + \Delta\tau$ (i.e. the linear system with the boundary conditions and initial conditions included has a non singular matrix).

Furthermore, some of these conditions, say strong conditions, enhance the convergence and offer stability to any solver. While the non-slip and initial conditions shall consider strong sets of equations, the free-slip, fixed gradients, convective and periodic boundary conditions are considered weak sets of equations. For instance, the pressure correction system of equations plus zero gradient in boundaries appear in problems with bounded domains. This system of equations is weakly closed. Since the matrix turns into singular, there are an infinite number of shifted solutions. Luckily, if one point is fixed to a constant value during all the procedure it is enough to get a single solution. Fixing a point means replacing one of the equations of the system in order to obtain a non singular matrix. By doing so, a shifted solution is obtained which matches with the pressure correction system of equations.

Another example is a repeated flow pattern in some directions which is treated with periodicity in these directions. So in these cases all boundary conditions are weak, and it is necessary to fix a value in one point.

A simple example with either double periodicity or free gradient in a two dimensional case shows an infinitive number of shifted solutions if no additional information (i.e. fixed point) is given. The partial differential equation and the boundary conditions are

$$\frac{\partial^2 \phi}{\partial x_i x_i} = b,\ \ i = \{1, 2\},\ \ x_i \in [0, 2\pi]$$

$$b = cos(x_1)sin(x_2)$$

See Fig. 3.1 for different shifted solutions ϕ which match the partial differential equation.

As mentioned before, equation systems with strong or weak sets of boundary conditions have different behaviours when attempts are made to solve them. In addition, these systems are affected by other important factors: the stability and sensitivity of the problem of perturbations. Since neither problem data nor computer arithmetic is exact the solution will not be exact. How small changes in the system of equations affects the final solution is a critical point in the discretization and the design of a robust or a stable solver.

From the point of view of the discretization process, stability is ensured if Scarborough's criterion (32) is satisfied. Or in other words, the coefficients of the matrix A are diagonally dominant. Then, once the discretization is done, the sensitivity is dealt in consideration to the solver. Let $Ax = b$ be a given system of equations, the sensitivity of this system is expressed mathematically as

$$||A + \delta A||_2 ||x + \delta x||_2 = ||b + \delta b||_2$$

$Ax = b$ is a well-conditioned system if for small changes or perturbations in the system (i.e. the matrix $||\delta A||_2$ or the right hand side $||\delta x||_2$) the changes in the solution $||\delta x||_2$ are equal or smaller than the perturbations.

$$||\delta A||_2, ||\delta b||_2 \geq ||\delta x||_2$$

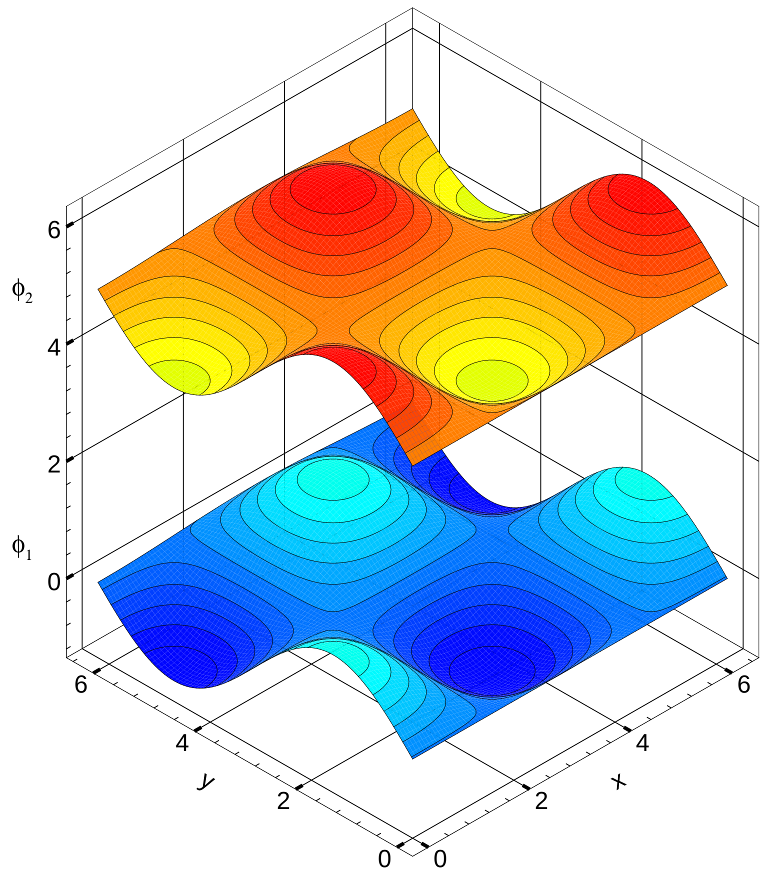

Figure 3.1: Two shifted solutions of the 2D problem with periodic boundary conditions.

Conversely, if small perturbations produce large changes in the solution, then it is said that the system is ill-conditioned.

$$||\delta A||_2, ||\delta b||_2 < ||\delta x||_2$$

It is easy to imagine what it means when solving a linearized system with an iterative procedure. Coefficients of matrix and right hand side term are updated at each iteration in function of values of an approximate solution. Then the system is solved again for a new iteration obtaining a different solution from the previous one and so on. For an ill-conditioned system, the differences between successive solutions increase at each iteration, i.e. the solution diverge.

This fact appears in both Navier-Stokes and pressure correction equation systems. From Navier Stokes system the convective term is linearized at each iteration of the global procedure (see SIMPLE-like algorithm in previous chapter). For high Reynold's numbers, this term dominate over the rest of the terms in the set of equations. Similar situation happens in the pressure correction system of equations for an incompressible fluid. Although the coefficients of the matrix remain constant during the overall algorithm the right hand side becomes very sensitive to small variations of the divergence of mass.

Since these equation systems are very sensitive to changes in solutions one simple way to prevent the divergence is based on the underrelaxation of solutions.

$$x_{new}^{(k+1)} = \alpha x_{old}^{(k+1)} + (1-\alpha)x^{(k)}$$

Where α is the relaxation parameter, a scalar value within the range $[0, 1]$.

Another more complicate way (41) but effective is based on the measure of how well or ill-conditioned is the matrix problem. That is to the condition number of the matrix $\kappa(A)$. A high value of the condition number points out an ill-conditioned matrix, while a low value indicates a well-conditioned matrix. Therefore, if one changes the problem for another with the same solution, say $\bar{A}x = \bar{b}$, but in addition, it has a reduced condition number

$$\kappa(\bar{A}) < \kappa(A)$$

then this problem would be easier to solve. Details of such technique (the so called preconditioning) are explained later on in this chapter.

3.1.1 Sparse matrix formats

The equation systems derived from CFD problems have been described mathematically. Now is time to have a look, from the computational point of view. How to handle and solve such equation systems is the principal argument of this work.

Since these equation systems contain several thousands (even millions) of unknowns in three dimensional cases with only a few set of unknowns linked per equation, clearly there are large matrices which many of the coefficients are zero. For practical reasons, a matrix is considered sparse if there is an advantage in exploiting the zeros by saving enormous computational storage and time of operations (14) (some operations are avoided where zero values are involved).

Due to the structured grid of points over the domain, they were ordered with a lexicographic or *natural* order. Such organization of unknowns that is *ijk* or any permutation of index like *jki*, *kij*,... gives a matrix with a constant set of diagonals. A number of diagonals depends on the formulation (schemes of discretization) and the dimension of the problem (two or three dimensional). For two dimensional problems, there arises *5-points* and *9-points* formulations while for three dimensional problems there are *7-points* and *19-points* formulations (42). The distribution of non zero coefficients (within some example of matrices) is shown in Figs. 3.2 and 3.3. Therefore it seems convenient and enough to store the sets of non zero coefficients by diagonals.

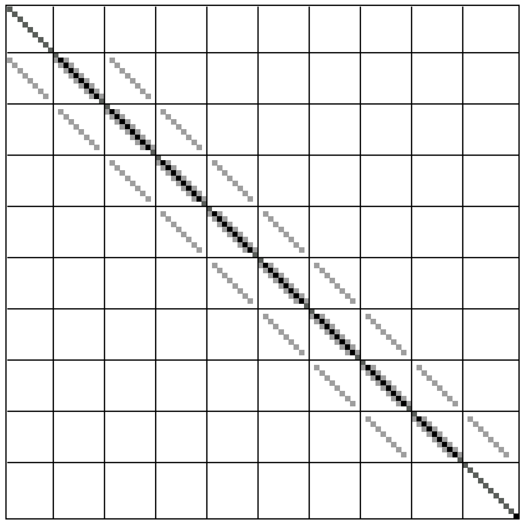
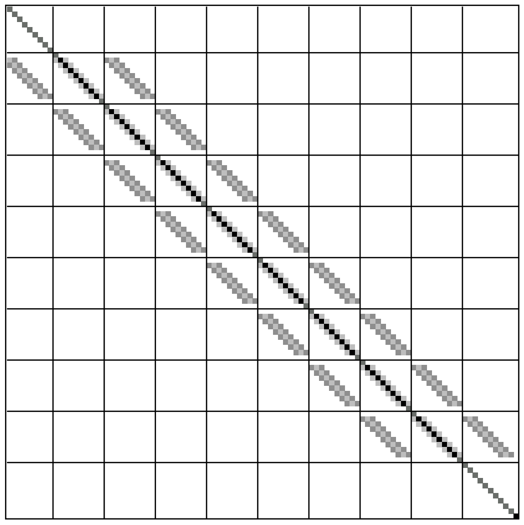

Figure 3.2: 2D-5PF (left) and 2D-9PF (right) for a 10 × 10 case. Each of these squares has 10 × 10 entries.

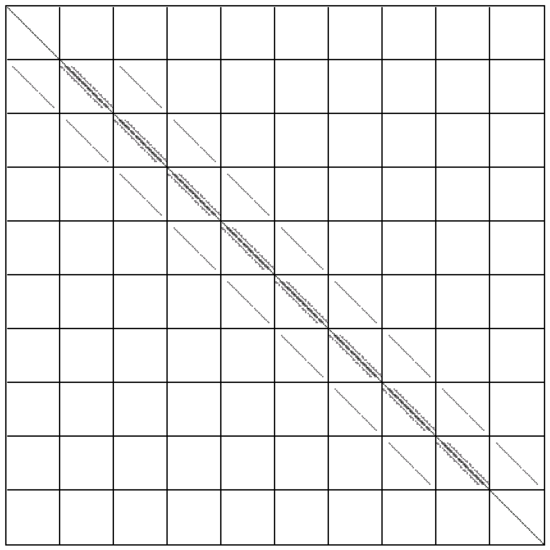
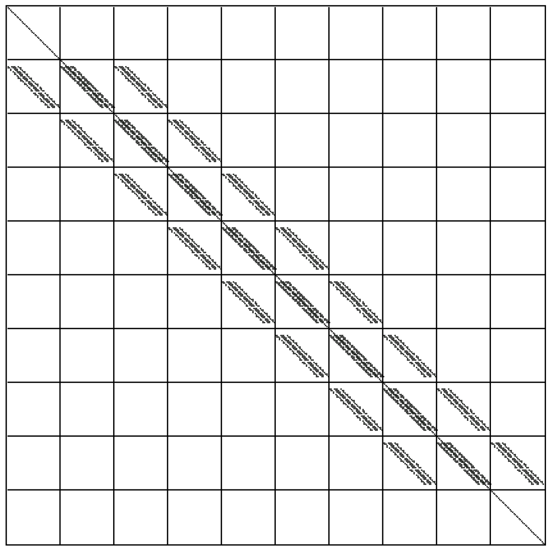

Figure 3.3: 3D-7PF (left) and 3D-19PF (right) for a 10 × 10 × 10 case. Each of these squares has 100 × 100 entries.

Another option which exploits such kind of sparsity is the band format. It includes all coefficients even zero entries contained between both sides of the main diagonal and fitted in a narrowed band. Although the band width storage format is more general or flexible than the diagonal storage format due to the possibility of modification of some zero entries by non zero values, it needs to store more values. See Fig. 3.4 for a draft of the band matrix storage format. Furthermore, a system with lexicographic ordering (i.e. the *ijk*

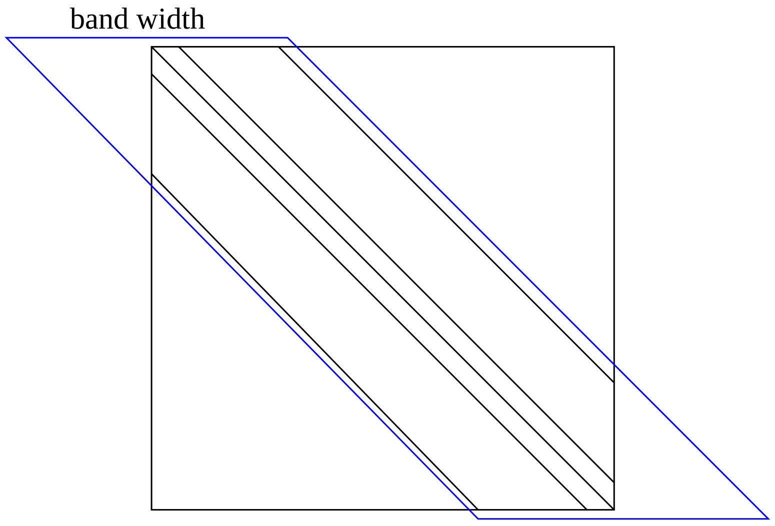

Figure 3.4: The draft of the 2D-5PF case with a band width painted in blue. It evolves the five diagonals.

order) and periodicity in k direction leads to a wider band width than the ordered in *ikj* or *kij*. So in this situation, any of these last orderings is suggested consequently leading to a reduction of the band width. This problem of reordering can be seen in Figs. 3.5, 3.6, 3.7 for a $10 \times 10 \times 10$ case with 7-point formulation.

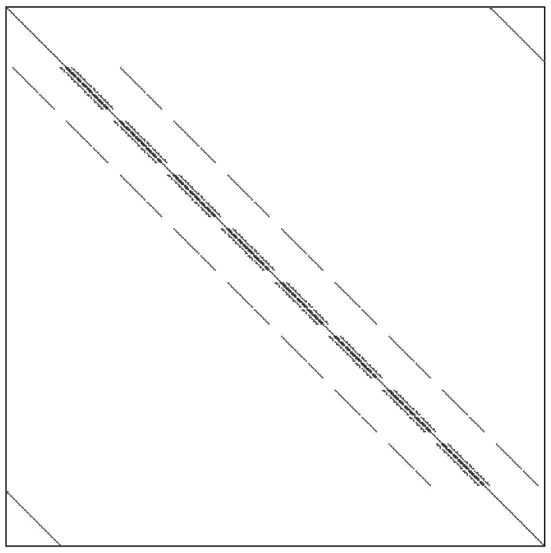

Figure 3.5: The *ijk* ordering for a $10 \times 10 \times 10$ case with *7-point* formulation.

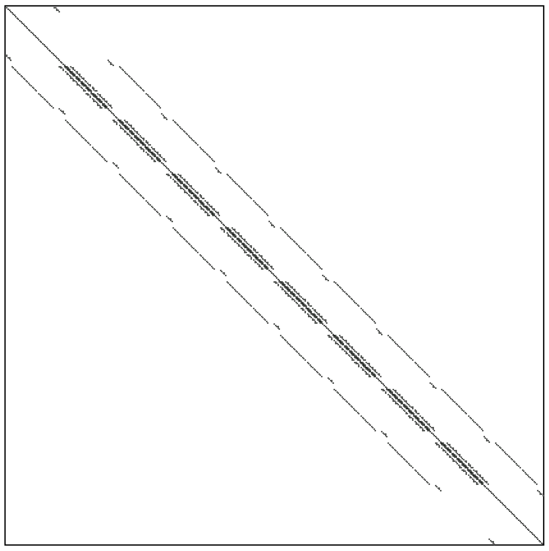

Figure 3.6: The *ikj* ordering for a $10 \times 10 \times 10$ case with *7-point* formulation.

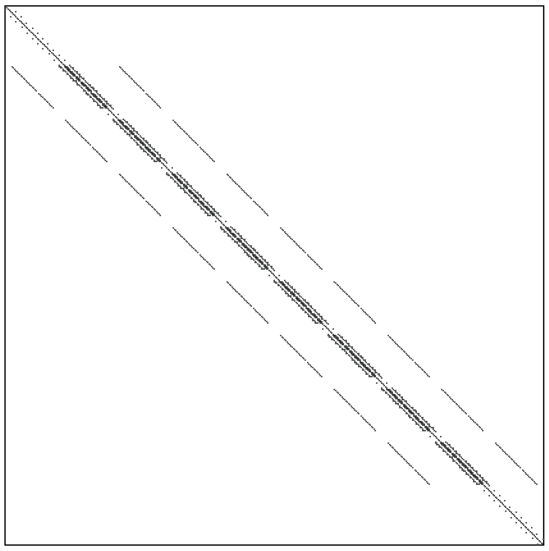

Figure 3.7: The *kij* ordering for a $10 \times 10 \times 10$ case with *7-point* formulation.

For periodicity in k direction the *ijk* ordering has the widest band width. This fact will be considered in the solvers or preconditioners which handle the system in band matrix storage format like complete LU decompositions (14; 43).

3.2 Solving equation systems

Nowadays a full description and even enumeration of all solvers present in the scientific literature would be impossible. However this section is firstly aimed to present a review of the most efficient solvers for CFD equation systems (i.e. the state of the art in CFD solvers). Although the reader interested in general implementations of each solver will find further details in the references, this section provides a compact view of them. And secondly, it will be illustrated not only the advantages and disadvantages of the implementations and efficiencies but also how to link them in order to obtain a more powerful solver by means of the efficiencies linked.

3.2.1 LU solver

Solvers based on factorizations are widely used in narrow banded equation systems (14). The factorization of a non singular matrix A in lower (L) and upper (U) matrices leads in a direct procedure for the evaluation of the inverse so there by leads to a direct evaluation of the solution for a given right hand side. Once given b and factorized A in L and U, the solution is obtained in a two step process (see Alg. 3): a forward substitution followed by a backward substitution.

Algorithm 3: Complete LU factorization: LU

```
evaluate complete LU factorization of $A$
    $Ax = (LU)x = L(Ux) = Ly = b$
solve $Ly = b$ by forward substitution
    $y = L^{-1}b$
solve $Ux = y$ by backward substitution
    $x = U^{-1}y$
```

Sparse matrices, symmetric or non symmetric, are stored in band matrix formats: A, L and U. As mentioned in cases with periodicity, it is better to reorder the system of equations before factorization in order to reduce the band width and definitely to save in storage requirements and computations. For instance, the well known Cholesky factorization (14) suits for symmetric matrices where only the half part of the band matrix is needed for the factorization and storage. The Crout's factorization (43) suits well for non symmetric matrices but a partial pivoting by rows is recommended in order to reduce roundoff errors produced in the inexact arithmetic of the machine.

Although this solver supplies directly the solution within the machine accuracy, it is only feasible for relatively small size systems. This solver comes to fall into large equation systems due to the requirements of memory and number of floating point operations. However, for the pressure correction system of equations where the matrix remains constant along the iterative algorithm of coupling and only the right hand side changes, it may be feasible to compute and store just once such amount of data.

3.2.2 ILU solver

The incomplete LU factorization, also so called ILU factorization, overcomes the problems of the complete LU factorization. It computes and stores the ‘main’ values of L and U matrices from the factorization of such large matrix size. Since the ILU gives an approximation to the matrix, it yields in a iterative method.

The several incomplete factorizations (10; 11; 12), are generalized in the concept of fill-in or ILU(fill-in) and are stored in the band matrix format or the set of diagonals. Furthermore, a threshold (26) can also be introduced to achieve a significant reduction of coefficients in cases with a wide band width. These versions lead to different degrees of sparsity, approximation and finally in more or less efficient iterative algorithms. The ILU factorization proposed by Stone (10), and Zedan (8) for non symmetric matrices is presented here (see Alg. 4).

Algorithm 4: Incomplete LU factorization: ILU

```
evaluate incomplete LU factorization of A
    $Ax = (M - N)x = (LU - N)x = b$
    $Mx^{(k+1)} = Mx^{(k)} - (Ax^{(k)} - b) = Mx^{(k)} + r^{(k)}$ , such that $||M|| >> ||N||$
    $M(x^{(k+1)} - x^{(k)}) = Md^{(k)} = LUd^{(k)} = r^{(k)}$
    $LUd^{(k)} = L(Ud^{(k)}) = Ly^{(k)} = r^{(k)}$
set a guess
    $k = 0,\ x^{(k)}$
    $r^{(k)} = b - Ax^{(k)}$
while $(||r^{(k)}||_2 \geq \epsilon)$ do
    $r^{(k)} = b - Ax^{(k)}$
    solve $Ly^{(k)} = r^{(k)}$ by forward substitution
        $y^{(k)} = L^{-1}r^{(k)}$
    solve $Ud^{(k)} = y^{(k)}$ by backward substitution
        $d^{(k)} = U^{-1}y^{(k)}$
    update solution
        $x^{(k+1)} = x^{(k)} + d^{(k)}$
end while
```

The evaluation of coefficients of the ILU factorization shall be described. Matrices L and U are selected, such that the product of these two matrix, say M, gives a good approximation to the matrix A. A first option is the ILU(0) (i.e., the L and U matrices have the same fill-in that the lower and the upper submatrices of A).

In order to fix ideas, an example for the 3D case with a *19-point* formulation is given (11). Setting an *ijk* ordering of the unknowns, each diagonal of the matrix A and its factorization matrices L and U are represented in Figs. 3.8, 3.9 respectively.

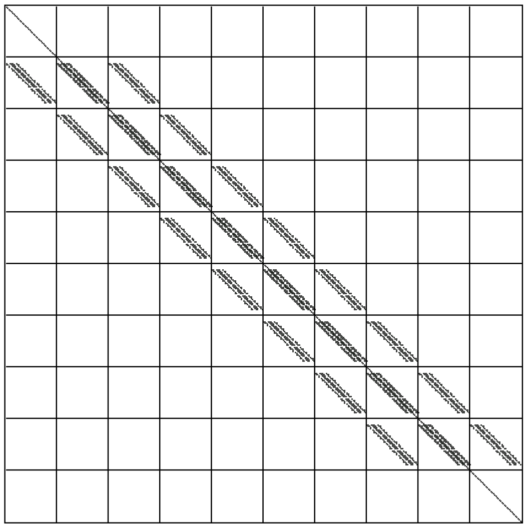

Figure 3.8: Matrix A with *19-point* formulation for a $10 \times 10 \times 10$ size. Each of these squares has 100×100 entries.

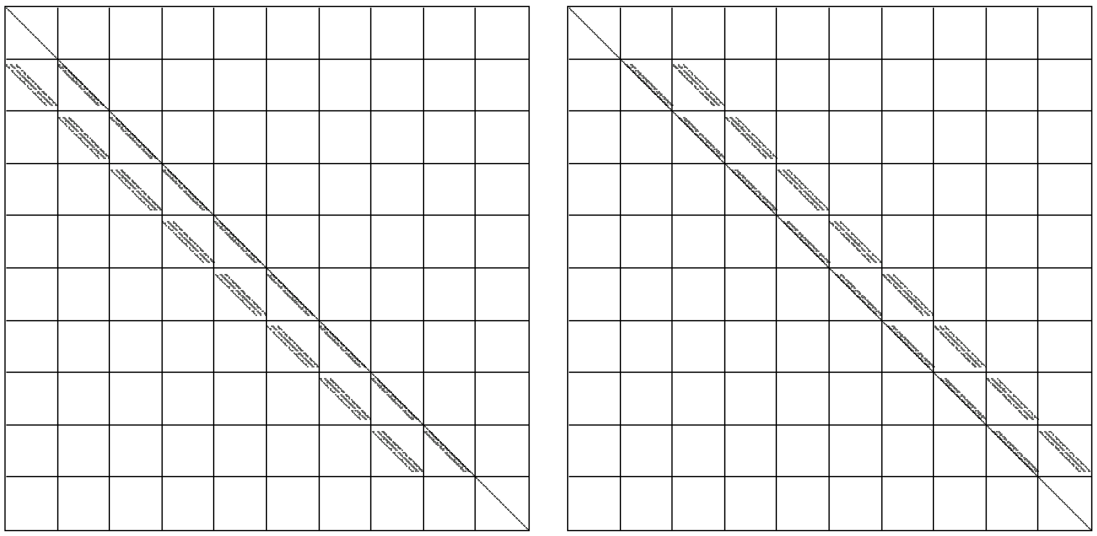

Figure 3.9: (Left) Lower matrix for a $10 \times 10 \times 10$ size.(Right) Upper matrix for a $10 \times 10 \times 10$ size.

After the product of L by U it can be seen that, in addition to the 19 entries in the original matrix A, there are 24 additional coefficients in matrix M. Despite of the additional entries, the equations to be used to determine the coefficients of L and U requires that the 19 coefficients in M remains unchanged from A. In Fig. 3.10, the original entries with points filled in black and half part of the additional entries filled in blue are pictured.

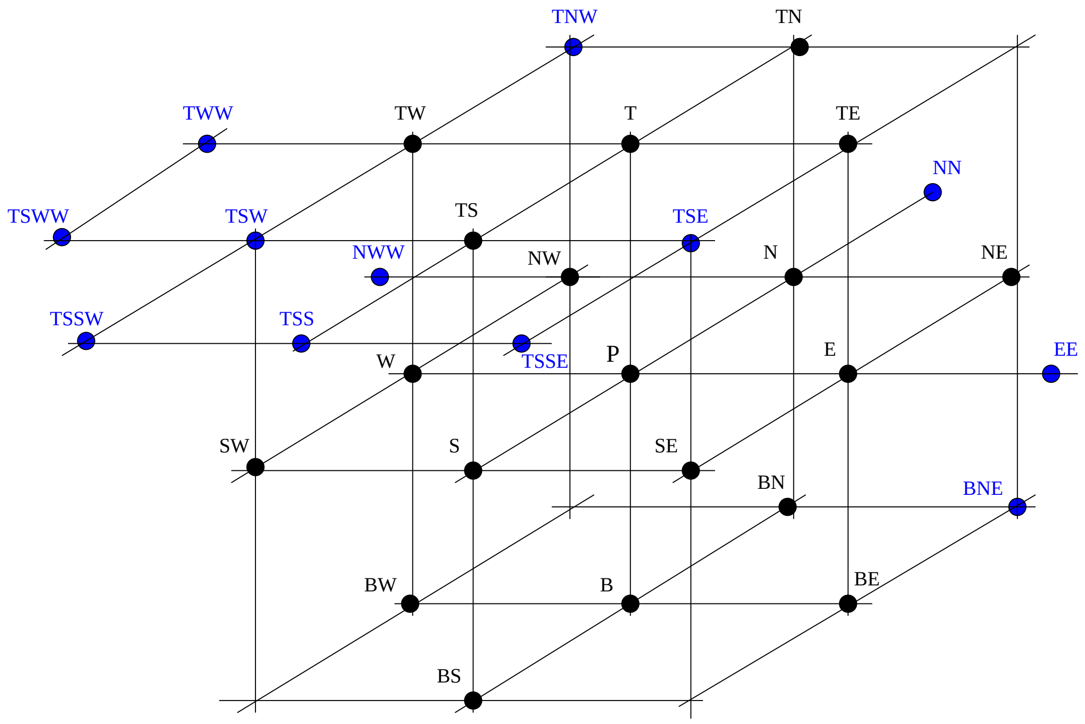

Figure 3.10: Computational molecule with 19 point formulation. Original entries are represented with black filled points. The half part of the additional entries are filled in blue.

The additional 24 entries can be partially canceled through the use of Taylor series expansions and considering points from the 19 entries. For instance,

$$NN = \alpha(-P + 2N)$$

$$TWW = \alpha(-2P + 2W + T)$$

$$BNE = \alpha(-2P + E + N + B)$$

$$TSWW = \alpha(-3P + 2W + S + T)$$

The implementation details can be found in Stone (10) and Zedan (11).

For periodic boundary conditions, it has been seen in previous section that the matrix A contains few additional entries. These entries match well for the additional entries of M, so there is no need to cancel these coefficients. Therefore an ILU for the 3D case is built with or without periodicity just cancelling the additional entries from the original matrix.

The degree of cancellation is controlled by the cancellation parameter α within a range of $[0 : 1)$. Some numerical studies (12; 13) show a good performance for $\alpha = 0.9$ but it is less stable than $\alpha = 0.5$.

Another kind of factorization, SIS (13) (see Alg. 5) is implemented in order to reduce the number of operations, therefore the time of computation per iteration is reduced. Having a look into this algorithm, the computation of the residual vector is unnecessary.

Algorithm 5: Strongly Implicit Solver: SIS

```
evaluate incomplete LU factorization of $A$
    $Ax = (M - N)x = (LU - N)x = b$
    $Mx^{(k+1)} = Nx^{(k)} + b$ , with $||M|| >> ||N||$
    $Mx^{(k+1)} = LUx^{(k+1)} = c^{(k)}$
    $LUx^{(k)} = L(Ux^{(k+1)}) = Ly^{(k)} = c^{(k)}$
set a guess
    $k = 0,\ x^{(k)}$
    $r^{(k)} = b - Ax^{(k)}$
while $(||r^{(k)}||_2 \geq \epsilon)$ do
    evaluate new right hand side
        $c^{(k)} = Nx^{(k)} + b$
    solve $Ly^{(k)} = c^{(k)}$ by forward substitution
        $y^{(k)} = L^{-1}c^{(k)}$
    solve $Ux^{(k+1)} = y^{(k)}$ by backward substitution
        $x^{(k+1)} = U^{-1}y^{(k)}$
end while
```

For 2D cases, the number of new entries present in the N matrix is less than the number of original entries present in the A matrix. Therefore the reduction of the number of operations when performing the right hand side term $c^{(k)}$ instead of the residual $r^{(k)}$ is clear. However, there is no advantage for the 3D case. The 24 additional entries are greater than the 19 entries needed for the evaluation of the residual.

If the additional N entries are used the cancellation parameter is avoided so it is more robust than SIP and MSIP. Finally, the periodic boundary conditions are treated implicitly like in SIP or MSIP.

A variable ILU decomposition (26) can be also useful for some matrix whose bandwidth is not well defined. In such cases a threshold parameter is used in order to neglect some coefficients, thus decreasing time of computation and saving storage in memory. It is specially used in high order wavelet matrix transformations (see the section of multiresolution analysis with wavelets). Following this method a robust ILU is defined without regarding the real entries of the matrix.

3.3 Krylov solvers

In this section some Krylov solvers (40; 25) shall be described. Although the Krylov solvers are in theory direct solvers, the inexact arithmetic of that operations leads to iterative solvers. The derivation of Krylov solvers starts from the idea of finding out the solution for x of the linear system $Ax = b$ by means of orthogonal directions of descent in a minimization functional problem $\phi(x)$ like

$$\phi(x) = \frac{1}{2} < x, Ax > - < x, b >$$

Let us assume that $x = (x_1, x_2)$, then the minimal of the function $\phi(x)$ (see Fig. 3.11 for descent directions) which is also the solution to the linear system is found.

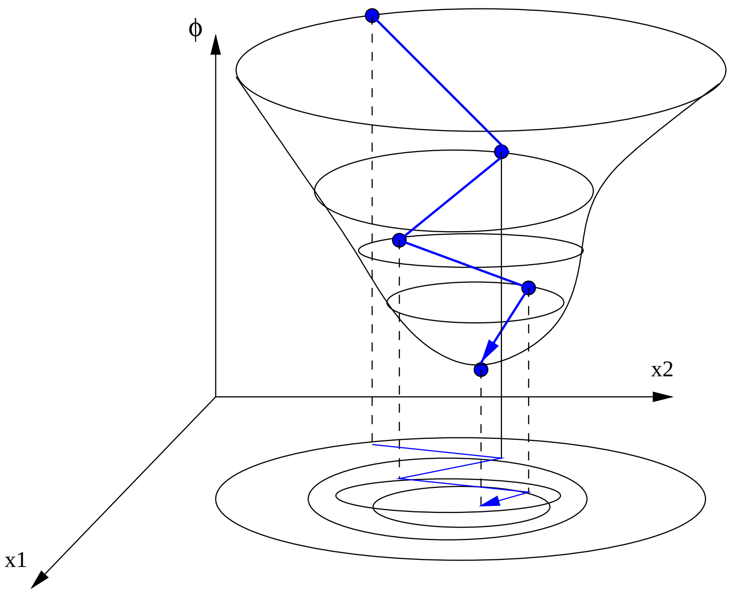

Figure 3.11: Steepest descent directions over the surface of the function ϕ until they arrive to the minimal of the function.

With the initial guess $x^{(0)}$ and the initial residual $r^{(0)}$, the Krylov solver computes in the k-th iteration the optimal correction $p^{(k)}$ over the k-th Krylov subspace associated with A and $r^{(0)}$ $K^k(A, r^{(0)})$

$$K^k(A, r^{(0)}) \equiv span\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, ..., A^{k-1}r^{(0)}\}$$

in the sense that the 2-norm of the residual $r^{(k)} = b - A(x^{(0)} + p^{(k)})$ is minimized. The relation between the minimal residual correction, $p^{(k)}$, and the orthogonality of the new residual $r^{(k)}$ to the shifted Krylov space $AK^k(A, r^{(0)})$

$$AK^k(A, r^{(0)}) \equiv span\{Ar^{(0)}, A^2r^{(0)}, ..., A^{k-1}r^{(0)}\}$$

is given by the following theorem.
Theorem The vector $p^{(k)} \in K^k(A, r^{(0)})$ satisfies

$$p^{(k)} = min||b - A(x^{(0)} + p)||_2, \quad p \in K^k(A, r^{(0)})$$

if and only if

$$r^0 - Ap^k \perp AK^k(A, r^{(0)})$$

□

The Conjugate Gradient solver (CG), the BiConjugate Gradient STABilized solver (BiCGSTAB), and the Generalized Minimal RESidual solver (GMRES) have been widely used in CFD problems (44). Features of Krylov solvers and guidelines about their implementations have been summarized.

3.3.1 CG solver

The Conjugate Gradient (40) (see Alg.6) is an effective solver for symmetric positive definite matrices. It is based on the steepest descent solver but it is improved using conjugate directions $p^{(k)}$. Each new direction is orthogonal to the set of previous directions. Moreover it is scaled by the factor α in order to update the solution minimizing the residual norm.

Algorithm 6: Conjugate Gradient: CG

```
set a guess
    $k = 0,\ x^{(k)}$
    $r^{(k)} = b - Ax^{(k)}$
while $(||r^{(k)}||_2 \geq \epsilon)$ do
    $\rho^{(k)} =< r^{(k)}, r^{(k)} >$
    evaluate new direction and scaled factor
        if $(k = 0)$
            $p^{(k+1)} = r^{(k)}$
        else
            $\beta = \frac{\rho^{(k)}}{\rho^{(k-1)}}$
            $p^{(k+1)} = r^{(k)} + \beta p^{(k)}$
        end if
    evaluate new direction and scaled factor
        $q^{(k+1)} = Ap^{(k+1)}$
        $\alpha = \frac{\rho^{(k)}}{< p^{(k+1)}, q^{(k+1)} >}$
    update solution and residual
        $x^{(k+1)} = x^{(k)} + \alpha p^{(k+1)}$
        $r^{(k+1)} = r^{(k)} - \alpha q^{(k+1)}$
    $k = k + 1$
end while
```

In absence of roundoff errors the series of conjugate directions and series of residuals are so called A-orthogonal. Then the following theorem is written.

Theorem. Let A be a symmetric positive definite matrix and $p^{(0)}, ..., p^{(N-1)}$ are A-orthogonal, for any $x^{(0)} \in \mathbb{R}^N$ given the algorithm converges to the exact solution in less or equal than N iterations. $\square$

This theorem guarantees not only the convergence of the algorithm but also in exact arithmetic, it would be a direct solver with N steps.

3.3.2 BiCGSTAB solver

The conjugate gradient method is not suitable for non symmetric matrices because the series of residual vectors loose its orthogonality. Thus a bi orthogonal process (45) is done in order to improve the conjugate directions. For BiCGSTAB solver (see Alg. 7), the conjugate directions and the residuals are updated in two steps. In this case two parameters α and β are needed to scale the conjugate directions and residuals.

Algorithm 7: Bi Conjugate Gradient STABilized: BiCGSTAB

```
set a guess
    $k = 0,\ x^{(k)}$
    $r^{(k)} = b - Ax^{(k)}$

while $(||r^{(k)}||_2 \geq \epsilon)$ do
    $\rho^{(k)} =< r^{(0)}, r^{(k)} >$
    evaluate new direction and scaled factor
        if $(k = 0)$
            $p^{(k+1)} = r^{(k)}$
        else
            $\beta = \frac{\rho^{(k)}}{\rho^{(k-1)}} \frac{\alpha}{\omega}$
            $p^{(k+1)} = r^{(k)} + \beta(p^{(k)} - \omega q^{(k)})$
        end if

    evaluate new direction and scaled factor
        $q^{(k+1)} = Ap^{(k+1)}$
        $\alpha = \frac{\rho^{(k)}}{< r^{(0)}, q^{(k+1)} >}$
        $s^{(k+1)} = r^{(k)} - \alpha q^{(k)}$

    check direction
        if $(||s^{(k+1)}||_2 < \epsilon)$
            update solution
                $x^{(k+1)} = x^{(k)} + \alpha p^{(k+1)}$
            break while
        end if

    evaluate new direction and scaled factor
        $t^{(k+1)} = As^{(k+1)}$
        $\omega = \frac{< t^{(k+1)}, s^{(k+1)} >}{< t^{(k+1)}, t^{(k+1)} >}$

    update solution and residual
        $x^{(k+1)} = x^{(k)} + \alpha p^{(k+1)} + \omega s^{(k+1)}$
        $r^{(k+1)} = s^{(k)} - \omega t^{(k+1)}$
        $k = k + 1$

end while
```

BiCGSTAB solver can also be used in symmetric positive definite matrices instead of CG solver. However, the convergence rate is equal to CG but at twice the cost per iteration.

3.3.3 GMRESR solver

Like BiCGSTAB solver, the Generalized Minimal RESidual solver (46) and the restarted version GMRESR (47) deals with the non symmetric matrices. The orthogonalization process is based on a modified version of the Gram-Schmidt orthogonalization (43). Each direction is found in order to minimize the following problem.

$$p^{(k)} = min||b - A(x^{(0)+p})||_2 = min||r^{(0)} - Ap||_2,\ p \in K^k(A, r^{(0)})$$

Let us consider V_k the orthogonal basis for the Krylov's subspace $K^k(A, r^{(0)})$ and H_k the upper $k \times k$ Hessenberg matrix generated from the orthogonalization process. The direction $p^{(k)}$ can be computed as

$$p^{(k)} = V_k y^{(k)},\ y^k = H_k^{-1}||r^{(0)}||_2 e_1$$

Where e_1 is the unit vector $e_1 = (1, 0, ..., 0)^T$. Substituting the last expression in the minimization problem the below is found

$$x^{(k)} = x^0 + V_k y^{(k)} \leftarrow min||\ ||r^{(0)}||_2 e_1 - H_k y^{(k)}\ ||_2$$

The solution of the minimization problem is done by a QR factorization with matrix transformations: the Given's rotations.

$$Q_k H_k = R_k$$

Introducing the QR factorization in the 2-norm of the minimization problem

$$Q_k(||r^{(0)}||_2 e_1 - H_k y^{(k)}) = Q_k||r^{(0)}||_2 e_1 - R_k y^{(k)}$$

The minimization problem is reduced to solve the following triangular matrix system

$$R_k y^{(k)} = Q_k||r^{(0)}||_2 e_1 = g_k$$

Once evaluated $y^{(k)}$ the found direction $p^{(k)}$ can be updated easily.

Like CG and BiCGSTAB, GMRES converges in no more than N steps so it could be considered a direct solver. However, the storage of search directions is limited by the available RAM of the computer. Therefore, a restarted version must be used leading to an iterative solver, i.e. the named GMRESR(m).

Choosing m of the searched directions, the solution must be updated and directions cleared. This procedure is repeated until convergence is achieved. If m is too small, GMRESR(m) may be slow to converge or even may be stagnated. A large than necessary value of m will have good convergence rate per iteration but at highly computation cost (time and storage). Although the range of the restart parameter is fixed between 5 and 50, it is said to be problem dependent.

Moreover, the inexact arithmetic produces loses of orthogonality in the search of new directions so a orthogonal check and a reorthogonalization process (48) is added in order to improve from the robustness point of view the algorithm (see details in Alg. 8).

Algorithm 8: Generalized Minimal RESidual Restarted: GMRESR(m)

```
***set*** *a guess*
    $k = 0,\ x^{(k)}$
    $r^{(k)} = b - Ax^{(k)}$
    $\beta = ||r^{(k)}||_2,\ q_1 = \frac{r^{(k)}}{\beta}$
    $g = g(1:k+1) = \beta(1,0,\dots,0)^T$
**while** $(||r^{(k)}||_2 \geq \epsilon)$ **do**
    ***orthogonalization*** *by modified Gramm Schmidt*
        $k = k+1$
        $v_k = v,\ v = Aq_k$
    ***evaluate*** *Hessenberg matrix*
        ***for*** $(i = 1\ to\ k)$
            $H_{i,k} =< q_i, v_k >$
            $v_k = v_k - H_{i,k}q_i$
        ***end for***
        $H_{k+1,k} = ||v_k||_2$
    ***check*** *orthogonality*, $\delta = 10^{-3}$
        ***if*** $(||v||_2 + \delta||v_k||_2 = ||v||_2)$
            ***for*** $(i = 1\ to\ k)$
                $\mu =< q_i, v_k >$
                $H_{i,k} = H_{i,k} - \mu$
                $v_k = v_k - \mu q_i$
            ***end for***
        ***end if***
        $q_{k+1} = \frac{v_k}{H_{k+1,k}}$
    ***evaluate*** *QR factorization by Given's rotations*
        ***if*** $(k > 1)$
            $H_{*,k} = Q_{k-1}H_{*,k}$
        ***end if***
        $\nu = \sqrt{H_{k,k}^2 + H_{k+1,k}^2}$
        $c_k = \frac{H_{k,k}}{\nu},\ s_k = \frac{H_{k+1,k}}{\nu}$
        $H_{k,k} = c_kH_{k,k} - s_kH_{k+1,k},\ H_{k+1,k} = 0$
        $g = G_kg,\ ||r^{(k)}||_2 = |g_{k+1}|$
    ***restart*** *at m-th vector v*
        ***if*** $(k = m)$
            ***solve*** $Ry = w$ *where* $R =$*upper triangular* $(H)_{k\times k},\ w = g(1:k)$
                $y^{(k)} = R^{-1}w$
            ***update*** *solution*
                $x^{(k)} = Qy^{(k)}$
            ***set*** $k = 0$
        ***end if***
***end while***
```

3.4 Preconditioners

In the section 3.1 related with well and ill-conditioned systems it was mentioned that the convergence rate of the iterative methods depends on the condition number $\kappa(A)$, or in other words, on the spectral properties of the coefficient matrix A. Hence the linear system is transformed into one that has the same solution but has better condition number.

Let us call M the non singular preconditioning matrix operating over the linear system $Ax = b$. The matrix A is approximate to the matrix M in such a way that

$$M^{-1}Ax = M^{-1}b$$

is easiest to solve than the original system, thus $\kappa(M^{-1}A) < \kappa(A)$, in spite of the additional computational cost and storage. In this case, the preconditioner is applied to the right of both sides of the linear system. There are two equivalent expressions, the so called left preconditioner and the symmetric preconditioner.

$$AM^{-1}Mx = b$$

$$M^{-\frac{1}{2}}AM^{-\frac{1}{2}}M^{\frac{1}{2}}x = M^{-\frac{1}{2}}b$$

Preconditioners can be applied explicitly, implicitly and both. However, an explicit left preconditioner or a symmetric preconditioner can be implemented in a preprocess and post-process steps. The implementation of each preconditioner into a general solver is outlined in Algs. 9, 10.

Algorithm 9: Left preconditioner

```
***precondition*** *the problem*
    $Ax = b,\ M$
    $\bar{A} = M^{-1}A,\ \bar{b} = M^{-1}b$
***solve*** *the preconditioned problem*
    ***solve*** $\bar{A}x = \bar{b}$
```

Algorithm 10: Symmetric preconditioner

```
***precondition*** *the problem*
    $Ax = b,\ M^{\frac{1}{2}}, M^{-\frac{1}{2}}$
    $\bar{A} = M^{-\frac{1}{2}}AM^{\frac{1}{2}},\ \bar{x} = M^{-\frac{1}{2}}x,\ \bar{b} = M^{-\frac{1}{2}}b$
***solve*** *the preconditioned problem*
    ***solve*** $\bar{A}\bar{x} = \bar{b}$
    ***solve*** $M^{-\frac{1}{2}}x = \bar{x}$
```

The left preconditioner can be considered as a scaling preprocess. Usually, the main diagonal of matrix A, say D_A serves as M preconditioner. Therefore, the storage is very reduced and the inverse M^{-1} can be computed directly by the inverse of all coefficients of D_A.

$$\bar{A} = M^{-1}A = D_A^{-1}A$$

The right preconditioner is usually implemented implicitly in the algorithm. In the scientific literature we found many kinds of preconditioners for Krylov's solvers for sparse matrix based on scaling (40), incomplete LU factorizations (49), polynomial preconditioners (27) and sparse approximate inverses (29) (SPAI).

A diagonal preconditioner was explained also as an explicit left preconditioner. It is also called point jacobi preconditioner, and it is usually considered thought as a scaling technique instead of a preconditioner. Due to the low efficiency, more powerful preconditioners were looked for such as those based on ILU factorizations or on SPAI.

The preconditioned Krylov's solvers can be implemented by adding an intermediate step into the search direction procedure. For instance, BiCGSTAB and GMRESR algorithms are rewritten (see Algs. 11, 12 respectively) including this intermediate step where it is needed.

3.4.1 Factorizations and SPAI

Solvers based on ILU factorizations can be used directly as implicit preconditioners in Krylov's solvers. Our work has been focused on the incomplete factorization due to easy implementation in fixed sparse matrix patterns, low computational cost and low storage requirements. For instance, if our system has a symmetric coefficient matrix, the CG solver has to be used with a symmetric incomplete factorization preconditioner like the Incomplete Cholesky factorization thus leading in the well known ICCG solver (50). For non symmetric coefficient matrix, the ILU can be implemented into the BiCGSTAB or into the GMRESR.

On the other hand, the Sparse Approximate Inverse preconditioner is given as an alternative for the common preconditioners pointed above. It is known that a good preconditioner M must have similar spectral properties to A. Moreover the inverse of the preconditioner M^{-1} must also have similar properties to A^{-1}. Therefore we try to directly find the inverse of the preconditioner by an approximation to the inverse of the matrix A. The inverse is approximated by rows or columns in the dependence whether it is used as a left or a right preconditioner.

One possible implementations is given by Grote (28). The main idea of SPAI consists of guessing the entries of the inverse matrix and finding by minimization in the Frobenius norm each row or column. The experience in such kind of matrix give us a guideline to define quickly this shape in band. However the results provided by different authors (51) show a problem dependent convergence behaviour.

In the following sections two acceleration techniques for any solver are provided. The first one is based on the algebraic multigrid AMG (16; 18) and the second one on the multiresolution analysis MRA (20) with wavelets (23). Both accelerators have similar properties and convergence rates.

Algorithm 11: Preconditioned BiCGSTAB

```
set a guess
    $k = 0,\ x^{(k)}$
    $r^{(k)} = b - Ax^{(k)}$
while $(||r^{(k)}||_2 \geq \epsilon)$ do
    $\rho^{(k)} =< r^{(0)}, r^{(k)} >$
    evaluate new direction and scaled factor
        if $(k = 0)$
            $p^{(k+1)} = r^{(k)}$
        else
            $\beta = \frac{\rho^{(k)}}{\rho^{(k-1)}} \frac{\alpha}{\omega}$
            $p^{(k+1)} = r^{(k)} + \beta(p^{(k)} - \omega q^{(k)})$
        end if
    evaluate new direction and scaled factor
        solve the preconditioned problem
            $M\bar{p} = p^{(k+1)}$
        $q^{(k+1)} = A\bar{p}$
        $\alpha = \frac{\rho^{(k)}}{< r^{(0)}, q^{(k+1)} >}$
        $s^{(k+1)} = r^{(k)} - \alpha q^{(k)}$
    check direction
        if$(||s^{(k+1)}||_2 < \epsilon)$
            update solution
                $x^{(k+1)} = x^{(k)} + \alpha p^{(k+1)}$
            break while
        end if
    evaluate new direction and scaled factor
        solve the preconditioned problem
            $M\bar{s} = s^{(k+1)}$
        $t^{(k+1)} = A\bar{s}$
        $\omega = \frac{< t^{(k+1)}, s^{(k+1)} >}{< t^{(k+1)}, t^{(k+1)} >}$
    update solution and residual
        $x^{(k+1)} = x^{(k)} + \alpha p^{(k+1)} + \omega s^{(k+1)}$
        $r^{(k+1)} = s^{(k)} - \omega t^{(k+1)}$
        $k = k + 1$
end while
```

Algorithm 12: Preconditioned GMRESR(m)

```
set a guess
    $k = 0,\ x^{(k)}$
    solve the preconditioned problem
        $Mr^{(k)} = b - Ax^{(k)}$
    $\beta = ||r^{(k)}||_2,\ q_1 = \frac{r^{(k)}}{\beta}$
    $g = g(1 : k+1) = \beta(1, 0, \dots, 0)^T$
while $(||r^{(k)}||_2 \geq \epsilon)$ do
    orthogonalization by modified Gramm Schmidt
        $k = k + 1$
        solve the preconditioned problem
            $Mv = Aq_k$
        $v_k = v$
    evaluate Hessenberg matrix
        for $(i = 1\ to\ k)$
            $H_{i,k} =< q_i, v_k >$
            $v_k = v_k - H_{i,k}q_i$
        end for
        $H_{k+1,k} = ||v_k||_2$
    check orthogonality, $\delta = 10^{-3}$
        if $(||v||_2 + \delta||v_k||_2 = ||v||_2)$
            for $(i = 1\ to\ k)$
                $\mu =< q_i, v_k >$
                $H_{i,k} = H_{i,k} - \mu$
                $v_k = v_k - \mu q_i$
            end for
        end if
        $q_{k+1} = \frac{v_k}{H_{k+1,k}}$
    evaluate QR factorization by Given's rotations
        if $(k > 1)$
            $H_{*,k} = Q_{k-1}H_{*,k}$
        end if
        $\nu = \sqrt{H^2_{k,k} + H^2_{k+1,k}}$
        $c_k = \frac{H_{k,k}}{\nu},\ \ s_k = \frac{H_{k+1,k}}{\nu}$
        $H_{k,k} = c_kH_{k,k} - s_kH_{k+1,k},\ \ H_{k+1,k} = 0$
        $g = G_kg,\ \ ||r^{(k)}||_2 = |g_{k+1}|$
    restart at m-th vector $v$
        if $(k = m)$
            solve $Ry = w$ where $R$ =upper triangular $(H)_{k\times k},\ \ w = g(1 : k)$
                $y^{(k)} = R^{-1}w$
            update solution
                $x^{(k)} = Qy^{(k)}$
            set $k = 0$
        end if
end while
```

3.5 Algebraic Multigrid algorithm

There are numerous publications (15), tutorials,... about algebraic multigrid (AMG) and here one refers mainly to Huttchinson (52) and Wesseling (16). AMG is based on a hierarchy of linear systems $A_l x_l = b_l$ constructed directly from the original linear system $Ax = b$ at fine grid $l = 1$ to a maximal level $l = l_{max}$ by means of transfer operators. The first system $A_1 x_1 = b_1$ is solved roughly and the error $e_1 = x - x_1$ is erased by adding succesive corrections x_l from the different levels $l = 2, 3.., l_{max}$. Solving each system, different frequency range of corrections are dealt with.

For instance, a two level multigrid scheme begins solving the first system $A_1 x_1 = b_1$ within a certain number ν_1 of iterations. The measure of the difference from the numerical solution x_1 to the exact solution x in this first level $l = 1$ is the error $e_1 = x - x_1$.

If the frequency components of this error are analyzed at using the discrete Fourier transform (DFT), the solver smoothes well the high frequency components and keeps the lowest frequency components nearly untouched. For this reason it is said that the solver acts as smoother of low frequency components of the error. This iterative process is called the pre-smoothing step. An additional system may be found in order to compute this error numerically and then correct the numerical solution.

$$A_1 e_1 = A_1(x - x_1) = b_1 - A_1 x_1 = r_1$$

Unfortunately, the last linear system is computationally expensive as the first system. Furthermore, this error solution contains low frequencies which cannot be solved efficiently by the iterative solver at fine grid. Therefore a transformation of this system into another one is carried out in order to convert the low frequency components into higher frequency components. This transformation is done with the so called restriction R_1^2 and prolongation P_2^1 operators (16).

$$A_2 = R_1^2 A_1 P_2^1$$

$$b_2 = R_1^2 r_1$$

$$A_2 x_2 = b_2$$

Having a look at such transformation a reduced system of equations is obtained, which seems to be derived from a coarsest problem. Hence, the transformation receives the name of coarse grid approximation (16) (CGA) of the original problem. Then, a post-smoothing step is performed within ν_2 iterations in order to obtain an approximation to the error e_1 but in the coarse level $l = 2$, let us say x_2.

This approximation to the error is also called the correction x_2 and it must be transferred and added to the solution x_1 at fine grid. The prolongation operator transfers the vector from the coarsest level $l = 2$ to the finest level $l = 1$.

$$x_1 = x_1 + P_2^1 x_2$$

Finally, this algorithm is repeated until a desired error threshold ϵ is achieved or a maximum number of iterations it_{max} is done. The algebraic multigrid (see Alg. 13), the prediction (see Alg. 14) and the correction (see Alg. 15) algorithms are written as follows:

Algorithm 13: Algebraic Multi Grid: AMG

```
***Fix*** *parameters of multigrid cycle*
    $l_{max}, it_{max}, \nu_1, \nu_2, \epsilon$
**for**($l = 1$ *to* $l_{max} - 1$)
    ***evaluate*** matrix coeff. *by Coarse Grid Approximation*
        $A_{l+1} = R_l^{l+1} A_l P_{l+1}^l$
    ***end do***
***set*** *a guess in finest level*
    $l = 1,\ k = 0,\ x_l^{(k)}$
**while** ($||r||_2 \geq \epsilon$ or $k \leq it_{max}$)
    ***smooth*** *with* $\nu_1$ *iterations*
        ***solve*** $A_l x_l = b_l$
    **if**($l < l_{max}$)
        ***go*** *to the coarser level* $l + 1$
            **predict** $x_{l+1}$
        ***add*** *the correction to the level* $l$
            ***correct*** $x_l$
    **end if**
    $k = k + 1$
**end while**
```

Algorithm 14: Prediction

```
***evaluate*** *right hand side of coarser level* $l + 1$
    $r_l^{(\nu_1)} = b_l - A_l x_l^{(\nu_1)}$
    $b_{l+1} = R_l^{l+1} r_l^{(\nu_1)}$
***set*** *a guess at coarser level*
   $l = l + 1,\ k = 0, x_l^{(k)} = 0$
    ***smooth*** *with* $\nu_1$ *iterations*
        ***solve*** $A_l x_l = b_l$
    **if**($l < l_{max}$)
        ***go*** *to the coarser level* $l + 1$
            **predict** $x_{l+1}$
        ***add*** *the correction to the level* $l$
            ***correct*** $x_l$
    **end if**
```

Algorithm 15: Correction

```
***add*** *the correction to the level l*
    $x_l^{(\nu_1)} = x_l^{(\nu_1)} + P_{l+1}^l x_{l+1}^{(\nu_1)}$
***smooth*** *with $\nu_2$ iterations*
    **solve** $A_l x_l = b_l$
```

3.5.1 Transfer operators

These operators transfer information from one level to the closest level. The restriction operator does it from the fine level l to the coarse level $l+1$ while the prediction transfer operator does it in an opposite way, (i.e., from the coarse level to the fine level).

In order to give an easy explanation to such operators, a suitable set of two dimensional piece wise constant (16) restriction and prolongation operators has been chosen and applied to a two dimensional problem with *5-point* formulation.

Let G_l and G_{l+1} be a fine and coarse grid respectively defined over the domain Ω (see Fig. 3.12).

$$l = \{1, 2, \ldots, l_{max}\}$$

$$G_l = \{(x_1, x_2) \in \Omega,\ \Omega : [0, L_1] \times [0, L_2]\}$$

$$x_i(j) = \{0, \ldots, (j-1)h_i, \ldots, L_i\},\ j = 1, \ldots, \frac{n_i}{2^{l-1}},\ h_i = \frac{L_i}{\frac{n_i}{2^{l-1}} - 1},\ i = \{1, 2\}$$

Let (x_1, x_2) be a point in G_{l+1} associated with a $(i, j)_{l+1}$ index point. Then, there is a set of index points in G_l which are placed at same position that G_{l+1}

$$G_{l+1} = \{(2i, 2j)_l, (2i+1, 2j)_l, (2i, 2j+1)_l, (2i+1, 2j+1)_l$$

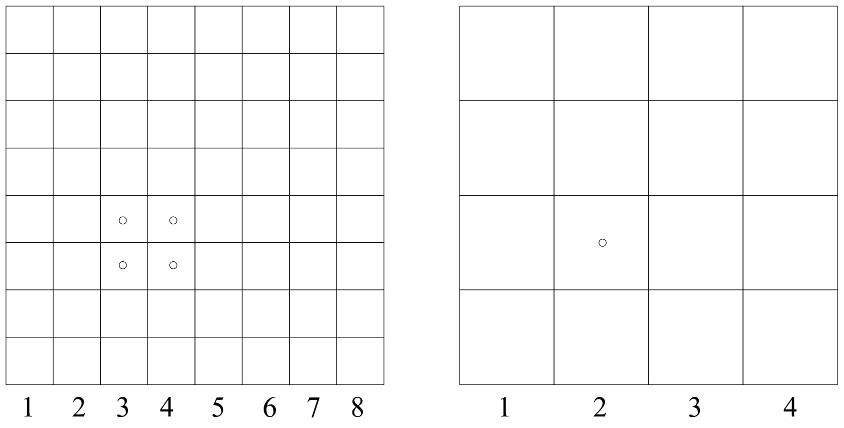

Figure 3.12: Example of two related grids: $G_1 : 8 \times 8$ and $G_2 : 4 \times 4$. G_2 has double size-mesh of G_1.

For a better comprehension, the stencil notation gives a simply representation in two dimensions of these operators.

$$x_l : G_l \to \mathbb{R},\ x_{l+1} : G_{l+1} \to \mathbb{R}$$

$$R_l^{l+1} : G_l \to G_{l+1}$$

$$R_l^{l+1} = \left[\begin{array}{cc} 1 & 1 \\ 1 & 1 \end{array} \right]$$

$$x_{l+1} = R_l^{l+1} x_l$$

$$x(i,j)_{l+1} = x(2i,2j)_l + x(2i+1,2j)_l + x(2i,2j+1)_l + x(2i+1,2j+1)_l$$

In this example (see Fig. 3.13), the restriction operator R performs a summation over blocks of four grid points weighted by a unit factor.

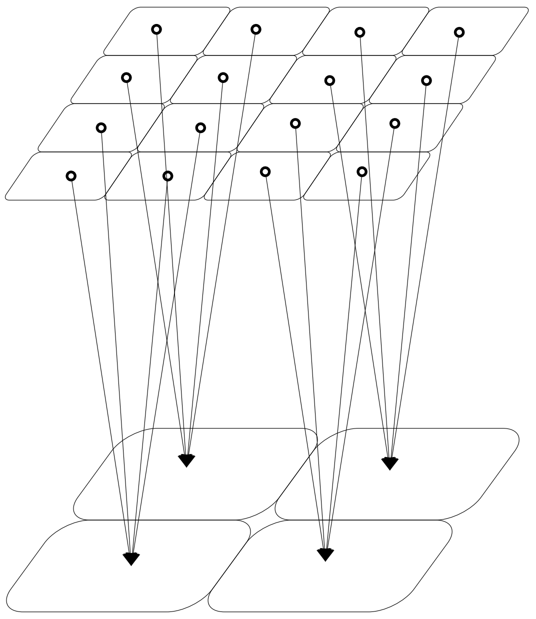

Figure 3.13: Restriction process over blocks of four grid points.

If natural ordering is considered by applying the restriction operator to the example with fine level $l = 1$, and coarse level $l = 2$ leads to the matrix form

$$x_1 = (x(1,1)_1, x(2,1)_1, \ldots, x(7,8)_1, x(8,8)_1)^T_{64\times 1}$$

$$x_2 = (x(1,1)_2, x(2,1)_2, \ldots, x(3,4)_2, x(4,4)_2)^T_{16\times 1}$$

$$x_2 = R_1^2 x_1 = R_{1,y}^2 R_{1,x}^2 x_1$$

$$R^2_{1,x} = \begin{bmatrix} \begin{bmatrix} 11 & & \\ & \cdots & \\ & & 11 \end{bmatrix}_{4\times 8} & & \\ & \ddots & \\ & & \begin{bmatrix} 11 & & \\ & \cdots & \\ & & 11 \end{bmatrix}_{4\times 8} \end{bmatrix}_{32\times 64}$$

$$R^2_{1,y} = \begin{bmatrix} \begin{bmatrix} 1 & & 1 & \\ & \cdots & & \\ & 1 & & 1 \end{bmatrix}_{4\times 8} & & \\ & \ddots & \\ & & \begin{bmatrix} 1 & & 1 & \\ & \cdots & & \\ & 1 & & 1 \end{bmatrix}_{4\times 8} \end{bmatrix}_{16\times 32}$$

In general, weight factors in restriction operator must satisfy certain rule (16):

$$\sum_{i,j} R_{i,j} = \left(\frac{h_{l+1}}{h_l}\right)^{\alpha}$$

Where α varies from inside to boundary points and depends of boundary conditions implemented. Here let us assume $\alpha = 2$ in the whole domain. Then $\sum_{i,j} R_{i,j} = 4$ everywhere.

Prolongation operator P (see Fig. 3.14) is expressed in matrix form as the transpose of the restriction operator R. $P^l_{l+1} = R^{l+1,T}_l$.

$$P^l_{l+1} : G_{l+1} \to G_l$$

$$P^l_{l+1} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$x_l = P^l_{l+1} x_{l+1}$$

$$x(2i,2j)_l = x(2i+1,2j)_l = x(2i,2j+1)_l = x(2i+1,2j+1)_l = x(i,j)_{l+1}$$

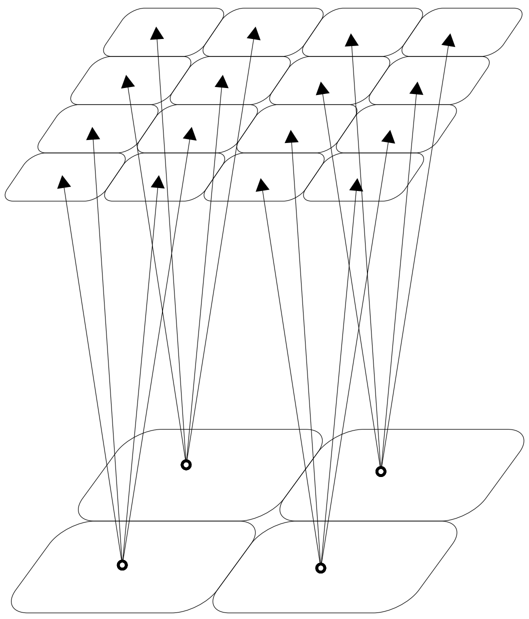

Figure 3.14: Prolongation process to blocks of four grid points.

Furthermore the product of both operators leads into a scaled identity matrix.

$$P^{l}_{l+1}R^{l+1}_{l} = \alpha I$$

In this example with piece wise constant operators, $\alpha = 4$. However, it is possible to find a pair of transfer operators were the product is equal to the identity. In such case, there is an important feature between transfer operators

$$P^{l}_{l+1} = R^{l+1,T}_{l} = R^{l+1,-1}_{l}$$

This feature points out the ability to use a single matrix and its transpose like a prolongation or a restriction operator. This idea will be recovered and exploited in multiresolution analysis by wavelets (21).

Finally, the coarse grid approximation is evaluated by

$$A_{l+1} = R^{l+1}_{l,y}R^{l+1}_{l,x}A_l P^{l}_{l+1,y}P^{l}_{l+1,x}$$

Seeing the example, each 2×2 set of equations on fine grid G_l is transformed in to one equation on coarse grid G_{l+1} with the same stencil.

$$A_l = \begin{bmatrix} & A^n_l & \\ A^w_l & A^p_l & A^e_l \\ & A^s_l & \end{bmatrix}$$

$$A_{l+1} = \begin{bmatrix} & A^n_{l+1} & \\ A^w_{l+1} & A^p_{l+1} & A^e_{l+1} \\ & A^s_{l+1} & \end{bmatrix}$$

Where

$$A^s_{l+1}(i,j) = A^s_l(2i,2j) + A^s_l(2i+1,2j)$$

$$A^w_{l+1}(i,j) = A^w_l(2i,2j) + A^w_l(2i,2j+1)$$

$$A^e_{l+1}(i,j) = A^e_l(2i+1,2j) + A^e_l(2i+1,2j+1)$$

$$A^n_{l+1}(i,j) = A^n_l(2i,2j+1) + A^n_l(2i+1,2j+1)$$

$$\begin{aligned} A^p_{l+1}(i,j) = {} & A^p_l(2i,2j) + A^p_l(2i+1,2j) + \\ & A^p_l(2i,2j+1) + A^p_l(2i+1,2j+1) + \\ & A^s_l(2i,2j+1) + A^s_l(2i+1,2j+1) + \\ & A^w_l(2i+1,2j) + A^w_l(2i+1,2j+1) + \\ & A^e_l(2i,2j) + A^e_l(2i,2j+1) + \\ & A^n_l(2i,2j) + A^n_l(2i+1,2j) \end{aligned}$$

Similar expressions can be obtained for the *9-point* formulation, the *7-point* formulation and the *19-point* formulation. Instead of a transfer operator acting over a 2×2 set of grid points for a *5-point* formulation, the extension can be derived at the *9-point* formulation with 3×3 set of grid points. In three dimensional cases, for the *7-point* formulation there is $2 \times 2 \times 2$ set of grid points and for the *19-point* formulation $3 \times 3 \times 3$ set of grid points.

However, for three dimensional cases the stencil notation does not helps to the representation of these operators and only an algebraic expression can be given as

$$A_{l+1} = R^{l+1}_{l,x} R^{l+1}_{l,y} R^{l+1}_{l,x} A_l P^l_{l+1,z} P^l_{l+1,y} P^l_{l+1,x}$$

A more detailed description of transfer operators is exposed in Zeeuw (17).

3.6 Multiresolution Analysis with wavelets

The basic idea behind wavelet multiresolution analysis (20) (MRA) is to represent a function in terms of basis of functions called wavelets (23) having discrete scales and locations. In other words, wavelet analysis can be viewed as a multilevel or multiresolution representation of a function, where each level of resolution consists of basis of functions having the same scale but located at different positions.

Therefore, a function (continuous or sampled), a vector or a matrix can be split by wavelet analysis into low and high frequency parts. Both resulting parts have about half the dimension of the original one. Again the part containing the low frequency components can be decomposed in the same way. This procedure predestines the wavelets to serve as a restriction and prolongation operators (22) in the usual multigrid method.

3.6.1 Multilevel representation of a function

In order to develop a multilevel representation of a function $f(x)$ in $L^2(\mathbb{R})$ a sequence of embedded subspaces V_i is looked for

$$\{0\} \cdots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \cdots \subset L^2(\mathbb{R})$$

with the following properties:

1. $\bigcup_{l \in Z} V_l$ is dense in $L^2(\mathbb{R})$.
2. $\bigcap_{l \in Z} V_l = \{0\}$.
3. The embedded subspaces are related by a scaling law
$$f(x) \in V_l \Longleftrightarrow f(2x) \in V_{l-1}$$
4. Each subspace is spanned by integer translates of a single function $g(x)$ such that
$$f(x) \in V_0 \Longleftrightarrow f(x+1) \in V_0$$

Since the space V_1 lies within the space V_0, we can express any function in V_1 in terms of the basis of functions of V_0. In particular,

$$\phi(x) = \sum_{i=-\infty}^{\infty} a_i \phi(2x - i), \ i \in \mathbb{Z}$$

where i is a finite number and a_i is a square summation sequence. This equation is called dilation equation or scaling relation.

If we define
$$\phi_{l,i} = 2^{l/2}\phi(2^l - i)$$
then $\phi_{l,i}, \ i \in \mathbb{Z}$ forms a basis for the space V_l. The dilation parameter l shall be referred to as the scale.

In general, a function may be approximated by the projection $P_l f$ onto the space V_l:

$$P_l f = \sum_{i=-\infty}^{\infty} c_{l,i}\phi_{l,i}(x), \ i \in \mathbb{Z}$$

and in fact $P_l f$ approaches f when $l \to \infty$.

Let us now define a new subspace W_{l+1} such that it is the orthogonal complement of V_{l+1} in V_l,
$$V_l = V_{l+1} \oplus W_{l+1}, \ V_{l+1} \perp W_{l+1}$$
where $\oplus$ represents a direct sum. Let us introduce a wavelet function ψ such that $\psi(x-i)$ form a basis for the subspace W_0. Then

$$\psi_{l,i} = 2^{\frac{l}{2}}\psi(2^l x - i)$$

It is a basis for W_l. If in addition, the $\{\psi(x-i),\ i \in \mathbb{Z}\}$ forms an orthonormal set of functions, then it follows that $\{\psi_{l,i},\ l, i \in \mathbb{Z}\}$ forms an orthonormal basis for $L^2(\mathbb{R})$.

Let us denote the projection of f on W_l as $Q_l f$. Then we have

$$P_l f = P_{l+1} f + Q_{l+1} f$$

This means that $Q_l f$ represents the detail that needs to be added to get from one level of approximation to the next finer level of approximation.

Furthermore, since the space W_1 is contained in the space V_0, the wavelet function can be expressed in terms of the scaling function at the next higher scale,

$$\psi(x) = \sum_{i=-\infty}^{\infty} b_i \psi(2x - i),\ i \in \mathbb{Z}$$

There is a relation between the so called filter coefficients a_i and b_i

$$b_i = (-1)^i a_{N-1-i}$$

The derivation (53) of filter coefficients requires the solution of an N coefficient system where the approximation of order $p = \frac{N}{2}$ of any function f is involved.

3.6.2 Multiresolution decomposition and reconstruction

Multiresolution decomposition (54) takes the values $c_{l,i}$ of a function f vector x_l or a matrix A_l at level l and decomposes them into

1. The values $c_{l+1,i}$ of the approximation, $P_{l+1}f$ (low frequency components) at next coarser level $l+1$.

2. The values $d_{l+1,i}$ of the detail component, $Q_{l+1}f = P_l f - P_{l+1} f$ (high frequency components) at next coarser level $l+1$.

Consider a function f. Let $P_l f$ denote the projection of f onto the subspace V_l and $Q_l f$ denote the projection onto the subspace W_l. Thus,

$$P_l f = \sum_{i=-\infty}^{\infty} c_{l,i} \phi_{l,i}(x),\ c_{l,i} =< f, \phi_{l,i} >$$

$$Q_l f = \sum_{i=-\infty}^{\infty} d_{l,i} \psi_{l,i}(x),\ d_{l,i} =< f, \psi_{l,i} >$$

Since W_{l+1} is the orthogonal complement of V_{l+1} in V_l

$$P_{l+1} f = P_l f - Q_{l+1} f$$

Substituting this in

$$c_{l+1,i} =< P_{l+1} f, \phi_{l+1,i} >$$

leads to the following result:

$$c_{l+1,i} = \frac{1}{\sqrt{2}} \sum_{j=-\infty}^{\infty} c_{l,j} a_{j-2i}$$

Similarly, it can be shown that

$$d_{l+1,i} = \frac{1}{\sqrt{2}} \sum_{j=-\infty}^{\infty} d_{l,j} a_{N-1-j+2i}$$

Multiresolution reconstruction (54) uses coefficients $c_{l+1,i}$ and $d_{l+1,i}$ at level $l+1$ to reconstruct the coefficients $c_{l,i}$ at next finer level l.

Since W_{l+1} is the orthogonal complement of V_{l+1} in V_l,

$$P_l f = P_{l+1} f + Q_{l+1} f$$

Substituting this in

$$c_{l,i} =< P_l f, \phi_{l,i} >$$

leads to

$$c_{l,i} = \frac{1}{\sqrt{2}} \sum_{j=-\infty}^{\infty} c_{l+1,j} a_{i-2j} + \frac{1}{\sqrt{2}} \sum_{j=-\infty}^{\infty} d_{l+1,j} (-1)^i a_{N-1-i+2j}$$

3.6.3 Mallat's transform and inverse transform

The Mallat's transform (54) provides a simple means of transforming data from one level of resolution, l, to the next coarser level of resolution $l+1$. The inverse Mallat's transform is a transformation from the coarser level $l+1$ to the finer level l.

The Mallat's transform algorithm implements the decomposition process as follows. Consider a string of data $c_{l,i}$ of finite and even length n which represents the approximation , $P_l f$, to a function. For convenience, suppose that this data is periodic with period $n >> N$. Then the matrix form of multiresolution decomposition equation is

$$\begin{bmatrix} c_{l+1,0} \\ \times \\ c_{l+1,1} \\ \times \\ c_{l+1,2} \\ \times \\ \cdots \\ \cdots \\ c_{l+1,\frac{n}{2}-1} \\ \times \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{N-1} & \cdots & 0 \\ 0 & a_0 & a_1 & \cdots & a_{N-2} & \cdots & 0 \\ 0 & 0 & a_0 & \cdots & a_{N-3} & \cdots & 0 \\ 0 & 0 & 0 & \cdots & a_{N-4} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & \cdots & \cdots & a_{N-1} \\ a_{N-1} & 0 & 0 & \cdots & \cdots & \cdots & a_{N-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_2 & a_3 & a_4 & \cdots & 0 & \cdots & a_1 \\ a_1 & a_2 & a_3 & \cdots & 0 & \cdots & a_0 \end{bmatrix} \begin{bmatrix} c_{l,0} \\ c_{l,1} \\ c_{l,2} \\ c_{l,3} \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ c_{l,n-2} \\ c_{l,n-1} \end{bmatrix}$$

in which $\times$ represents information of no value. The effect of the periodicity is simply a wrap around of the coefficients at the bottom left corner of the matrix.

A similar process gives the coefficients, $d_{l+1,i}$ of the detail which is lost in reducing the resolution of the data. The matrix form is

$$\begin{bmatrix} d_{l+1,0} \\ \times \\ d_{l+1,1} \\ \times \\ d_{l+1,2} \\ \times \\ \cdots \\ \cdots \\ d_{l+1,\frac{n}{2}-1} \\ \times \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} a_{N-1} & -a_{N-2} & \cdots & -a_0 & \cdots & 0 \\ 0 & a_{N-1} & \cdots & a_1 & \cdots & 0 \\ 0 & 0 & \cdots & -a_2 & \cdots & 0 \\ 0 & 0 & \cdots & a_3 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cdots & \cdots & -a_0 \\ -a_0 & 0 & \cdots & \cdots & \cdots & a_1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{N-3} & -a_{N-4} & \cdots & 0 & \cdots & -a_{N-2} \\ -a_{N-2} & a_{N-3} & \cdots & 0 & \cdots & a_{N-1} \end{bmatrix} \begin{bmatrix} c_{l,0} \\ c_{l,1} \\ c_{l,2} \\ c_{l,3} \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ c_{l,n-2} \\ c_{l,n-1} \end{bmatrix}$$

The inverse Mallat transform implements the reconstruction process. The matrix form is

$$\begin{bmatrix} c_{l,0} \\ c_{l,1} \\ c_{l,2} \\ c_{l,3} \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ c_{l,n-2} \\ c_{l,n-1} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} a_0 & 0 & 0 & \cdots & a_{N-1} & \cdots & a_1 \\ a_1 & a_0 & 0 & \cdots & 0 & \cdots & a_2 \\ a_2 & a_1 & a_0 & \cdots & 0 & \cdots & a_3 \\ a_3 & a_2 & a_1 & \cdots & 0 & \cdots & a_4 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{N-2} & a_{N-3} & a_{N-4} & \cdots & \cdots & \cdots & a_{N-1} \\ a_{N-1} & a_{N-2} & a_{N-3} & \cdots & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & a_{N-3} & \cdots & 0 \\ 0 & 0 & 0 & \cdots & a_{N-2} & \cdots & a_0 \end{bmatrix} \begin{bmatrix} c_{l+1,0} \\ 0 \\ c_{l+1,1} \\ 0 \\ c_{l+1,2} \\ 0 \\ \cdots \\ \cdots \\ c_{l+1,\frac{n}{2}-1} \\ 0 \end{bmatrix}$$

$$+\frac{1}{\sqrt{2}} \begin{bmatrix} a_{N-1} & 0 & 0 & \cdots & -a_0 & \cdots & -a_{N-2} \\ -a_{N-2} & a_{N-1} & 0 & \cdots & a_1 & \cdots & a_{N-3} \\ a_{N-3} & -a_{N-2} & a_{N-1} & \cdots & -a_2 & \cdots & -a_{N-4} \\ -a_{N-4} & a_{N-3} & -a_{N-2} & \cdots & 0 & \cdots & a_{N-5} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_1 & -a_2 & a_3 & \cdots & \cdots & \cdots & -a_0 \\ -a_0 & a_1 & -a_2 & \cdots & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & -a_2 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & a_1 & \cdots & a_{N-1} \end{bmatrix} \begin{bmatrix} d_{l+1,0} \\ 0 \\ d_{l+1,1} \\ 0 \\ d_{l+1,2} \\ 0 \\ \cdots \\ \cdots \\ d_{l+1,\frac{n}{2}-1} \\ 0 \end{bmatrix}$$

3.6.4 Wavelet transfer operators

Let us denote the Mallat transformations H, G of a vector $x \in \mathbb{R}^n$,with n even.

$$H, G : \mathbb{R}^n \to \mathbb{R}^{\frac{n}{2}}$$

$$x_l = c_{l,i}, i = 0, \cdots, n-1$$

$$Hx_l = c_{l+1,i}, i = 0, \cdots, \frac{n-1}{2}$$

$$Gx_l = d_{l+1,i}, i = 0, \cdots, \frac{n-1}{2}$$

The Mallat transformations satisfy

$$H^T H + G^T G = I$$

$$HH^T + GG^T = I$$

$$GH^T = HG^T = 0$$

We use I to denote the identity matrix of appropriate size. Therefore, H may be interpreted as an averaging or smoothing operator whose smoothing effect increases with N. Conversely, G can be viewed as a difference approximation operator.

Following the philosophy of the multigrid methods, we first apply to the l-system $A_l x_l = b_l$ the iterative solver as pre-smoother ν_1 times with the initial guess $x_l^{(0)}$. An approximate solution $x_l^{(\nu_1)}$ is obtained. In general, the smoother only reduces error components in the direction of eigenvectors corresponding to large eigenvalues of A_l (22). But in the presence of strong anisotropies, say in x-direction, the error $e_l^{(\nu_1)} = x - x_l^{(\nu_1)}$ may contain not only a part which is a low frequency in both the x and y-directions, but also a part which is a high frequency in x direction. Consequently, we approximate the error $e_l^{(\nu_1)}$ in both spaces $Vx_{l+1} \otimes Vy_{l+1}$ and $Wx_{l+1} \otimes Vy_{l+1}$ instead of only in $Vx_{l+1} \otimes Vy_{l+1}$ as in a standard multigrid procedure. Where $\otimes$ denotes the tensor product of the spaces, operators and vectors. Then, the resulting coarse grid correction for the two level multigrid is

$$x_l = x_l^{(\nu_1)} + [(H_x^T \otimes H_y^T) A_{l+1}^{-1} (H_x \otimes H_y) + (G_x^T \otimes H_y^T) A_{l+1}^{-1} (G_x \otimes H_y)](b_l - A_l x_l^{(\nu_1)})$$

It may be viewed as an additive subspace correction with respect to the orthogonal subspaces $Vx_{l+1} \otimes Vy_{l+1}$ and $Wx_{l+1} \otimes Vy_{l+1}$.

One has to remark that the transition from A_l to A_{l+1} increases the band width moderately as far as N becomes not too large. This fact must be kept in mind in order to obtain a constant band width if an incomplete LU factorization with fixed number of off diagonals is used for all levels.

3.6.5 The Haar's wavelet transfer operator

The Haar basis (53) is an orthogonal basis which has been known since 1910. It is also the earliest known example of wavelet basis, and perhaps one of the simplest. Furthermore, Haar wavelets supplies transfer operators without increasing the bandwidth of matrix. In this case $N = 2$ and the Mallat's transformation is quite close to piece wise constant restriction and prolongation operators described in the algebraic multigrid section.

$$a_0 = 1, \ a_1 = 1$$

$$H_x = \frac{1}{\sqrt{2}} R_x$$

$$H_y = \frac{1}{\sqrt{2}} R_y$$

$$H_{1,x}^2 = \frac{1}{\sqrt{2}} \begin{bmatrix} \begin{bmatrix} 11 & & \\ & \cdots & \\ & & 11 \end{bmatrix}_{4\times 8} & & \\ & \ddots & \\ & & \begin{bmatrix} 11 & & \\ & \cdots & \\ & & 11 \end{bmatrix}_{4\times 8} \end{bmatrix}_{32\times 64}$$

$$H_{1,y}^2 = \frac{1}{\sqrt{2}} \begin{bmatrix} \begin{bmatrix} 1 & & 1 & \\ & \cdots & & \\ & 1 & & 1 \end{bmatrix}_{4\times 8} & & \\ & \ddots & \\ & & \begin{bmatrix} 1 & & 1 & \\ & \cdots & & \\ & 1 & & 1 \end{bmatrix}_{4\times 8} \end{bmatrix}_{16\times 32}$$

Finally, the transformation of A_l into A_{l+1} for a 5-point formulation matrix leads

$$A_{l+1} = H_y H_x A_l H_y^t H_x^t$$

$$\begin{aligned}
A_{l+1}^s(i,j) &= \frac{1}{4}(A_l^s(2i,2j) + A_l^s(2i+1,2j)) \\
A_{l+1}^w(i,j) &= \frac{1}{4}(A_l^w(2i,2j) + A_l^w(2i,2j+1)) \\
A_{l+1}^e(i,j) &= \frac{1}{4}(A_l^e(2i+1,2j) + A_l^e(2i+1,2j+1)) \\
A_{l+1}^n(i,j) &= \frac{1}{4}(A_l^n(2i,2j+1) + A_l^n(2i+1,2j+1)) \\
A_{l+1}^p(i,j) &= \tfrac{1}{4} \quad (A_l^p(2i,2j) + A_l^p(2i+1,2j) + \\
&\qquad A_l^p(2i,2j+1) + A_l^p(2i+1,2j+1) + \\
&\qquad A_l^s(2i,2j+1) + A_l^s(2i+1,2j+1) + \\
&\qquad A_l^w(2i+1,2j) + A_l^w(2i+1,2j+1) + \\
&\qquad A_l^e(2i,2j) + A_l^e(2i,2j+1) + \\
&\qquad A_l^n(2i,2j) + A_l^n(2i+1,2j))
\end{aligned}$$

For a 7-point formulation in a three dimensional case the Haar wavelet is already used

$$A_{l+1} = H_z H_y H_x A_l H_z^T H_y^T H_x^T$$

For a 9-point formulation in a two dimensional case or a 19-point formulation in a three dimensional case, it is better to use a wavelet with $N > 2$. For example the Daubechies wavelet (53) of fourth order $N = 4$. However, as it was pointed out above, the pattern of the resulting matrix changes to a more dense pattern. The number of diagonal entries increases slightly at each level.

3.7 Comparison between AMG and MRA

We have seen that AMG and MRA share the same algorithm. Both of them have different grids or levels of resolution and the corrections or the details to the finest grid solution are added. Furthermore, the algebraic expression of the coarse grid approximation is quite similar. However, the transfer operators are implemented in different ways. For AMG, the restriction and prolongation operators act in blocks well defined (i.e. 2×2, 3×3, $2 \times 2 \times 2$ and $3 \times 3 \times 3$). Hence it is necessary to know the grid size and the distribution of grid points. For MRA based on wavelets, the transfer operators are applied directly as they were defined: products between matrices and vectors. This fact gives to the wavelet multiresolution analysis a great generalization to any sparse matrix pattern. Moreover, just changing the order of a wavelet, say Daubechies with orders 4, 6, 12 and 20, different coarse grid approximations can be more easily obtained than AMG does by blocking in predefined sets of grid points.

Finally this generalization can be extended even to non structured grids in order to have a non structured multigrid (55).

3.8 Stopping criteria

We have already described some families of iterative solvers based on ILU's and Krylov's subspaces. In the algorithm of each iterative solver, there is a common step, the stopping criterion. It decides when the algorithm has achieved the right solution within a certain error. Since the error in such linear systems is never known, the 2-norm of the residual vector $||r||_2$ has to be used.

$$||r||_2 = \sqrt{\sum_{i,j,k} r_{i,j,k}^2}$$

Thus, let k be the k-th iteration, if the $||r^{(k)}||_2$ is small enough, say smaller than a given threshold ϵ, the iterative solver can be stopped and get the solution of $x^{(k)}$.

Most solvers perform at least once the computation of the residual vector so it is easy to implement the stopping criterion based on its 2-norm. Others like GMRESR performs this computation directly without needing the residual vector. And others perform an approximation to the residual like the SIS (13). In any case, the 2-norm of the residual vector leads to a good criterion of convergence to the solution.

However, this criterion must be generalized for all kind of problems. Different coefficient matrices and right hand sides can have solutions with large values and generate large residual values. Conversely, solutions with small values generate small residual values. Therefore a relative residual norm $\rho^{(k)}$ should be used instead of the residual norm $||r^{(k)}||$. For instance the residual norm is divided by the factor $||b||_2$ which features the problem.

$$\rho^{(k)} = \frac{||r^{(k)}||_2}{||b||_2}, \; ||b||_2 \neq 0$$

Another stopping criterion is based on a fixed number of iterations ν. The solver stops when the counter of iterations achieve this value. That happens, for example, in inner loops of the multigrid in the pre-smoothing and the post-smoothing steps. In this case, the global stopping criterion is based on the residual norm of the finest level. And the fixed number of iterations can be used as local criterion. For example, the preconditioning step is included in the krylov's solver and since it has to be with low computational cost, it performs only a fixed small number ν of iterations.

Since the number of iterations does not change, we say that the criterion is static. In other cases, the local number of iterations is dynamic and limited by a relative threshold ϵ_r and a maximum number of iterations it_{max}. The relative threshold ϵ_r is usually fixed in the range $70\% - 90\%$ of the residual norm $||r^{(0)}||_2$ at the guessed solution or before performing the local iterations.

$$\epsilon_r = \alpha||r^{(0)}||_2, \ \alpha \in [0.7, 0.9]$$

Finally, there is another dynamic criterion based on the convergence rate of the local iterative solver. The convergence rate β^k for the k-th iteration is expressed as

$$\beta^{(k)} = \frac{r^{(k-1)}}{r^k}$$

Representing $\beta^{(k)}$ versus the iteration number stagnation can be detected in the iterative procedure and exit the algorithm. This dynamic criterion is used in the decision of change to a coarse grid or a coarse multiresolution within the multigrid algorithm.

These criteria are embedded in the multigrid algorithm 16 with a preconditioned solver 17.

3.9 Sequential performance of solvers

The scope of this section is to provide a comparison between the different solvers built in the solver layer for a given model problem. The comparison is based on two issues: the time spent in the resolution until a defined residual criterion and the amount of memory needed to store the vectors and matrices involved in the computation of the solution. On the other hand, the model problem is designed to show the stability and convergence behaviour of each solver. The main idea underlying on this test is to find out a robust solver of a wide range of CFD problems.

3.9.1 CFD model problem

We are interested on the behaviour of solvers for convection and diffusion equations under the boundary conditions that produce ill conditioned linear systems. For this purpose, a two-dimensional partial differential equation with different boundary conditions has been designed:

$$\frac{\partial u\phi}{\partial x} + \frac{\partial v\phi}{\partial y} - \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = f$$

$$(x, y) \in \Omega : [0, 2] \times [0, 2]$$

Where the velocity field $\vec{V} = \{u, v\}$ in function of the Reynolds number satisfies the mass conservation equation.

$$u = x^2(1 - 2y)Re, \quad v = 2x(y^2 - y)Re$$

Algorithm 16: AMG + solver + preconditioner + criteria

```
**if**$(l = 1)$
    ***fix*** *parameters of multigrid cycle*
        $l_{max}, it_{max}, \nu_1, \nu_2, \epsilon, \alpha, \beta$
    **for**$(l = 1$ *to* $l_{max} - 1)$
        ***evaluate*** *matrix coeff. by Coarse Grid Approximation*
            $A_{l+1} = R_l^{l+1} A_l P_{l+1}^l$
        $\epsilon_r = \epsilon$
    ***end do***
**else**
    ***evaluate*** *right hand side of coarse level* $l + 1$
        $r_l = b_l - A_l x_l$
        $b_{l+1} = R_l^{l+1} r_l$
        $l = l + 1$
    $\epsilon_r = \alpha\epsilon$
**end if**
***set*** *a guess in level* $l$
    $k = 0,\ x_l^{(k)}$
*global static iteration*
***while***$(\rho^{(k)} \geq \epsilon$ *and* $k \leq it_{max})$
        ***solve*** $A_l x_l = b_l$ *with preconditioner*
        ***if*** $(l < l_{max})$
            ***go*** *to the coarser level* $l + 1$
                ***call*** *AMG at l+1*
            ***add*** *the correction to the level* $l$
                $x_l = x_l + P_{l+1}^l x_{l+1}$
            ***solve*** $A_l x_l = b_l$ *with preconditioner*
        ***end if***
        $k = k + 1;$
***end while***
```

Then, the convective and diffusive terms are discretized to obtain the coefficient matrix A with a 5-point formulation.

In order to satisfy all boundary conditions (i.e. Dirichlet, Neumann and periodic conditions), a single solution ϕ is set to this problem.

$$\phi = cos(\pi x) + cos(\pi y) + cos(3\pi x) + cos(3\pi y)$$

Finally, the right hand side b is evaluated from the product of the matrix A by the vector ϕ instead of the integration of f at each volume $\mathcal{V}$.

$$b = f\mathcal{V} = A\phi$$

If the solution ϕ is known, it is possible to evaluate and analyze at each iteration the numerical error $e^{(k)}$ at any iteration k. Therefore a comparison of the convergence rates between solvers may be done. In this sense the Discrete Fourier Transform of the error help us to explain the spectral properties of the different solvers.

3.9.2 Sequential performance

For the given problem, the performance of each solver in a single process $np = 1$ is done. Since no communications are needed the convergence behaviour of each solver gives an idea of how good may it be solving the problem in a np parallel environment.

Solvers are compared from two points of view: the convergence behaviour and the memory resources required. A list of solvers is given in table 3.1.

Solver		Implementation features
0	band LU	Crout's implementation + partial pivoting
1	Gauss-Seidel	not overrelaxed
2	ILU	MSIP implementation, $\alpha = 0.5$
3	BiCGSTAB	preconditioned with MSIP($\alpha = 0.5$)
4	GMRESR	restart=10, preconditioned with MSIP($\alpha = 0.5$)
5	AMG	10 levels, smoother MSIP($\alpha = 0.5$)
6	MRA	10 levels, smoother MSIP($\alpha = 0.5$)

Table 3.1: List of the solvers used in this test.

Each solver performs iterations until the normalized residual norm reaches the accuracy $\epsilon = 10^{-4}$.

$$\frac{||r^{(k)}||_2}{||b||_2} < \epsilon$$

A battery of cases for different Reynolds numbers and different square grid sizes $I \times I$ are tested following Alg. 18. The results of the test, executed in the PC cluster (see appendix), are grouped by the Reynolds number and represented with the pair of axis: size problem(x) - time of computation(y). For Dirichlet boundary conditions, the results for three Reynolds $(0, 10^2, 10^4)$ are given in Figs. 3.15, 3.16 and 3.17.

Algorithm 17: Preconditioned solver

```
local dynamic iteration
while($\frac{||r^{(k-1)}||_2}{||r^{(k)}||_2} \geq \beta$ and $k < \nu_1$)
    solve $A_l x_l = b_l$
        local static iteration
        for ($k = 1$ to $k = \nu_2$)
            solve $M_l \bar{p}_l = p_l$
        end for
end while
```

Algorithm 18: Sequential performance of solvers

```
for ($solver = 0$ to $solver = 6$)
    for ($Re = 0$ to $Re = 10000$, $Re = Re * 100$)
        for ($I = 32$ to $I = 512$, $I = I * 2$)
            generate the problem $Ax = b$
            fix accuracy of the solution to $\epsilon = 10^{-4}$
            initialize unknown $x = 0$
            t0=start_wall_clock(gettimeofday)
                solve $Ax = b$ such that $\frac{||r^2||}{||b^2||} < \epsilon$
            t1=stop_wall_clock(gettimeofday)
            tcomp=t1-t0

            write throughput: tcomp and memory resources

        end for
    end for
end for
```

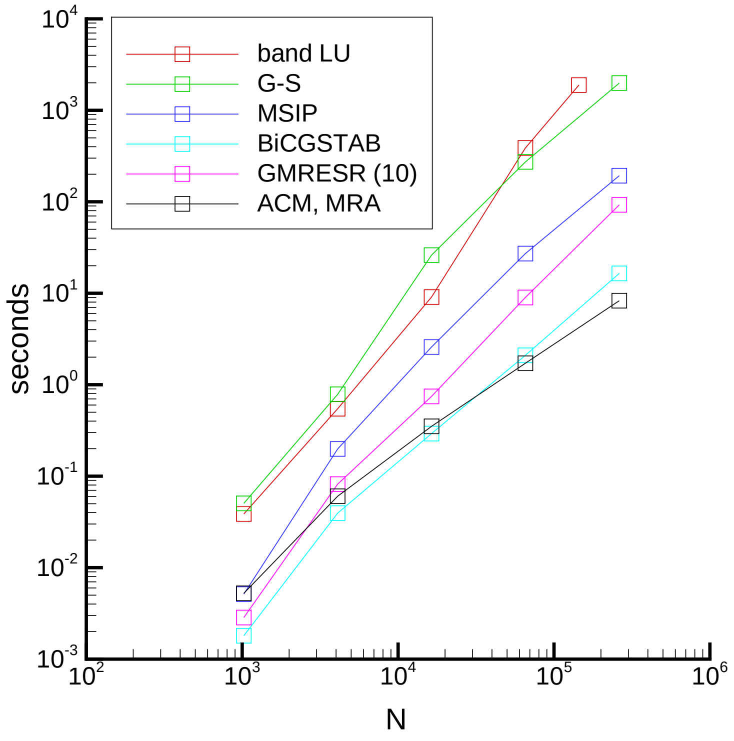

Figure 3.15: Time of computation at Re=0 for different size problems.

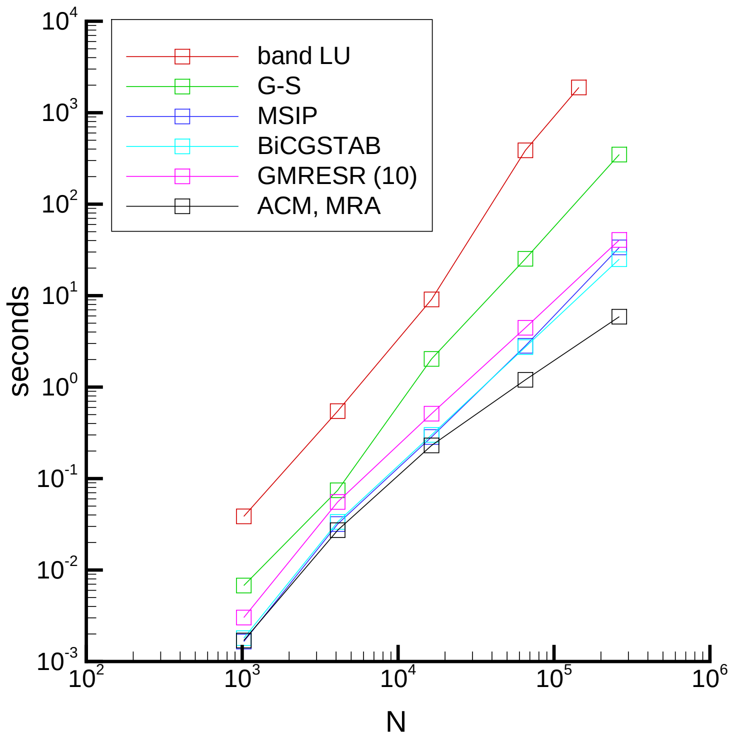

Figure 3.16: Time of computation at Re=100 for different size problems.

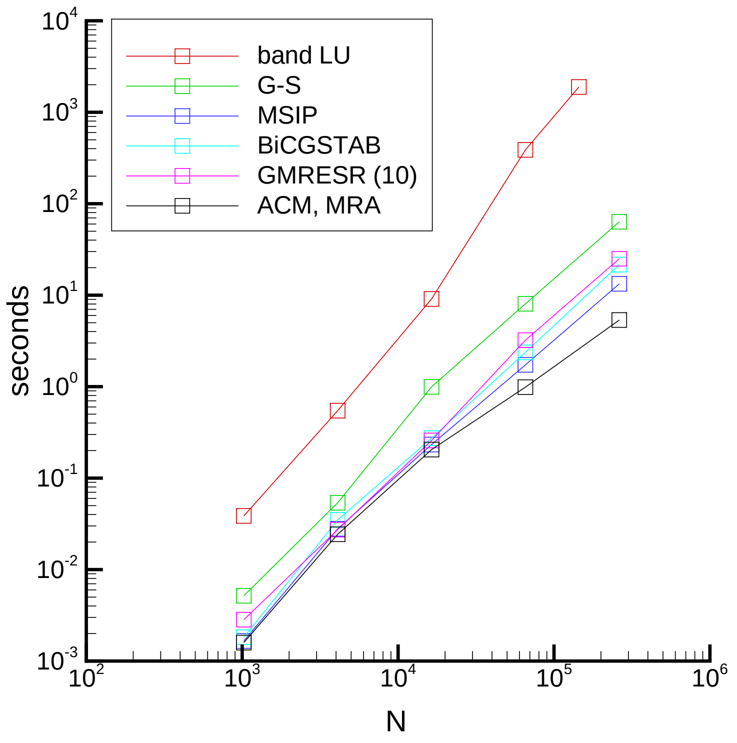

Figure 3.17: Time of computation at Re=10000 for different size problems.

Although full results have not been shown due to the large amount of tested cases, the above results show that ACM and MRA are the best solvers for all Reynolds numbers and problem sizes. Furthermore, the difference of time of computation among solvers grows with the size problem, being the worst case for the pure diffusion problem, (i.e. $Re = 0$) where the matrix is symmetric. For large Reynolds numbers the matrix becomes non symmetric and the solvers work better. Since the pure diffusion problem is the hardest case, we reduce the number of cases here in after to such case and for any boundary condition.

For Neumann boundary conditions and for the pure diffusion case, (i.e. $Re = 0$), the same sequential performance test is repeated. Since the Neumann boundary conditions lead to a singular matrix, a point at the center of the domain is fixed in order to set a single solution. The results represented in Fig. 3.18 are quite similar to the obtained for Dirichlet boundary conditions.

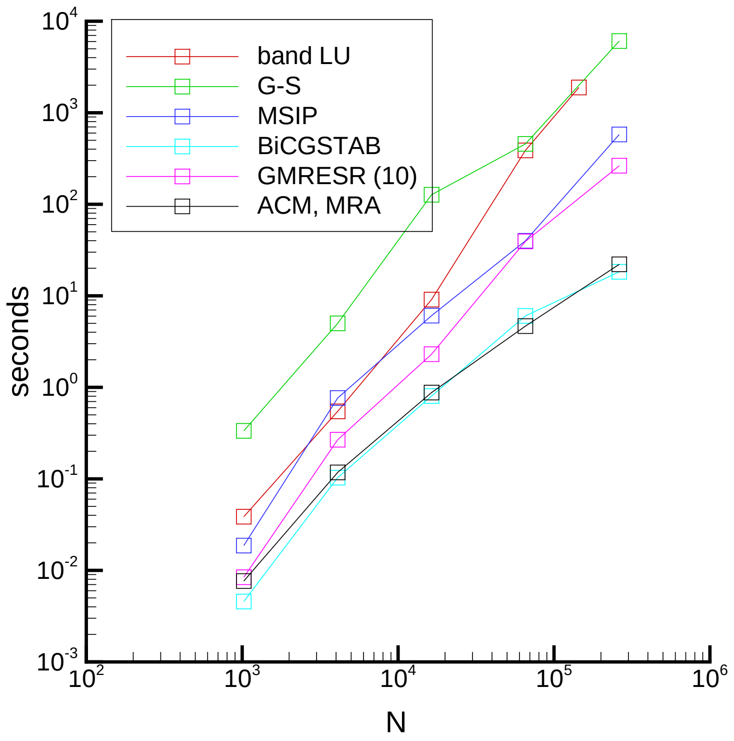

Figure 3.18: Time of computation at Re=0 for different size problems.

Fig. 3.18 shows that although BiCGSTAB reported good results the best convergence behaviour is with ACM and MRA solvers.

The memory requirements for each solver at each size are represented in Fig. 3.19.

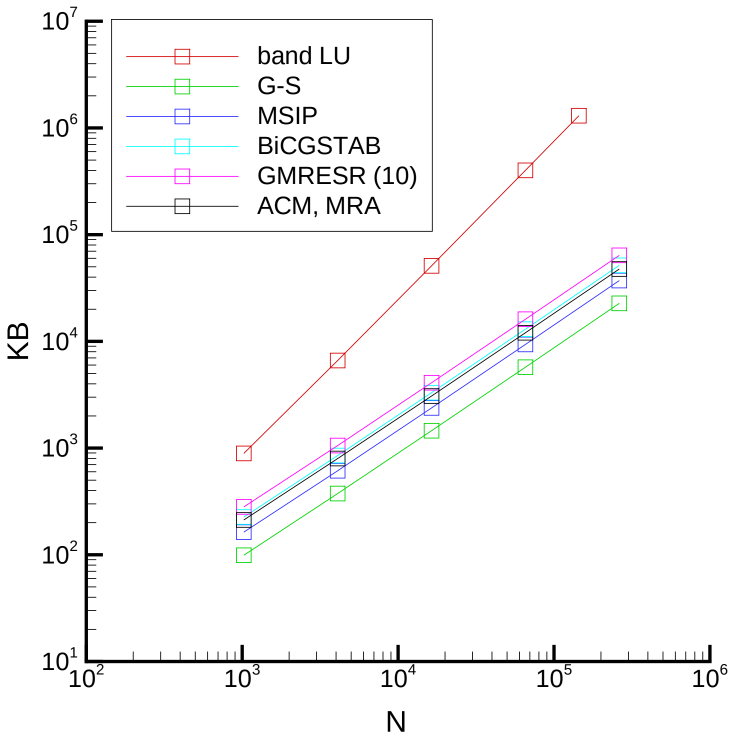

Figure 3.19: Memory requirements in Kbytes for different size problems.

It is worth noting that GMRES with a restart of 10 requires more memory resources than the rest of solvers. A higher value of the restart parameter is also tested but it increases largely the memory without a reduction of the time of computation. Therefore, ACM and MRA are also better than BiCGSTAB and GMRESR from the cost-memory point of view.

3.10 Nomenclature

A	discretization matrix
a	coeff. in A
	filter coeff.
b	right hand side, filter coeff.
CV	control volum
c	filter coeff.
d	auxiliar vector
e	unit vector
G	grid
g	similar to d
H	Hessenberg matrix, Haar wavelet
I	unknowns per direction
ILU	incomplete LU decomposition
i	index coordinate
K	Krylov space
L	lower matrix from LU
LU	lower upper decomposition
l	coeff. of matrix L
M	auxiliar matrix for LU
N	number of unknowns, similar to M
P	prediction, projection
p	similar to d
Q	matrix from QR, projection
QR	QR factorization
q	similar to d
R	restriction, similar to Q
Re	Reynolds number
r	residual
s	similar to d
t	similar to d
U	upper matrix from LU
u	coeff. of matrix U,
V	Krylov base, subspace
v	similar to d
W	wavelet, subspace
w	similar to d
x	unknown
y	similar to d
z	similar to d
$\{u, v, w\}$	fluid flow velocity components
$\{x, y, z\}$	cartesian coordinates

Greek symbols

α	relaxation parameter, scalar value
β	scalar value
δ	general perturbation
κ	condition number
ϵ	precission
μ	similar to β
ν	fixed number of iterations
Ω	domain
ω	similar to β
ϕ	general variable, scaling function
ψ	wavelet function
ρ	similar to β

Other symbols

$5 - PF$	five point formulation
$7 - PF$	seven point formulation
$9 - PF$	nine point formulation
$19 - PF$	nineteen point formulation
$\mathcal{V}$	volume of the CV
$<, >$	inner product
$\|\|.\|\|_2$	Euclidean norm
$\oplus$	tensor product

Subscripts

NGB	general neighbour grid point
$*,$	for all elements of a row
$, *$	for all elements of a column
l	level, go down to the level l

Superscripts

(k)	k-th iteration
-1	inverse
T	transpose
$\frac{1}{2}$	square root
$-$	modified value
l	go up to the level l

Chapter 4

Parallel linear solvers

4.1 Introduction

The reliability of the engineering community on CFD is growing due to the ability to solve complex fluid flow and heat transfer phenomena with accuracy and within a reasonable elapsed time. During the past decade, the increase of the speed of processors and memories has contributed to the reduction of this time, and hence, it has enabled to afford large engineering problems. However, while the complexity of such problems grows, the improvements in processor technology become physically limited. For that reason, only parallelism is able to boost performance significantly and to deal with long time and large memory consuming problems.

4.1.1 Hardware for parallel computing

Among the different architecture types derived along the past two decades, only a few of them have become nowadays the commonly used machines for scientific computing.

Following the classification of computers introduced by Flynn (1972), the most recent architectures fit into the SIMD (Single Instruction stream/Multiple Data stream) and MIMD (Multiple Instruction stream/Multiple Data stream) categories. The term stream relates to the sequence of data or instructions as seen by the machine during the execution of a program. However, it was not until the early to mid 1980s that machines conforming to the MIMD classification were available commercially. At this point, it is worth distinguishing between two MIMD categories, shared memory and distributed memory MIMD computers. The shared memory machines are considered to be tightly coupled, whilst the distributed memory machines are regarded as loosely coupled and employ the relatively slow message passing approach. See (56) for a comparative study of various computers.

More recently and following the development of distributed memory computers, there is an increasing interest in the use of clusters of workstations (57) connected together by high speed networks. The trend is mainly driven by the cost effectiveness of such systems as compared to large multiprocessor systems with tightly coupled processors and memories.

Although the numerical algorithms presented throughout this work are developed for any MIMD machine, the validation and measures of performance are carried out only for distributed memory machines. A Cray T3E with 32 tightly coupled processors and a 32 PC cluster with fast ethernet based network are used for the numerical experiments. Further details of these computers may be found in the appendix.

4.1.2 Parallel programming models

The two basic models used in parallel computing are the Single Program Multiple Data (SPMD) model and the Multiple Program Multiple Data (MPMD) model. For our purpouses, i.e. CFD problems, the model most commonly used is SPMD. In this approach, each processor p runs an identical program but only computes on its own data $data_p$. To do so, the whole data or computational domain is distributed over all the processors, say np. For CFD problems, the distribution of data is done via domain decomposition, as it will be described later on. Furthermore, since the communication of data among processors may be necesary, each processor knows who is it and who are its neighbour processors nbg_p. Hence, the SPMD model is written in Alg. 19 as follows.

Algorithm 19: SPMD model

```
get a processor, an identification number from $1, .., np$ : $p$
set the neighbour processors: $ngb_p$
get a portion of data from the $np$-partitioned domain: $data_p$

compute on $data_p$
communicate with $ngb_p$
```

This model enables the programmer to write the same or different operations (both computations and communications) for each processor. This depends on the parallelization of the operations.

For example, the matrix-vector product, say $y = Ax$, where A, x and y represent the whole computational domain is partitioned into np subdomains containing the data: A_p, x_p and y_p for $p = 1.., np$. In order to perform the global product, each processor performs its own subproduct and exchanges data with its neighbour processors ngb_p where needed. Since all processors does the same this operation is summarized with independence of the processor in Alg. 20.

Algorithm 20: SPMD example: y=Ax

```
***get*** *a processor, an identification number from* $1, .., np$ *:* $p$
***set*** *the neighbour processors:* $ngb_p$
***get*** *a portion of data from the* $np$*-partitioned domain:* $data_p$
    $A_p$=partition($A, np$)
    $x_p$=partition($x, np$)
    $y_p$=partition($y, np$)
***exchange*** *data with processors* $ngb_p$
***evaluate*** *the matrix-vector product with the* $p$*-portion of the domain*
    $y_p = A_p x_p$
```

Although Alg. 20 will be explained in detail later on, it is executed at each processor doing more or less all processors the same, and hence, taking similar timings. However, this fact depends on the load of processors in both the computation and communication sense. It is worth noting that a significative delay among processors would produce bottlenecks. Hence, in order to ensure the high efficiency of the parallel algorithm, a few synchronizations, like barriers, must be introduced in strategic points and for all processors. By doing so, the processors drop the continuously unbalanced load that may arise during the computation or communication of several, say k, consecutive steps. The synchronization points are introduced implicitly in the communication steps such as the represented in Alg. 21.

Algorithm 21: Synchronization of computation and communication

```
*computation* (1)
*communication + synchronization* (1)
*computation* (2)
*communication + synchronization* (2)
    ...
*computation* ($k$)
*communication + synchronization* ($k$)
```

More precisely, the communication is composed by an exchange of data followed by a synchronization. At the exchange step, there is a pair of send and receive processes from processor p with the neighbour processors ngb_p. Once a processor has finished these tasks, it waits until the rest have done their respective tasks. This procedure is detailed in Alg. 22.

Algorithm 22: Communication + synchronization

```
communications of processor $p$
    send local data to neighbour processors $ngb_p$
    receive local data from neighbour processors $ngb_p$

synchronization with all processors
    wait for all processors
```

MPI implements this algorithm in different manners as it will be discussed in detail later on.

Although most of the parallel algorithms implemented under the SPMD model do the same operations for all processors, there are few operations that, for their implementation, require a hierarchy of processors, and hence, a different set of operations. As an example, we refer to the master-slave paradigm.

An example of the master-slave paradigm is the following. There is a master processor (often it is identified with $p = 0$) which collects the data sent from the rest of processors, named slaves (for $p = 1, \ldots, np$). The master carries out a set of operations which are considered global operations, and finally, if it is required, the final result is distributed back to all the slaves. As represented in Alg. 23, there is an idle step for the group of slave processors while the master is doing its work. Therefore, a synchronization point at the end of the algorithm must be introduced in order to drop the arising bottlenecks.

Algorithm 23: Master-slave paradigm

```
if ($p$=master)
    receive local data from $p$ =slaves
    compute master operations
    send global data to $p$ =slaves
end if

if ($p$=slave)
    send local data to $p$ =master
    keep waiting idle
    receive global data from $p$ =master
end if

synchronize master and slaves
```

The inner product of two vectors can be coded as an example of the master-slave paradigm. Although all processor compute a part of the inner product, namely local inner product, one processor (the master) collects these partial results and compute a global summation. After that, the master distributes back to the rest of processor (the slaves) the resulting value. However, there are better implementations of this operation by minimizing the number of messages and hence, improving the efficiency. Other operations that can be implemented with the master-slave paradigm are the input(read)/output (write) of global results from/to a file disk respectively.

4.1.3 Message-passing programming

On these distributed memory machines, the parallel programming model is based on explicit message-passing programming. The first MIMD machines implemented proprietary message-passing libraries, basically specifying the sending and receiving messages between processors and grouped in two categories of communication: the point-to-point communication and the group communication. Although the concepts of these proprietary libraries were equal, the no portability between platforms was not considered as ideal. For that reason, research efforts have been conducted to develop portable and standard message-passing libraries. Currently, the PVM (Parallel Virtual Machine) and more recently, MPI (Message Passing Interface) are adopted by all of the parallel computer vendors.

Since MPI is becoming de-facto standard for message passing, it has been set for the implementation and description of the communication subroutines of this work.

4.2 Performance measurements of an implementation

Once an algorithm is implemented for the parallel execution the next step is the evaluation of the parallel performance for np processors. To do so, we compare the wall clock time of the computation of the parallel implementation, denoted by $\tau(np)$ with the wall clock time of computation of the sequential version, namely $\tau(1)$. We call this rapport the efficiency $E(np)$ of the parallel implementation for np processors.

$$E(np) = \frac{\tau(1)}{np\tau(np)}$$

If a parallel implementation with np processes was ideally 100% efficient, it means a np reduction of the time of computation respect to the computation with one processor. This reduction is called the speed-up $S(np)$ and it is measured as

$$S(np) = npE(np) = \frac{\tau(1)}{\tau(np)}$$

This is the ideal case but under certain circumstances, it is also possible to obtain a super linear speed-up. This usually occurs when the sparsity of a matrix is exploited on a parallel architecture or advantageous caching occurs.

It should be noted that there is currently some debate as to which sequential time, the parallel computation should be compared to. In this work the sequential time has been evaluated from the serialized version $np = 1$ of the parallel implementations.

The efficiency and speed-up are closely related and give an indication of how well balanced the computational load is and how the problem scales. In practice, an ideal speed-up cannot be attained and there are several reasons for this, which are:

- Inherent sequential parts

- Unbalanced load
- Overhead of the communication and synchronization

The first item refers to the those parts of the algorithm that may not be perfectly parallel. For example, the inner product between two vectors and the factorization of a matrix. The second one refers to a the different distribution of load among processors. For example, the computational domain may not be equally distributed, or some operations require a master which performs more tasks than the slaves. These factors may be more or less controlled by our implementation. However, the major factor that contributes to the decrease of the efficiency of the parallel implementation is the overhead due to the exchange of data among processors. Each time a message is sent, an overhead in timing cost is incurred. Therefore, if these factors are included in the efficiency formula, it is possible to point out the drawbacks to a given implementation.

For instance, an improved measure of the efficiency is based on an accurate description of the operations and the time spent in each operation. This is in essence the measure of the sequential algorithm. For the parallel algorithm, we have to add the time spent in the communication processes needed in some operations. Then the efficiency is computed as

$$E(np) = \frac{n_o(1)\tau_o}{np(n_o(np)\tau_o + n_c(np)\tau_c(np))}$$

Where n_o stands for the number of operations, τ_o the time spent in one of these operations, $n_c(p)$ the number of communications and $\tau_c(np)$ the time spent in one of these communications. Notice that the time spent in the communication process depends on the number of processors p. More processors means more number of communications $n_c(np)$ but the quantity of data transferred per communication and processor is reduced. Furthermore it is convenient to express more accurately the time of communication τ_c split into two parts: the proper time of communication when sending a packet of data with size d at β^{-1} rate of communication and the time of setting up the communication τ_s.

$$\tau_c(np) = \tau_s + \beta d$$

Since the bandwidth β and the latency τ_s are hardware dependent parameters, the efficiency would be very different when it is evaluated in either tightly coupled processors or loosely coupled processors. Introducing these concepts in the previous expression the efficiency leads to

$$E(np) = \frac{n_o(1)\tau_o}{np(n_o(p)\tau_o + n_c(np)(\tau_s + \beta d(np)))}$$

Having a look at this expression and assuming that $n_o(1) \approx np n_o(np)$ in our algorithms, we see that the efficiency is always less than 100% and it decreasses as np increases. The number of communications $n_c(np)$ increases linearly with np, and although the time of communication of data $\tau_c(np)$ decreases because the overall data are better distributed, the latency remains constant and hence the overall time of communication per process increases.

If we partition our domain with a certain overlapping, the number of operations performed in sequential and in parallel per processor np is different. If we express the number of operations per process $n_o(np)$ in terms of the number of control volumes handled by the processor $n_{cv}(np)$ multiplied by the number of operations per control volume $o_{cv}(np)$, we get

$$n_o(np) = n_{cv}(np)o_{cv}(np)$$

and substituting it into the efficiency, we obtain an expression with three component effects: algorithm $E_a(np)$, load $E_l(np)$ and communication $E_c(np)$.

$$E(np) = E_a(np)E_l(np)E_c(np)$$

Where

$$E_a(np) = \frac{o_{cv}(1)}{o_{cv}(np)},\ E_l(np) = \frac{n_{cv}(1)}{np n_{cv}(np)},\ E_c(np) = \frac{1}{1 + \frac{n_c(np)}{n_o(np)} \frac{\tau_s + \beta d(np)}{\tau_o}}$$

The first effect means the rapport of operations performed in the sequential algorithm versus the parallel algorithm. For Krylov solvers, we consider an equal number of operations so this effect is neglected. The second effect takes account of the overlapping *ov*. Each processor contains its own data plus an overlapping. However for large scale problems the overlapping is much smaller than the data contained in the processor so this effect is reduced. The third effect is the rapport between the amount of work done of computation and communication per processor. Therefore, the efficiency not only depends on the parallel implementation but also on the parameters of communication, i.e. the latency τ_s and the bandwidth β^{-1} which are hardware dependent.

Finally, there is another performance measure so called scalability, which is related with the computing time of the problem with np and how it increase when increasing in the same ratio (e.g. doubling) both the problem size and the number of processors. It is clear that, for an ideal scalability of the implementation, the computing time will remain constant. Hence, the problem may be scaled by two, four, and so on, by means of $2np$, $4np$, ... processors.

Thus, the scalability may be computed as

$$Sc(np) = \frac{\tau_{np} size_{np}}{\tau_{np} size_1}$$

4.3 Modellization of the communication time

As mentioned above, the loss of performance of the parallel algorithm depends directly on the time spent in the communication of data between processors. A low communication cost is always desirable for high performance computing. The time of communication while sending a block of data is affected by two parameters the latency or setup time τ_s and the bandwidth or byte transferred rate per second β^{-1} . The correlation between the time of communication τ_c (given in seconds) and the transferred data d (given in bytes) was expressed in the first approximation by a linear equation:

$$\tau_c = \tau_s + \beta d$$

Indeed, both parameters depend clearly on the hardware (netcards, switch and crossbars) and the software (operating system, TCP/IP and MP library implementations, compilation and executing flags). Details about the influence of such hardware and software issues over the communication performance are out of the scope of this work.

An approximate evaluation of the latency and bandwidth parameters can be done by sending blocks of data with different sizes, and measuring the time spent in the communication (56). This procedure or test is commonly named microbenchmark of the network because it transfers small data packets that takes short time of communication. Since the measure of time can be either intrusive or do nor have enough accuracy, the time of communication is enlarged by returning the same packets of data (i.e. a round-trip of the packets) and halving the measured time. The test does a thousand of experiments by sampling the size of the packed data randomly within a range of 0-1MB and measuring the time of the round-trip. For instance, the type of data transferred is the *char*, which size in bytes is the unit.

In order to ensure non overlapped send and receive process which can reduce the time of the round-trip, the block communication mode is used. Alg. 24 summarizes this test.

Algorithm 24: Latency and bandwidth parameters

```
for (sample = 1 to sample ≤ 1000)
    set size[sample]=random(0,1000000)
    synchronize processors
        MPI_Barrier(MPI_COMM_WORLD)
        time the round-trip
            t0=MPI_Wtime()
            if (fmod(rank,2)=0)
                MPI_Send(char_send,size[sample],rank+1,..,..)
                MPI_Recv(char_recv,size[sample],rank+1,..,..)
            else
                MPI_Recv(char_recv,size[sample],rank-1,..,..)
                MPI_Send(char_send,size[sample],rank-1,..,..)
            end if
            t1=MPI_Wtime()
            time[sample]=(t1-t0)/2
end for
```

The MPI_Barrier step ensures the synchronization of the measured times and therefore reduces the dispersion of results. Furthermore, this test is executed by a wide range of even numbers of processors 2,4,6,8 obtaining results (see Fig. 4.2) without significative differences.

This figure shows the linear behaviour of the communication time of the transferred pack of bytes. The difference of slopes for each facility indicates different byte rates per second. For the estimation of the slope, the range of large messages (1KB,..,1MB) has been used. And the inverse of the slope is the estimated bandwidth.

For the estimation of the latency parameter, there are two possibilities. The first one is taking the time value for a zero pack of bytes. But as mentioned before, time measure may be inaccurate. In a second approach, a zoom (see Fig. 4.1) near the short transferred pack of bytes (0,..,1KB) is used with a linear fit regression. The latency parameter is then estimated for a zero pack of bytes. The different values of latency and bandwidth parameters for both facilities are summarized in the table 4.1.

Facility	Latency (1) μsec	Latency (2) μsec	Bandwidth MB/sec
PC cluster	226.98	218.95	10.46
Cray T3E	23.17	21.18	119.98

Table 4.1: Latencies measured (1) and estimated (2) and bandwidth estimated.

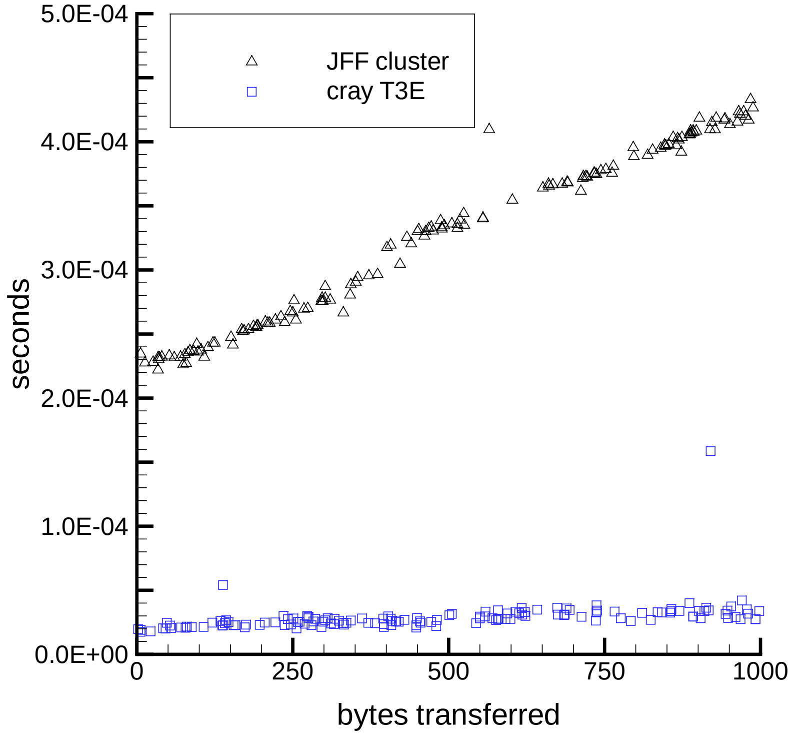

Figure 4.1: Zoom of previous figure near the short transferred pack of bytes.

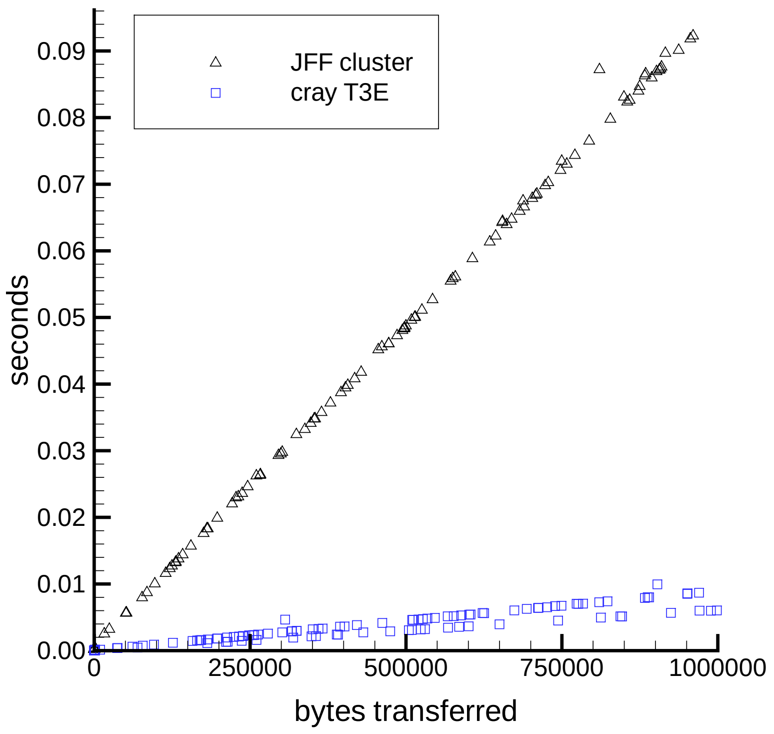

Figure 4.2: Timing results on two facilities the PC cluster and the Cray T3E.

These values clearly show the great difference of communication performance between facilities. The low latency and large bandwidth of the Cray T3E explain in part the high price against the PC cluster.

4.4 Communication modes

The exchange of data among processors can be performed over two different modes of communication: blocking and non blocking. The blocking communication mode disables the processors to perform other operations while the communication is being done. Since the process of sending or receiving the data involves the access to the buffer of memory, MPI library protects the data by blocking the processor and hence, avoiding the access to the data for reading or writing. The blocking communication mode among processor p and its neighbours ngb_p for a given buffer of data, say $data_p$, is described in Alg. 25.

Algorithm 25: Blocking communication

```
send $data_p$ to neighbours $ngb_p$
    block_send $(data_p, ngb_p)$
receive $data_p$ from neighbours $ngb_p$
    block_receive $(data_p, ngb_p)$
```

Since the communications in this mode follow the order of the algorithm, processors either the sender or the receiver keep idle waiting for completion of the communication processes, and hence, resulting in poor efficiency. However this mode has an implicit synchronization procedure so it is not necessary to explicitly synchronize the processors after the ends of the pairs of send and receive processes.

The order of the communication processes among processors has to be considered carefully. A very important fact in a bad scheduled communication is the dead lock. It appears when two or more processors send and receive data in a wrong order. When this happens, the processors do not understand themselves and keep waiting infinitely for the communication request. The dead lock may be avoided by organizing the communication in two simple steps. Firstly, all procesors send their respective data to their respective receivers, i.e., the neighbour processors. Any order for the senders could be defined. Secondly, the senders are ready to receive the data from their respective receivers. The dead locks are avoided by ensuring the existence of pairs of send and receive messages. If not, it will appear to have an orphan message process (a sender without a receiver or a receiver without a sender) and thus resulting into a dead lock. The use of tags is convenient for the right schedule of sending and receiving data.

The other mode of communication to be used in data exchange is the non blocking communication. Each processor copies the data to communicate x_p to another buffer y_p. Then MPI library acts on that data without blocking the processors to perform other operations on the original buffer. The non blocking communication for the same vector is described in Alg. 26.

Algorithm 26: Non blocking communication (x_p)

```
***copy*** *data from* $x_p$ *into another buffer* $y_p$
***send*** *data of* $y_p$ *from* $p$ *to neighbours* $ngb_p$
    *non_block_send* $(y_p, ngb_p)$
***operate*** *over* $x_p$
    ...
***receive*** *data in* $y_p$ *from neighbours* $ngb_p$ *to* $p$
    *non_block_receive* $(y_p, ngb_p)$
***operate*** *over* $x_p$
    ...
***synchronize*** *all processes*
    ***wait*** *for all*
***copy*** *data from* $y_p$ *into the buffer* $x_p$
```

However, this procedure could reduce the performance of parallel algorithms due to the elapsed times of the pre step of copying data from the buffer x_p to the buffer y_p and, after the communication step, the post step of copying back the data to the original buffer. In addition, for large amounts of data, the performance of buffering decreases due to the cache missings.

Furthermore, the data in x_p to be exchanged, e.g. halos and inside blocks, are usually located in non contiguous locations of the buffer. Therefore the data must be arranged continuously in the buffer y_p before any communication. After the communication is done, the data contained in the buffer y_p must be redistributed to x_p in the non contiguous locations.

In order to see the differences of performance of both modes of communication, a second test is done. This test (see Alg. 27) shows the performance of the implementation of an exchanged pack of bytes.

Algorithm 27: Blocking and non-blocking communication modes

```
for (mode=blocking to mode=non-blocking)
    for (sample=1 to 1000)

        size[mode][sample]=random(0,1000000)

        MPI_Barrier(MPI_COMM_WORLD)

        t0=MPI_Wtime()
        if (mode=blocking)
            if (fmod(rank,2)=0)
                MPI_Send(char_send,size[mode][sample],rank+1,..,..)
                MPI_Recv(char_recv,size[mode][sample],rank+1,..,..)
            else
                MPI_Recv(char_recv,size[mode][sample],rank-1,..,..)
                MPI_Send(char_send,size[mode][sample],rank-1,..,..)
            end if
        end if

        if (mode=non-blocking)
            if (fmod(rank,2)=0)
                MPI_Isend(char_send,size[mode][sample],rank+1,..,..)
                MPI_Irecv(char_recv,size[mode][sample],rank+1,..,..)
            else
                 MPI_Isend(char_send,size[mode][sample],rank-1,..,..)
                 MPI_Irecv(char_recv,size[mode][sample],rank-1,..,..)
            end if

            MPI_Wait(send)
            MPI_Wait(recv)

        end if
        t1=MPI_Wtime()
        time[mode][sample]=t1-t0

    end for
end for
```

Regarding the blocking communication mode, it performs the send and the receive processes of a given pack of bytes consecutively. It is like the round-trip of the pack described in the previous test, but the measure of time is not halved.

If the network enables communications among processors in both senses at same time (i.e. duplex network), a pair of send and receive processes can be done in half of time. Conversely, the non-blocking communication mode enables one to do this pair of processes at same time and to wait until it completes the two processes independently.

Like in the previous test, a thousand of experiments with randomized packet sizes of type *char* and covering the range of (0,..,1MB) is done.

Due to the duplex network feature, the order of the non-blocking communication processes within the algorithm does not affect the communication procedure and therefore no dead-locks could arise. The scope of the MPI_Wait step is to wait for termination of communication processes.

The test is carried out, analogously to the previous test, on both facilities PC cluster and Cray T3E, whose networks are duplex. The results are shown in Fig. 4.3.

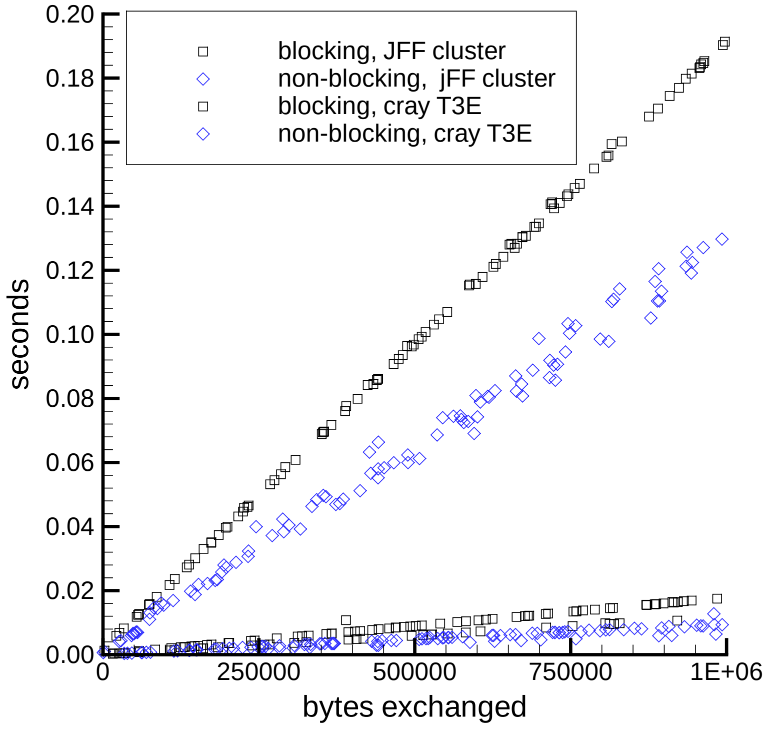

Figure 4.3: Blocking and non-blocking communication modes for the PC cluster and the Cray T3E.

Fig. 4.3 shows that the behaviour of both communication modes are similar in both machines but at different scales of time. The non-blocking communication of any pack of bytes is performed more efficiently and nearly at half time of the the time taken by the blocking communication. The byte exchanged rates (i.e. the inverse of the slopes), for each implementation and their rapports are given in table 4.2.

Therefore, it is convenient to use a non-blocking communication implementation for those operations where an exchange of information is required such the matrix-vector product. For that reason, MPI library provides a set of data types which enables to skip these pre and post copying steps, and hence, to improve the communication performance.

modes	PC cluster	Cray T3E
blocking (1) MB/sec	5.255	60.321
non-blocking (2) MB/sec	7.889	110.387
rapport: (2)/(1)	0.66	0.54

Table 4.2: Megabyte exchanged rates (MB/sec) and rapports for both modes of communication on the PC cluster and the Cray T3E.

4.5 Domain decomposition

The most popular approach to solving CFD problems on MIMD architectures whether it is memory shared or memory distributed is that of the domain decomposition (58). The objective is to distribute the computational domain onto a number of processors such that the work load on each processor is equivalent. The practical application involves the discretization and the solution of the systems of equations derived in previous chapters on an equally distributed load per processor. The details of the implementation of the domain decomposition are given hereinafter from an algebraic point of view.

Most algebraic operations involved in the discretization and solution procedures are based on the addition, the subtraction and the product of three types of objects: scalars, vectors and matrices. The implementation of these operations on these objects can be thought to be performed either in sequential or in parallel. In a one-processor machine the operation and the storage of objects are done as it is expressed mathematically. However, in a MIMD machine, the algebraic operations are performed on a part of the objects. The vector object, for example, is equally distributed as much as possible and stored among all processors. This equally distribution of objects is stressed to balance the load, and hence, to obtain a good efficiency of the implementation.

From here on, we shall use subindex for distinct parts of the objects. If we have a np-processor machine where np stands for the number of processors and object, named o, the object is partitioned into np objects and stored at each processor. Labeling the resulting objects from $p = 1$ to $p = np$ it follows that

$$o = \bigcup_{p=1,\ldots,np} o_i$$

An algebraic operation, named for generality $\oplus$, between two objects x and y is performed in each process p using only the respective parts x_p and y_p. The result can be stored in another distributed object z.

$$z = x \oplus y \Longleftrightarrow z_p = x_p \oplus y_p,\ p = 1, \ldots, np$$

Thinking this way, it seems easy to implement parallel $np > 1$ or sequential $np = 1$ algebraic operations with distributed objects. Nevertheless, some operations like the inner product between two vectors or the 2-norm of a vector, the maximum and minimum value of the components of a vector, and the product of a matrix with a vector involve information stored in the closest processors or even in all processors. Therefore, this information must be copied from these processors, named neighbour processors ngp_p to the affected processor p. This yields to an extra information per processor of the object.

$$ov \neq \bigcap_{p=ngb_p} o_p$$

Hereinafter, we shall call this additional information as the overlapping values ov. Most operations requires an overlapping of a single point, one line of points or a surface of points for a one, two or three dimensional object respectively. The implementation of these ideas to vectors and matrices are developed in next subsections.

4.5.1 Block vector

Let us suppose x to be a vector object which maps a d-dimensional domain Ω. The partition of this domain in np parts is carried out in an equally distributed manner among the np processors. Assuming that the domain Ω is discretized onto a structured grid of points, the partition is performed easily following the orthogonal directions. For instance, a vector that maps a three dimensional domain (see Fig. 4.4) is partitioned in two orthogonal directions $p_1 = 2$ and $p_2 = 3$ leading into 6 block vectors x_p with $p = 1, \ldots, np = p_1 \times p_2$.

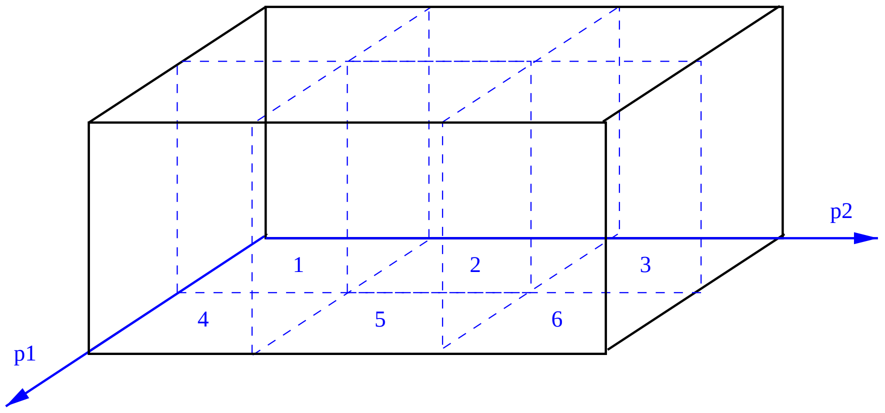

Figure 4.4: Two dimensional partition of a vector that maps a three dimensional domain.

Let $I \times J \times K$ be the overall grid points over the domain Ω, the block vector x_p maps the p-part of the domain at least in $(I/p_1) \times (J/p_2) \times K$ grid points. Notice that, a perfect load balancing in all directions is considered. An unbalanced partition leads into a generalized grid size $id \times jd \times kd$. Fig. 4.5 shows the mapping of x_p with a generalized index i, j, k.

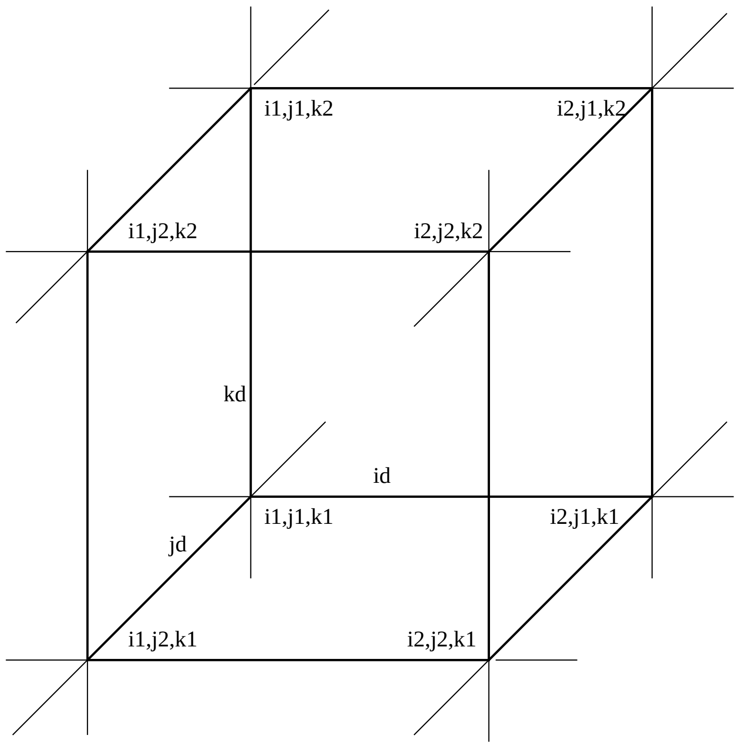

Figure 4.5: Generalized $id \times jd \times kd$ partition x_p of a three dimensional vector x. The index i, j, k has a range from $i1, j1, k1$ to $i2, j2, k2$.

The overlapping *ov* is added in those directions where the information of the neighbour processors is needed. The blue dashed lines in Fig. 4.6 show the overlapping areas among processors.

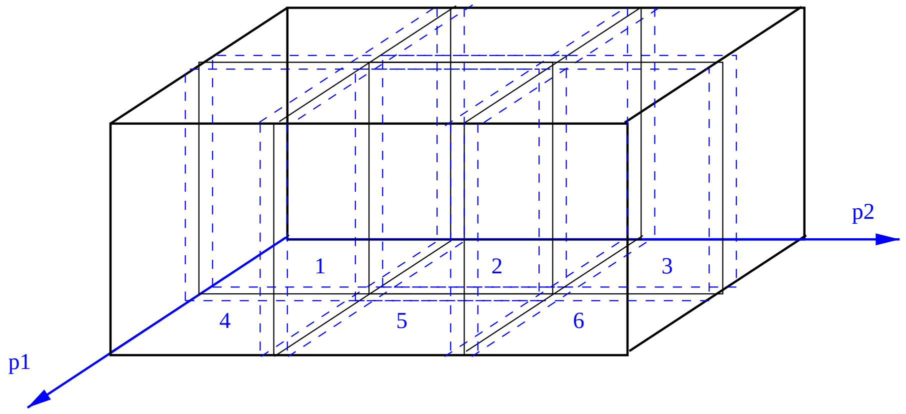

Figure 4.6: Overlapping areas *ov* for a 2×3 partitioned vector that maps a three dimensional domain.

The resulting dimension of a generalized x_p vector with grid size $id \times jd \times kd$ and with the same overlapping ov in all directions is represented in Fig. 4.7.

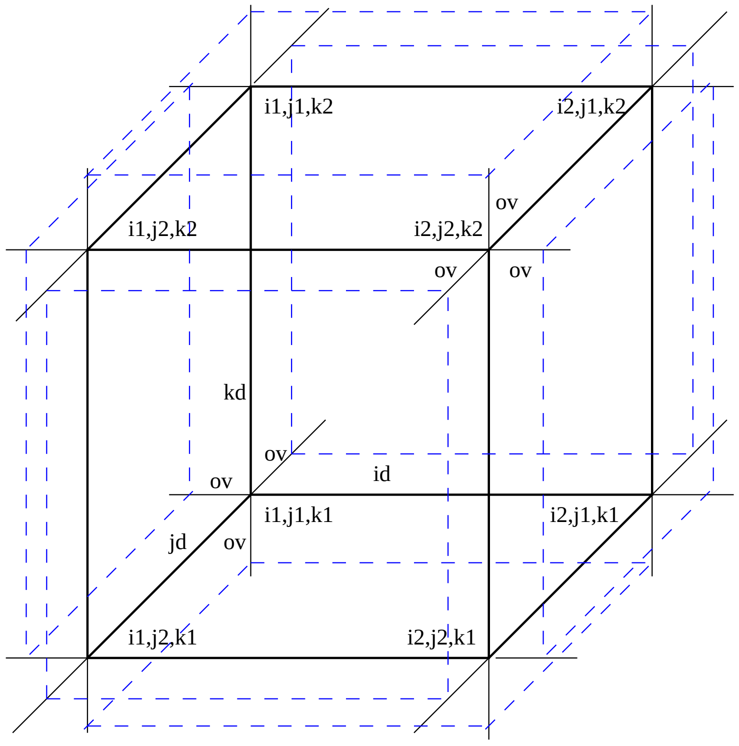

Figure 4.7: Overlapping areas ov for a generalized $id \times jd \times kd$ partition x_p of a three dimensional vector x.

Therefore, the three dimensional vector x that represents the domain Ω is expressed in terms of a set of np partitioned vectors x_p with a defined size $id \times jd \times kd$ plus an overlapping of ov. For practical implementation reasons, this overlapping is added to all block vectors and in all directions whether there are neighbour processors or not.

In order to gain clarity of the algebraic representation of the full vector x, the overlapping information is omitted yielding to a representation of the vector as follows

$$x = (x_1, x_2, \ldots, x_{np-1}, x_{np})^T$$

By doing so, it is easy to represent the operations with vectors. For instance, the copy of a vector x into another vector y is represented by omitting the overlapping as

$$y = x \Longleftrightarrow \left\{ \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_{np-1} \\ x_{np} \end{array} \right\} = \left\{ \begin{array}{c} y_1 \\ y_2 \\ \vdots \\ y_{np-1} \\ y_{np} \end{array} \right\}$$

However, the implementation of this operation considers the overlapping. Alg. 28 gives such example considering an overlapping of ov.

Algorithm 28: Copy_vect(x_p, y_p)

```
for ($i = i1 - ov$ to $i = i2 + ov$)
    for ($j = j1 - ov$ to $j = j2 + ov$)
        for ($k = k1 - ov$ to $k = k2 + ov$)
            $y_p(i, j, k) = x_p(i, j, k)$
        end for
    end for
end for
```

4.5.2 Block matrix

Let A be a N-square matrix such that $N = (I \times J \times K)$ arises from the discretization of the governing equations of a CFD problem onto a grid of $I \times J \times K$ points. We represent the matrix A partitioned by rows as balanced as possible in np blocks of matrices A_p:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{np-1} \\ A_{np} \end{bmatrix}$$

Each of these matrices A_p contains the matrix coefficients of the linear system $Ax = b$ associated with the partition p.

$$\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{np-1} \\ A_{np} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{np-1} \\ x_{np} \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{np-1} \\ b_{np} \end{Bmatrix}$$

In previous chapter, we described the different patterns of these sparse matrices for a natural ordering (i.e. the *7-point* and the *19-point* formulation in a three dimensional CFD problem. Since a narrowed band of coefficients are non zero, we shall store only the blocks of non zero coefficients of each partition.

Here, we used the following notation for a one dimensional partition:

$$\begin{bmatrix} A_1 & A_{1,2} & 0 & \cdots & & \\ A_{2,1} & A_2 & A_{2,3} & 0 & \cdots & \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ & \cdots & 0 & A_{np-1,np-2} & A_{np-1} & A_{np-1,np} \\ & & \cdots & 0 & A_{np,np-1} & A_{np} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{np-1} \\ x_{np} \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{np-1} \\ b_{np} \end{Bmatrix}$$

Having a look at the structure of this matrix each process p stores two different matrices A_p and A_{p,ngb_p}. Where ngb_p stands for the neighbour processes of process p. The first matrix indicates the operations performed with the information stored within the processor p and the second one indicates the operations performed with the information stored at neighbour processors.

For a generalized partition in two or three directions, the structure of blocks of matrix A the linear system is written as follows:

$$\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{np-1} \\ A_{np} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{np-1} \\ x_{np} \end{Bmatrix} + \begin{bmatrix} A_{ngb_1} \\ A_{ngb_2} \\ \vdots \\ A_{ngb_{np-1}} \\ A_{ngb_{np}} \end{bmatrix} \begin{Bmatrix} x_{ngb_1} \\ x_{ngb_2} \\ \vdots \\ x_{ngb_{np-1}} \\ x_{ngb_{np}} \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{np-1} \\ b_{np} \end{Bmatrix}$$

Thus, a set of communications between the processes involved in such operations must share the information of the ngb_p process to the p process. In the chapter ahead, we shall explain some issues of this communication between the different processes.

4.6 Exchange of data blocks

MPI data type has been used to send and receive at once the non contiguous data y_p of a vector x_p. MPI data type provides a new type of variables in order to send and receive blocks of information located in different points of the buffer at once. By doing so, we can reduce, on one hand, the number of communications and latency, and on the other, the number of cache missings in the buffering processes. A sender processor can explicitly pack noncontiguous data into a contiguous buffer and then send it. A receiver processor can explicitly unpack data received in a contiguous buffer and store in noncontiguous locations.

For simplicity, let us suppose that a x_p vector has dimensions 4×4 and the noncontiguous data y_p has dimensions 2×3 (see Fig. 4.8).

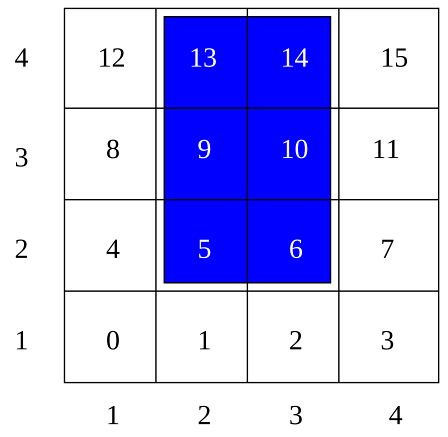

Figure 4.8: The 4×4 x_p vector and the 2×2 y_p data filled in blue. The numbers written within the x_p vector represent the order of data in the buffer.

A schematic representation of the buffer (see Fig. 4.9) shows the noncontiguous data y_p embedded in the map x_p.

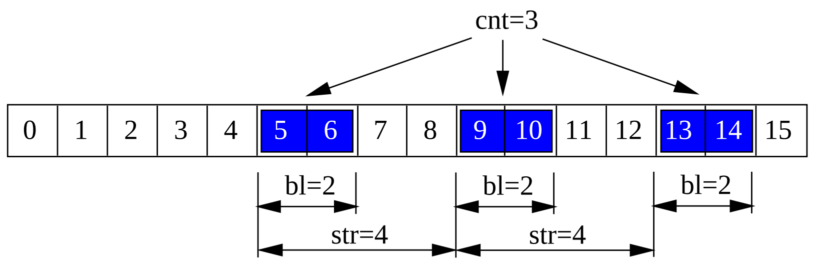

Figure 4.9: Buffer representation of x_p and the noncontiguous data y_p. The noncontiguous data follows a pattern of $cnt = 3$ noncontiguous blocks of length $bl = 2$ separated with an stride of $str = 4$.

Notice that the noncontiguous data follows a pattern of 3 noncontiguous blocks of dimension 2 separated with an stride of 4. This information is enough to construct a new data type with continuous data by packing the y_p data.

This idea has been easily extended to more complex noncontiguous data such as a three dimensional vectors, with two different number of blocks with different lengths and strides. Fig. 4.10 shows a three dimensional x_p vector with dimensions $I \times J \times K$ and a $i \times j \times k$ noncontiguous data y_p.

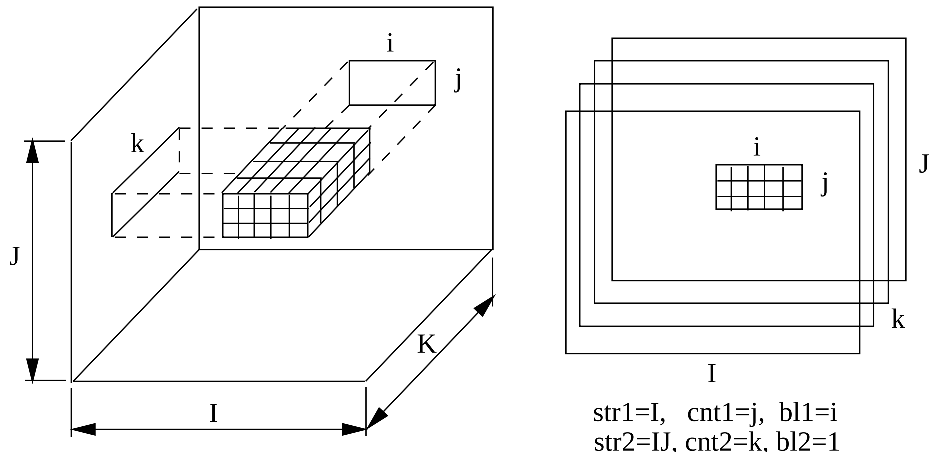

Figure 4.10: Representation of x_p vector and the noncontiguous data y_p. The noncontiguous data follows a pattern of two noncontiguous blocks cnt_1, cnt_2 with different lengths bl_1, bl_2 separated with strides str_1 and str_2 respectively.

In this case, we apply recursively two data types. The first one has j blocks of length i with an stride of I. The second one put over the first one, has k blocks of length 1 with an stride of IJ.

Finally, the data type implementation over a non blocking communication leads to Alg. 29, used in the product of a matrix by a vector.

Algorithm 29: Update (ov, x_p)

```
set the data type $y_p$ for noncontiguous data of $x_p$
    data_type $(ov, x_p, y_p)$
send $y_p$ data from $p$ to neighbours $ngb_p$
    non_blocking_send $(y_p, ngb_p)$
compute over $x_p$

    ...

receive $y_p$ data from neighbours $ngb_p$ to $p$
    non_blocking_receive $(y_p, ngb_p)$
compute over $x_p$

    ...

synchronization of all process
    wait for all
```

Further details on data types can be found in MPI documentation available in internet (7).

The following test measures and compares the communication performance of two possible implementations of the *update*(ov, x_p) subroutine. Both implementations use the non-blocking mode of communication which has been tested to be the better. Furthermore, both implementations perform a copy of values to be exchanged to a buffer and then they are exchanged. By doing so, it is possible to perform operations with the original values and exchange a copy at same time. Therefore, this procedure increases the performance of operations because it overlaps the computation and the communication. A draft (see Fig. 4.11) of the processes shows this procedure.

However the implementation of the buffering process affects this overlapping. The original buffer contains the amounts of data in non-contiguous blocks and the copied buffer must contains the data in contiguous blocks before they are sent to another process. This process is called packing. Conversely, after the communication is done, the received pack of data contained in a contiguous buffer has to be unpacked at non-contiguous locations in the original buffer of the receiver process.

In order to reduce the time of the whole procedure of exchange of data, an efficient implementation of the packing and unpacking procedures is desired. A first implementation does an explicit copy of the non-contiguous blocks of data in a contiguous buffer. This implementation is tedious to implement: dynamic allocations, initializations and copying data from one buffer to the other. In addition, it may suffer possible overheads due to the cache missings. This implementation is called simply non-blocking.

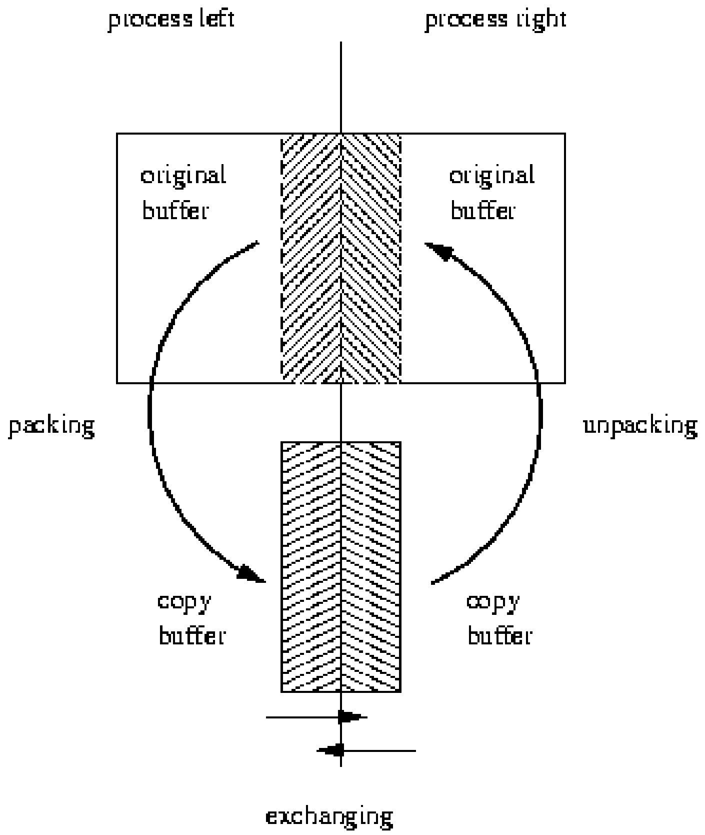

Figure 4.11: Pack, exchange and unpack procedure between two processes

A second and easier implementation use the MPI_datatype to do implicitly the copy from the original and non-contiguous buffer to a contiguous buffer, thus reporting coding and time savings. This implicit copy means that only a structure of pointers to the different locations of the non-contiguous blocks are stored in a MPI_datatype. After that, the MPI library performs the packing and unpacking processes implicitly and in an efficient way when the non-blocking send and receive subroutines are called.

Differences of both implementations are tested as follow. A three dimensional halo with random size in k direction is updated between two processes, left and right. It has been chosen to vary the k direction instead of i or j since the distribution of non-contiguous blocks is sparser and may produce more cache missings. See Fig. 4.12 for details of block data sizes and graphical explanation of the communication process.

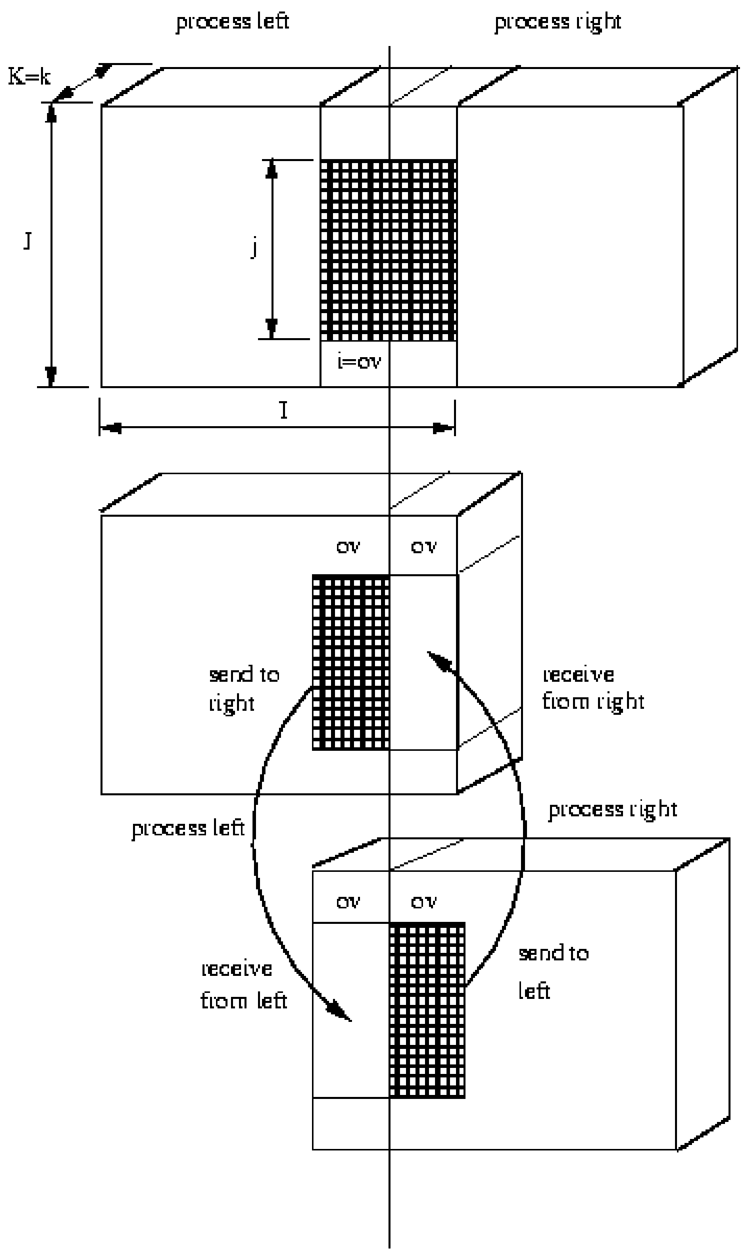

Figure 4.12: Update of *ov* halo between processes left and right.

The test is run at PC cluster for three different halos: ov=1, ov=2 and ov=4, which are often used in the algebraic operations with communications. For each size of halo, two measures of times are taken. The time of packing and unpacking, called pack, and the time of communication, called comm. These results and the overall time (i.e. the addition of these quantities) are presented for both implementations.

A comparison of all cases is given in Fig. 4.13:

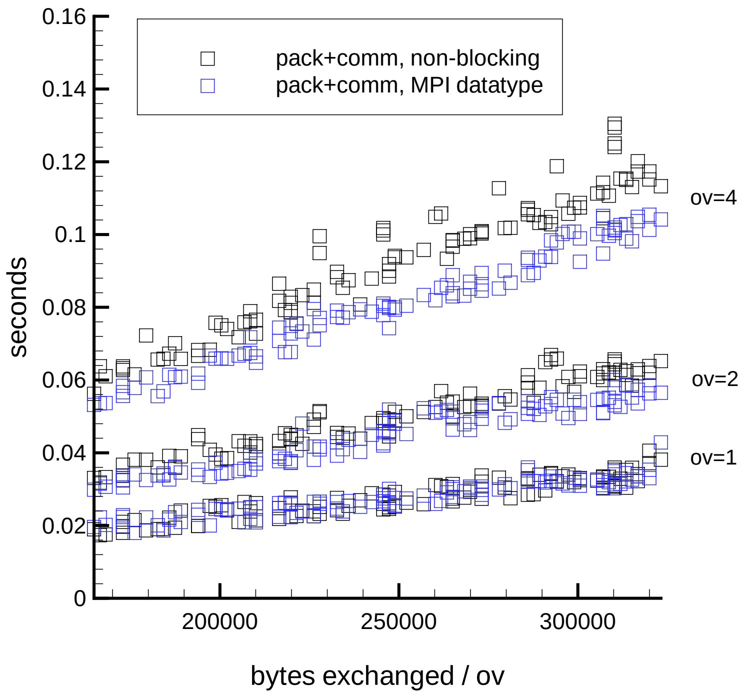

Figure 4.13: Comparison of both implementations for all halo sizes.

For each size of halos, the times of packing and unpacking of both implementations are very different. The details of each case are reported in Figs. 4.14, 4.15, 4.16, 4.17, 4.18 and 4.19.

The efficiency of the MPI_datatype implementation at PC cluster is based on the cache optimization for non-contiguous blocks when they are packed and unpacked. In addition, the packing and unpacking processes of this implementation have to be done only once because it only points to the non-contiguous data. Conversely, in the first implementation, the packing and unpacking processes are done at each exchange thus it is an explicit copy. So the performance of the overall communication is poor.

Although the results of Cray T3E has not been presented here, the difference of both implementations are not meaningful. We guess that this fact is due to the special cache built-in the cpu (see hardware issues in Appendix).

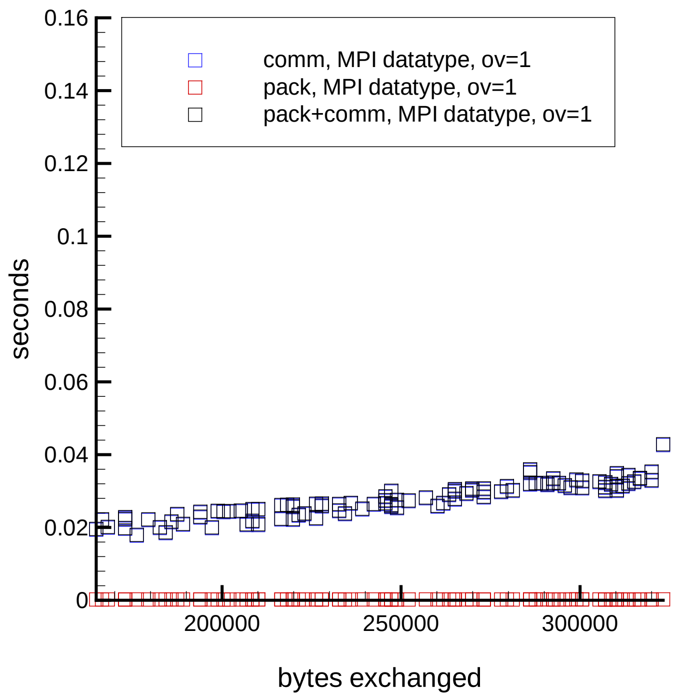

Figure 4.14: Update of $ov = 1$ with MPI_datatype.

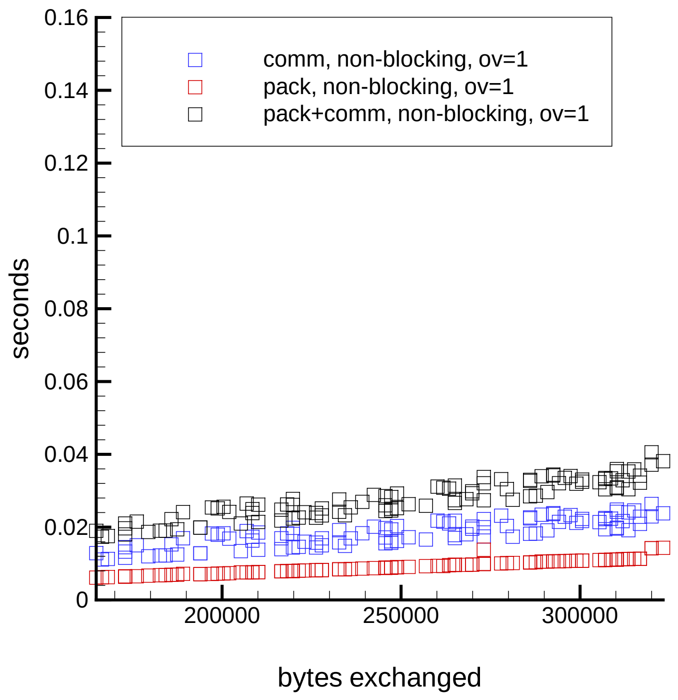

Figure 4.15: Update of $ov = 1$ with explicit copy.

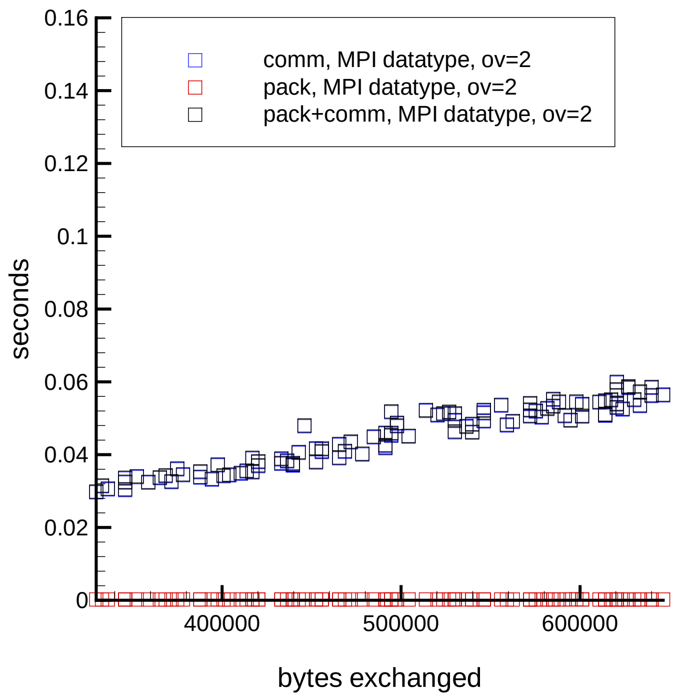

Figure 4.16: Update of $ov = 2$ with MPI_datatype.

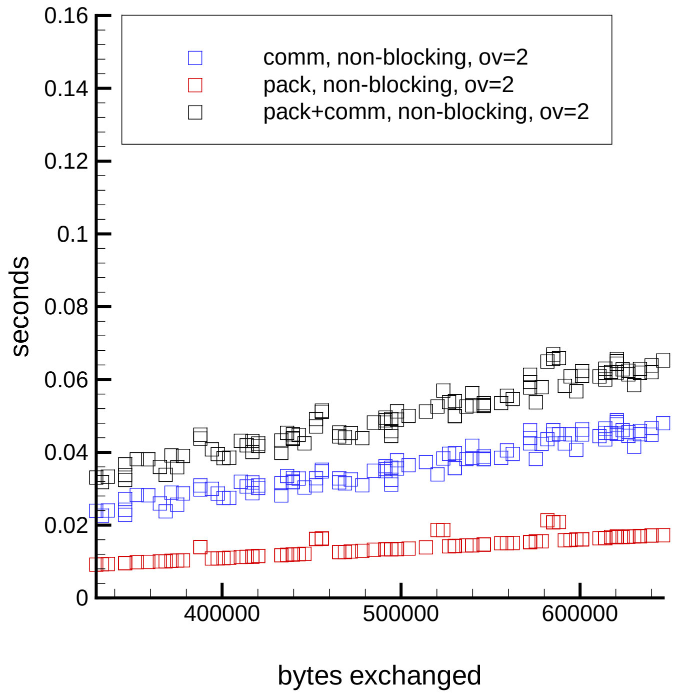

Figure 4.17: Update of $ov = 2$ with explicit copy.

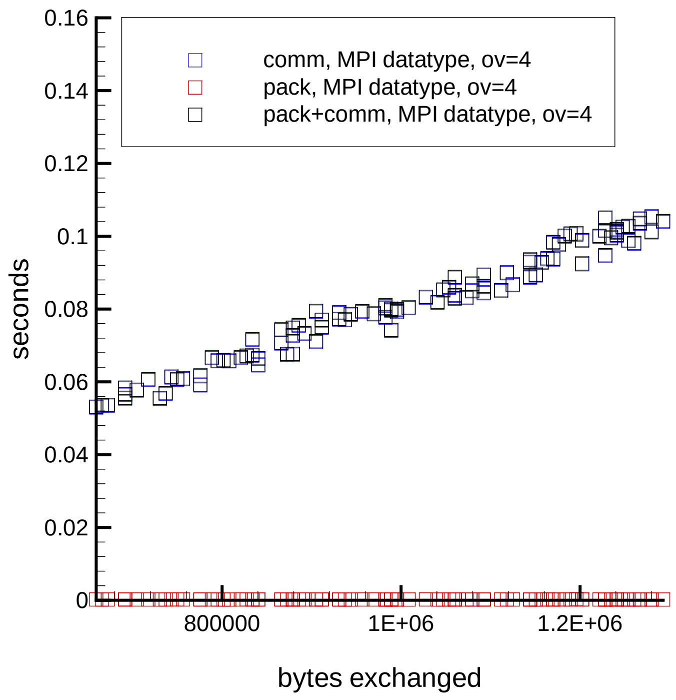

Figure 4.18: Update of $ov = 4$ with MPI_datatype.

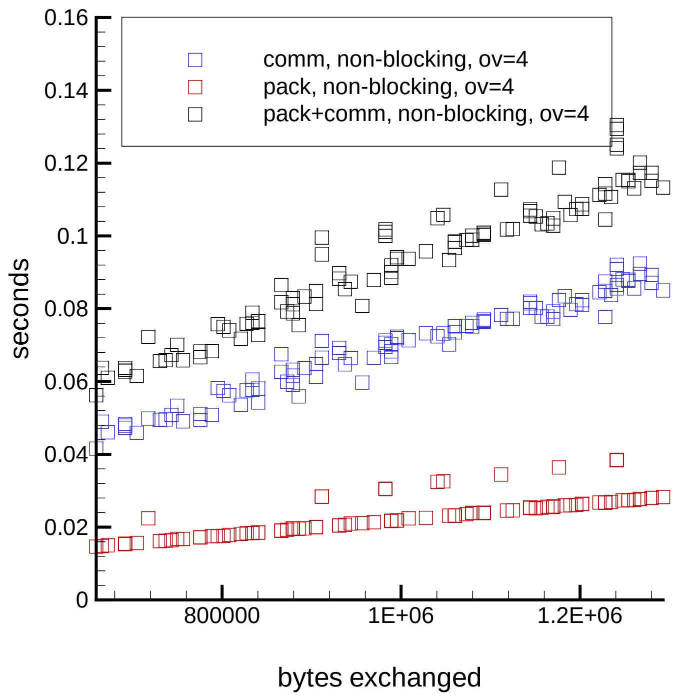

Figure 4.19: Update of $ov = 4$ with explicit copy.

4.7 Algebraic operations with vectors and matrices

Three types of algebraic operations are detailed: those without communication between processors, so the parallel efficiency is 100%, those operations that combine computation and communication, and those operations where most part of the job is the communication.

4.7.1 Addition, difference and scaling of vectors

The addition or the difference of two vectors x and y is stored in a third vector z. The algorithm (see Alg. 30) that represents any of these operations may be written as

Algorithm 30: operation_vect(x_p, y_p, z_p)

```
for $(i = i1 \text{ to } i = i2)$
    for $(j = j1 \text{ to } j = j2)$
        for $(k = k1 \text{ to } k = k2)$
            $z_p(i,j,k) = x_p(i,j,k) \pm y_p(i,j,k)$
        end for
    end for
end for
```

Another 100% parallel algebraic operation is the vector scaling (see Alg. 31). A vector x can be scaled by a real value α leading to a vector z.

Algorithm 31: Scal_vect(x_p, α, z_p)

```
for $(i = i1 \text{ to } i = i2)$
    for $(j = j1 \text{ to } j = j2)$
        for $(k = k1 \text{ to } k = k2)$
            $z_p(i,j,k) = \alpha x_p(i,j,k)$
        end for
    end for
end for
```

Notice that we have not considered the overlapping area in the computation of the vector z. This fact reduces the number of floating point operations, and it produces a reduction of time of the computation. Furthermore, it is possible to perform these operations reusing any vector, e.g., $x = x \pm x$, $x = x \pm y$ and $x = \alpha x$, and hence, reducing storage requirements.

4.7.2 Saxpy operation

The name of the subroutine saxpy (59) comes from the scientific literature and it represents a composition of two previous types of operations.

$$z = x + \alpha y$$

Although it can be implemented in two steps by means of the above algebraic operations, it has been packed in one step. Alg. 32 also enables the reuse of any vector.

Algorithm 32: Saxpy(x_p, α, y_p, z_p)

```
for ($i = i1$ to $i = i2$)
    for ($j = j1$ to $j = j2$)
        for ($k = k1$ to $k = k2$)
            $z_p(i,j,k) = x_p(i,j,k) + \alpha y_p(i,j,k)$
        end for
    end for
end for
```

4.7.3 Inner product of vectors

The computation of the inner product of two vectors is one of the most important keys in the parallel efficiency of most solvers, because it involves a global communication between all processors. Let

$$\rho =< x, y >$$

be the inner product of two vectors, it is performed in two steps (see Alg. 33). It starts with the inner product of each pair of sub vectors x_p and y_p where $p = \{1, 2, \ldots, np\}$. The resulting set of np inner products is stored in an auxiliar variable ρ_p. Then a global sum of these values is performed and shared to all the processors by means of a global communication, so there is a fraction of time spent on the communication, and hence a lose of efficiency.

Algorithm 33: Inner_product(x_p, y_p, ρ)

```
**evaluate** *inner product of vectors* $x_p$ , $y_p$
    $\rho_p = 0$
**for** $(i = i1$ *to* $i = i2)$
    **for** $(j = j1$ *to* $j = j2)$
        **for** $(k = k1$ *to* $k = k2)$
            $\rho_p = \rho_p + x_p(i,j,k) y_p(i,j,k)$
        **end for**
    **end for**
**end for**
**evaluate** *global summation of* $\rho_p$
    *global_sum* $(\rho_p, \rho)$
```

Notice that Alg. 33 enables to compute the 2-norm of a vector x (see Alg. 34).

The inner product operation contains a global communication that broadcast each sub inner product to the rest of processors. After that, a sum of all values is performed in each processor. The implementation of this broadcast plus the summation relays on the MPI library and it is performed in the MPI_Allreduce subroutine.

Algorithm 34: Norm_vect(x_p, ρ)

```
**compute** *the inner product of* $x$ *with itself.*
    *inner_product*$(x_p, x_p, \rho_0)$
**evaluate** *the* 2*-norm of* $x$
    $\rho = \sqrt{\rho_0}$
```

Since the number of messages and data does not depend on the partitioning configuration, differences of the speed-ups are due to the differences in computation. Therefore, only the cache missing effects may arise for some configurations for a given case with large size. For instance, the size of vectors are $20 \times 20 \times 20$, $40 \times 40 \times 40$, $60 \times 60 \times 60$, $80 \times 80 \times 80$ and $100 \times 100 \times 100$.

The test is executed on both facilities, PC cluster and Cray T3E within the range of 1 to 12 processors and for different partitioning directions (see Fig. 4.20).

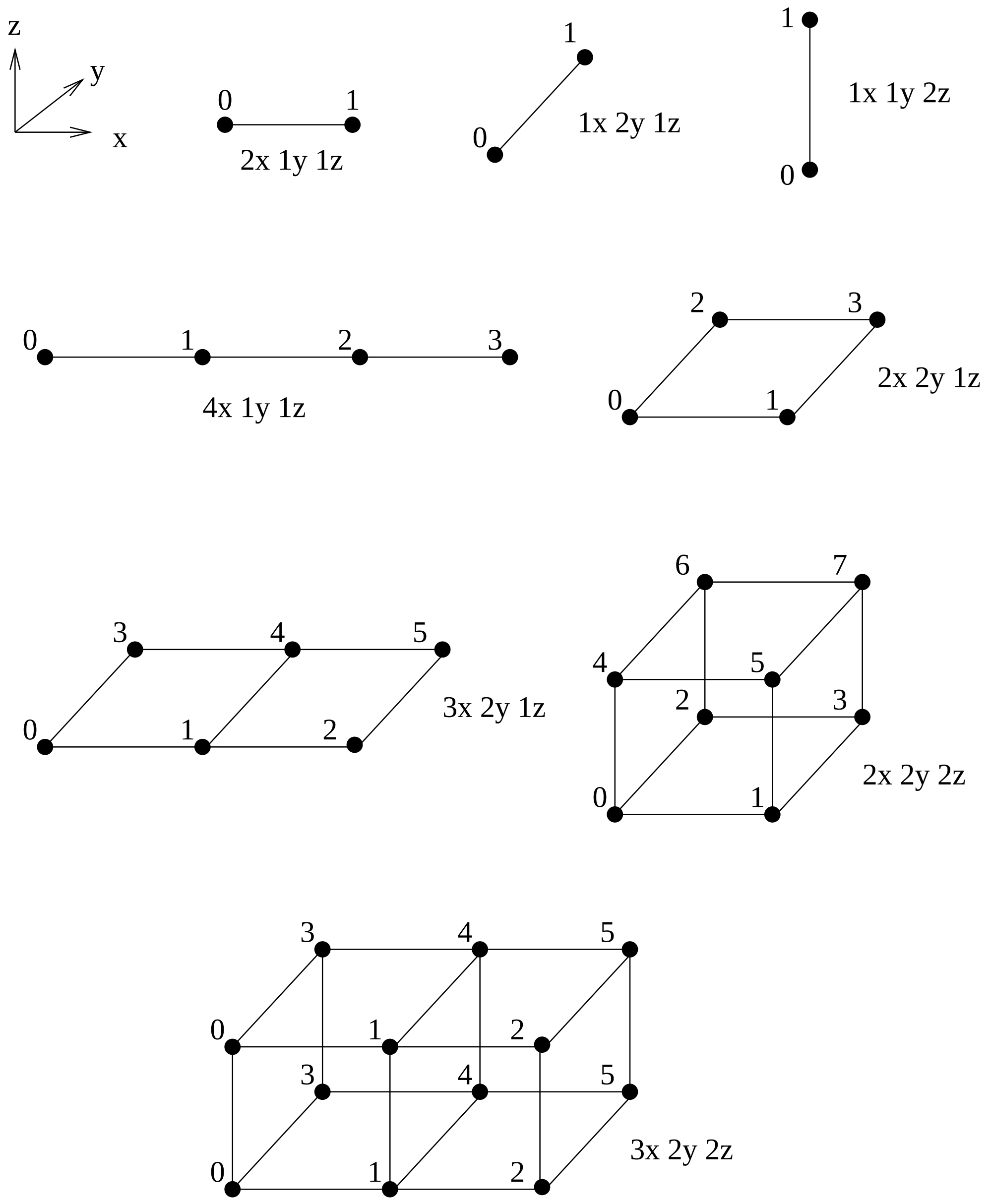

Figure 4.20: partitioning directions that yield different topologies of processors: (line, plane and hexahedron) with 2, 4, 6, 8 and 12 processors.

The results are represented in terms of the speed-up in Figs. 4.21,4.22 and tables 4.3,4.4 for PC cluster and the Cray T3E respectively.

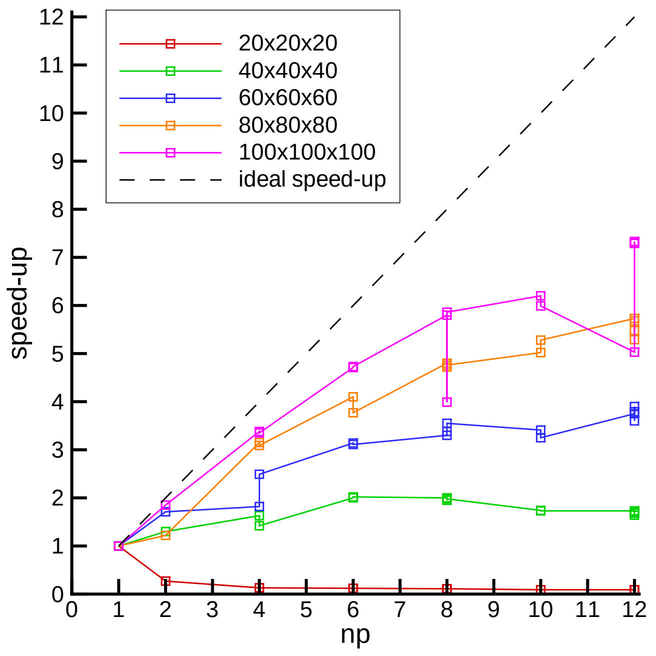

Figure 4.21: Speed-up for the inner product of 3D vectors in PC cluster.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$	$100 \times 100 \times 100$
1	1x1y1z	1.00	1.00	1.00	1.00	1.00
2	1x1y2z	0.27	1.30	1.71	1.22	1.85
4	1x1y4z	0.13	1.63	1.82	3.15	3.38
4	1x2y2z	0.13	1.42	2.49	3.09	3.35
6	1x1y6z	0.12	2.00	3.14	4.10	4.71
6	1x2y3z	0.12	2.02	3.11	3.77	4.73
8	1x1y8z	0.11	2.00	3.30	4.80	5.80
8	1x2y4z	0.11	1.95	3.38	4.72	3.99
8	2x2y2z	0.11	1.98	3.55	4.76	5.86
10	1x1y10z	0.09	1.74	3.41	5.02	6.20
10	1x2y5z	0.09	1.73	3.25	5.28	5.99
12	1x1y12z	0.09	1.73	3.75	5.73	5.03
12	1x2y6z	0.09	1.71	3.90	5.48	7.29
12	1x3y4z	0.09	1.68	3.79	5.68	7.30
12	2x2y3z	0.09	1.64	3.60	5.29	7.33

Table 4.3: Speed-up of the inner product of 3D vectors in the PC cluster.

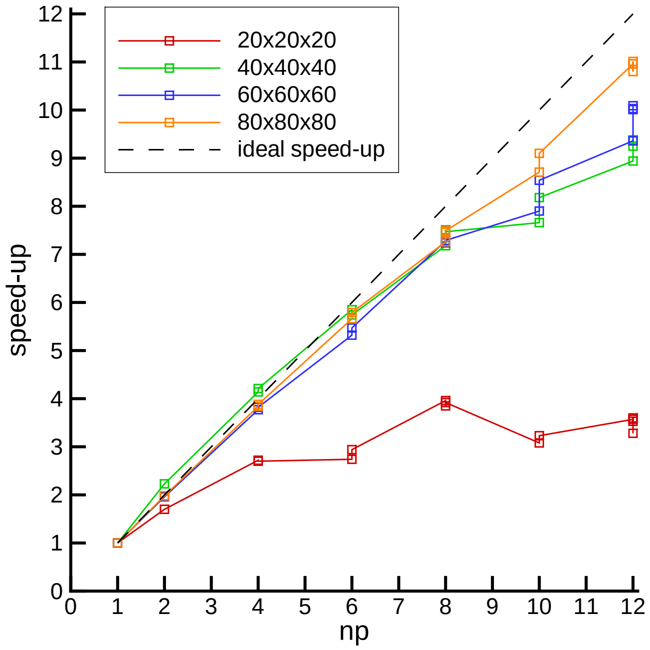

Figure 4.22: Speed-up for the inner product of 3D vectors in the Cray T3E.

np	partition	20 × 20 × 20	40 × 40 × 40	60 × 60 × 60	80 × 80 × 80
1	1x1y1z	1.00	1.00	1.00	1.00
2	1x1y2z	1.70	2.23	1.96	1.98
4	1x1y4z	2.72	4.14	3.77	3.84
4	1x2y2z	2.70	4.21	3.82	3.88
6	1x1y6z	2.74	5.85	5.32	5.66
6	1x2y3z	2.94	5.74	5.47	5.79
8	1x1y8z	3.96	7.18	7.28	7.26
8	1x2y4z	3.85	7.51	7.24	7.46
8	2x2y2z	3.92	7.47	7.29	7.49
10	1x1y10z	3.08	7.66	7.90	8.71
10	1x2y5z	3.23	8.18	8.54	9.10
12	1x2y12z	3.57	8.94	9.36	10.96
12	1x2y6z	3.53	9.38	10.01	10.80
12	1x3y4z	3.60	9.36	10.03	11.01
12	2x2y3z	3.28	9.25	10.09	10.96

Table 4.4: Speed-up of the inner product of 3D vectors in the Cray T3E.

As stated above, the effect of the partitioning configuration has a slight influence on the inner product speed-up.

4.7.4 Matrix-vector product

This operation appears in almost all the solver algorithms. Due to the intensive computational work, it is even used as a work counter in solvers instead of the number of iterations which involve additional operations but with cheaper work. Moreover in a np-parallel machine, the matrix-vector product becomes less effective due to the extra work of communication among the the processor p and the neighbour processors ngb_p. The information of neighbour processors ngb_p is previously "passed" and stored in the mentioned overlapping areas of processor p and then the operation is fully performed in processor p as

$$y_p = A_p x_p + A_{ngb_p} x_{ngb_p}$$

Indeed the size of the overlapping area plays an important role in the time spent in the communication processes. Furthermore, the type of formulation defines the sparsity of the matrix or in other words, the dependencies between the nodes stored in the processor p and those nodes stored in the neighbour processors ngb_p. For a matrix-vector product in a 5,7,9 or 19-point formulation, it is only necessary an overlapping of one ($ov = 1$) in the orthogonal directions of the domain.

Higher order schemes lead to formulations where the overlapping must be higher ($ov \geq 2$), and thus, the amount of data passed among processes increases the time of communication. A generalization of the product of a 7-point formulation matrix with a 3D vector named $mat - vect$ is written for a given overlapping ov in Alg. 35.

Algorithm 35: Mat_vect (A_p, x_p, y_p)

```
***update*** *overlapped information of* $x_p$
  update($ov, x_p$)

***evaluate*** *the matrix-vector product*
***for*** $(i = i1 - ov + 1$ *to* $i = i2 + ov - 1)$
   ***for*** $(j = j1 - ov + 1$ *to* $j = j2 + ov - 1)$
      ***for*** $(k = k1 - ov + 1$ *to* $k = k2 + ov - 1)$
         $y_p(i,j,k) = A_p^p(i,j,k)x_p(i,j,k)+$
                      $A_p^w(i,j,k)x_p(i-1,j,k) + A_p^e(i,j,k)x_p(i+1,j,k)+$
                      $A_p^s(i,j,k)x_p(i,j-1,k) + A_p^n(i,j,k)x_p(i,j+1,k)+$
                      $A_p^b(i,j,k)x_p(i,j,k-1) + A_p^t(i,j,k)x_p(i,j,k+1)$
      ***end for***
   ***end for***
***end for***
```

At this point, the above algebraic operations enable us to build more complicate operations. For example, let us show how the residual needed in the stopping criteria for a 7-point formulation matrix is built in Alg. 36.

Algorithm 36: Residual(A_p, x_p, b_p, r_p)

```
evaluate the operation $r_p = A_p x_p$
    mat_vect ($A_p, x_p, r_p$)
evaluate the operation $r_p = b_p - r_p$
    diff_vect ($b_p, r_p, r_p$)
```

4.7.5 Minimum matrix-vector product size per processor

The following test models the communication and computation timings of the matrix-vector product. It is designed to show the need of an overlapping of the communication and the serial local computation within the operation. The matrix-vector operation for both two dimensional $I \times I$ and three dimensional $I \times I \times I$ CFD problems has been taken (i.e. 5-point and 7-point formulations respectively). The measures of both times, the serial local computation of the matrix-vector product and the exchange of halos of size ov=1, are compared for a wide range of size problems.

The measures of the serial local computation timings are carried out in one processor for different size problems. For instance, the two dimensional problem version is described in Alg. 37.

Algorithm 37: Serial local computation of y=Ax

```
for ($sample = 1$ to $sample = 1000$)
    I[sample]=J[sample]=random(1,1000)
    MPI_Barrier()
    t0=MPI_Wtime()
    for ($j = 1$ to $j = J[sample]$)
        for ($i = 1$ to $i = I[sample]$)
            $y(i,j) = A^p(i,j)x(i,j)$
                $+ A^w(i,j)x(i-1,j)$
                $+ A^e(i,j)x(i+1,j)$
                $+ A^s(i,j)x(i,j-1)$
                $+ A^n(i,j)x(i,j+1)$
        end for
    end for
    t1=MPI_Wtime()
    tcomp[sample]=t1-t0
end for
```

Meanwhile, the exchange of halos is simulated in only two processors for the same range of size problems and on the assumption that the exchange of halos is in all directions. The measure of the times while the exchange of the halos of x for the two dimensional problem is described in Alg. 38.

Algorithm 38: Exchange of halos ov=1 of x

```
for (sample = 1 to sample = 1000)
    I[sample]=random(1,1000)
    MPI_Barrier()
    exchange sides
        t0=MPI_Wtime()
        exchange(I[sample]*ov)
        t1=MPI_Wtime()
        tcomm[sample]=4*(t1-t0)
end for
```

Here, in order to reduce as much as possible the time of communication, the exchange subroutine transfers the packed data from one processor to another in the non-blocking mode.

The test is run in both machines PC cluster and the Cray T3E. The time results versus the size problem are presented in Figs. 4.23, 4.24, 4.25 and 4.26.

If the communication process and the serial local computation process are performed consecutively (i.e. in non-overlapped fashion), the intersection points for each case define the minimum estimate sizes of the local problem which a processor could do with a parallel efficiency of 50%.

Let $tcomp(1)$ and $t_{comp(p)} + t_{comm(p)}$ be the sequential and parallel (for np processors) timings of the overall operation, the efficiency is approximately expressed as

$$E(np) = \frac{t_{comp(1)}}{np(t_{comp(np)} + t_{comm(np)})} \approx \frac{npt_{comp(np)}}{np(t_{comp(np)} + t_{comm(np)})}$$

In the intersection point $t_{comm(np)} = t_{comp(np)}$. Therefore

$$E(np) = \frac{np\ t_{comp(np)}}{np(t_{comp(np)} + t_{comp(np)})} = \frac{1}{2}$$

These points are given in table 4.5:

Problem	PC cluster	Cray T3E
Two dimensional	$40.000 = (200 \times 200)$	$2.500 = (50 \times 50)$
Three dimensional	$64.000 = (40 \times 40 \times 40)$	$3.375 = (15 \times 15 \times 15)$

Table 4.5: Minimum estimate size per processor for a parallel efficiency of 50% for the non-overlapped computation and communication.

Analogously to the previous section, the measure of the speed-up of the matrix-vector product for a given size problem is obtained by timing the algebraic operation at different

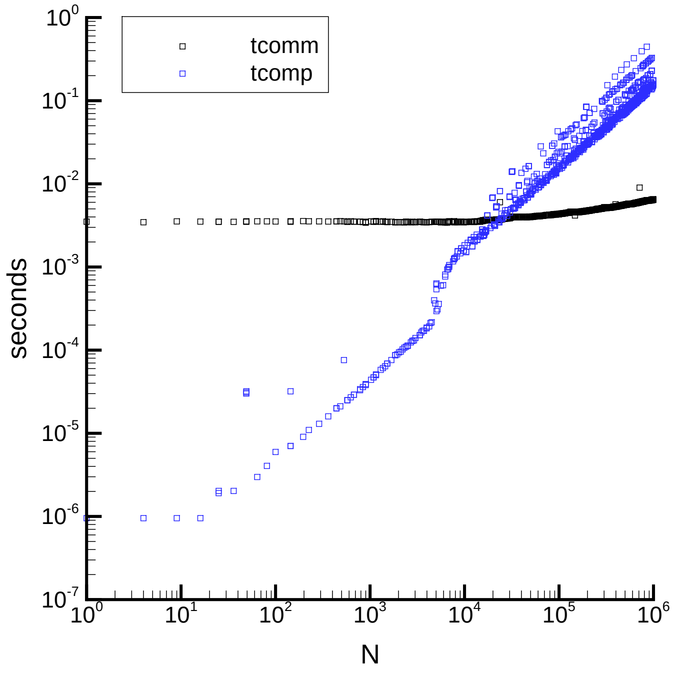

Figure 4.23: Timings of computation and communication of a two-dimensional matrix-vector product in PC cluster.

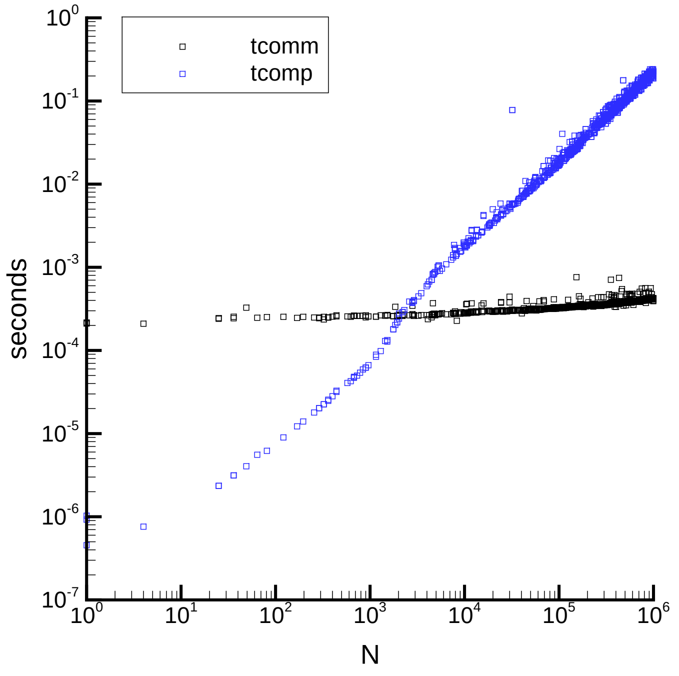

Figure 4.24: Timings of computation and communication of a two-dimensional matrix-vector product in the Cray T3E.

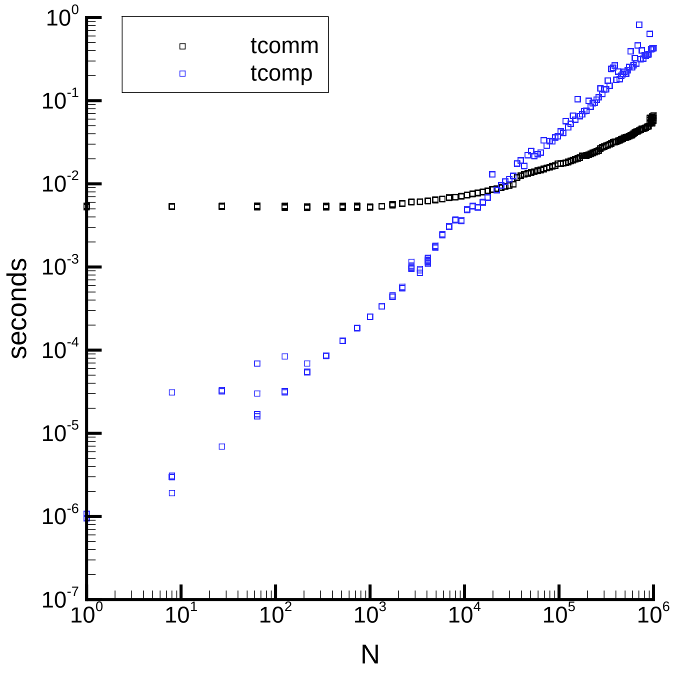

Figure 4.25: Timings of computation and communication of a three-dimensional matrix-vector product in PC cluster.

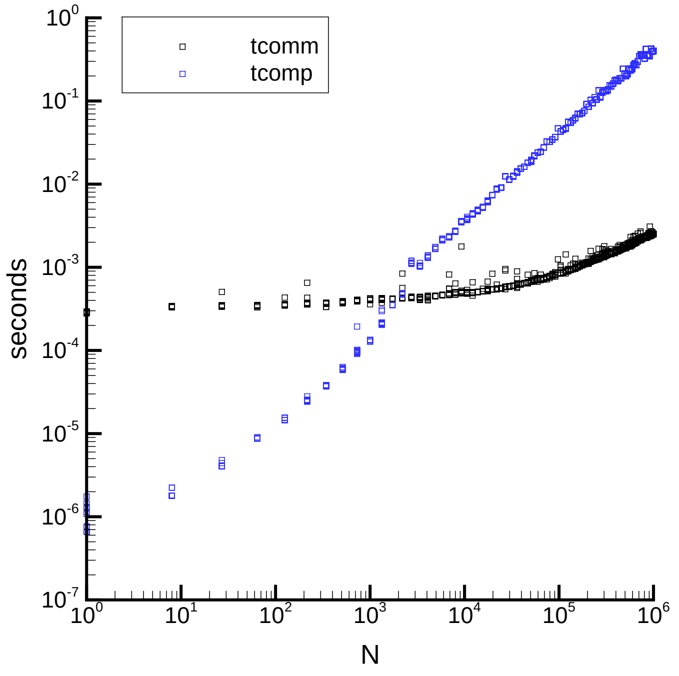

Figure 4.26: Timings of computation and communication of a three-dimensional matrix-vector product in the Cray T3E.

number of processors. The test is executed within the range of 1 to 12 processors in both facilities PC cluster and Cray T3E. Indeed, partitioning in two or three directions affects the efficiency due to (1) the different ratios of computation versus the exchange of data and (2) the cache effects for large amounts of data distributed in few processors.

These effects are analyzed for a set of cases $(20 \times 20 \times 20)$, $(40 \times 40 \times 40)$, $(60 \times 60 \times 60)$, $(80 \times 80 \times 80)$ and $(100 \times 100 \times 100)$ (the last one only for PC cluster).

As mentioned above, the size of all of these cases is over the minimum estimated for the Cray T3E $(15 \times 15 \times 15)$ so one may expect efficiencies higher than 50%. For PC cluster, these efficiencies are expected under the $(40 \times 40 \times 40)$ size.

The experiment is repeated several times for each number of processors and for each partitioning configuration. The speed-up for each case is evaluated with the averaged timings. Full results (all partitioning configurations) are represented in Figs. 4.27, 4.28 and tables 4.6, 4.7 for the PC cluster and the Cray T3E respectively.

The successive experiments have been computed with a generalized algorithm 39 where the number of processors, partitioning configurations, and problem sizes are expressed in a set of nested loops.

Algorithm 39: Performance of operations

```
for $(np = 1 \text{ to } np = 12)$
    for $(partitions = p_x,\ p_y,\ p_z = 1 \text{ to } 12, \text{ such that } p_x p_y p_z = np)$
        for $(size = 20 \text{ to } size = 100, size{=}size{+}20)$
            for $(sample = 1 \text{ to } sample = 20)$
                MPI_Barrier()
                t0=MPI_Wtime()
                evaluate algebraic operation or solve a problem
                t1=MPI_Wtime()
                time=time + t1-t0
            end for
            $t_{comp}[operation][size][partition][np] = \frac{time}{20}$
        end for
    end for
end for
```

Where the algebraic operations are the matrix vector product and inner product, and the solvers are Jacobi, Gauss-Seidel, MSIP, Conjugate Gradient, BiCGSTAB and GMRESR.

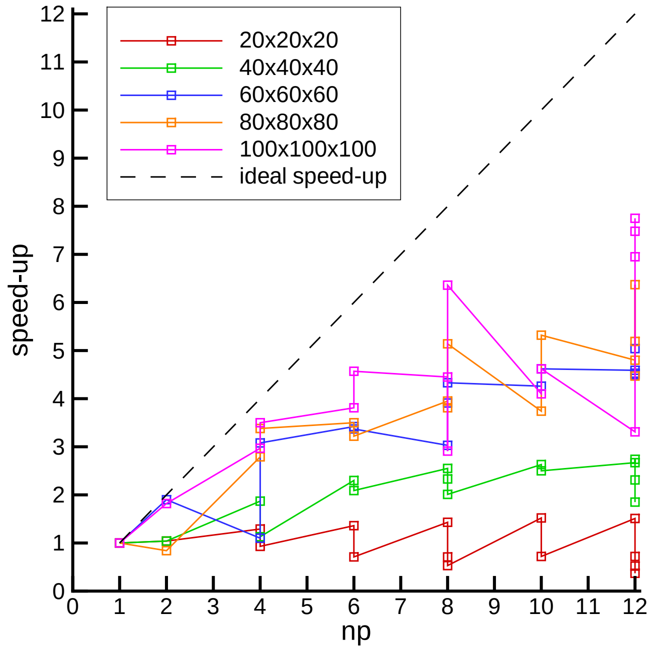

Figure 4.27: Speed-up for the matrix-vector product in PC cluster.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$	$100 \times 100 \times 100$
1	1x1y1z	1.00	1.00	1.00	1.00	1.00
2	1x1y2z	1.04	1.04	1.90	0.84	1.82
4	1x1y4z	1.29	1.87	1.10	2.79	2.97
4	1x2y2z	0.93	1.13	3.08	3.38	3.50
6	1x1y6z	1.36	2.30	3.42	3.50	3.81
6	1x2y3z	0.71	2.09	3.37	3.22	4.57
8	1x1y8z	1.43	2.55	3.03	3.95	4.45
8	1x2y4z	0.71	2.33	3.91	3.81	2.91
8	2x2y2z	0.53	2.01	4.33	5.14	6.36
10	1x1y10z	1.52	2.63	4.26	3.74	4.10
10	1x2y5z	0.72	2.50	4.62	5.32	4.62
12	1x2y12z	1.51	2.67	4.59	4.80	3.31
12	1x2y6z	0.72	2.74	5.04	5.19	6.95
12	1x3y4z	0.52	2.31	4.53	6.37	7.48
12	2x2y3z	0.37	1.85	4.50	4.47	7.75

Table 4.6: Speed-up of matrix-vector product in PC cluster.

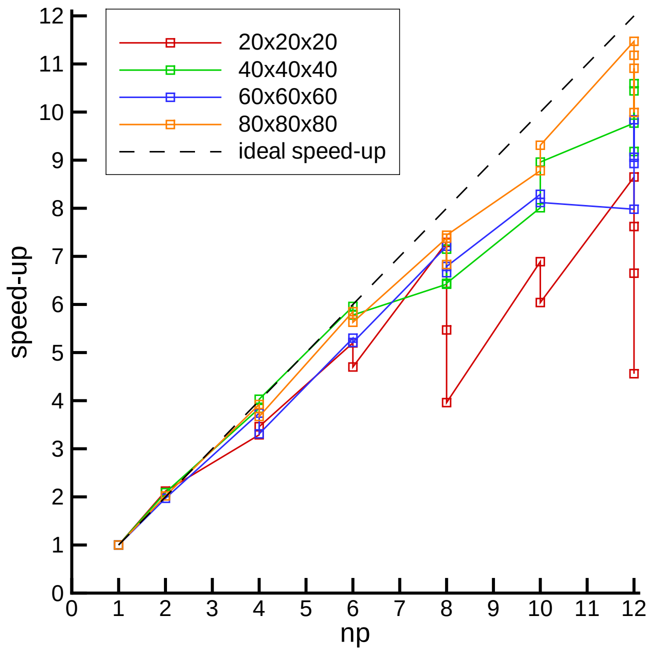

Figure 4.28: Speed-up for the matrix-vector product in the Cray T3E.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$
1	1x1y1z	1.00	1.00	1.00	1.00
2	1x1y2z	2.12	2.09	1.97	2.02
4	1x1y4z	3.29	3.84	3.74	3.92
4	1x2y2z	3.46	4.03	3.31	3.67
6	1x1y6z	5.20	5.96	5.30	5.85
6	1x2y3z	4.70	5.78	5.21	5.63
8	1x1y8z	7.29	6.42	7.21	7.38
8	1x2y4z	5.47	7.15	6.66	6.83
8	2x2y2z	3.96	6.44	6.79	7.44
10	1x1y10z	6.89	8.01	8.29	8.78
10	1x2y5z	6.04	8.96	8.12	9.31
12	1x2y12z	8.65	9.77	7.98	11.47
12	1x2y6z	7.62	10.59	8.93	11.18
12	1x3y4z	6.65	10.44	9.84	9.99
12	2x2y3z	4.56	9.18	9.06	10.91

Table 4.7: Speed-up of matrix by vector in the Cray T3E.

Seeing these figures it is stated that for a given size of the problem and number of processors, the partitioning configuration is definitively a critical factor on the speed-up. The reason is that while the time of computation remains constant independently of the partitioning configuration, the time of communication varies strongly. Then the ratio between the times of computation and communication varies with the partitioning configuration. This ratio in PC cluster is greater than the obtained in the Cray T3E because the time of communication of any packet of data is longer in PC cluster while the time of computation is quite similar in both facilities.

4.8 Parallel performance of solvers

The parallel implementation of these solvers has been tested and the performance, in terms of the speed-up, is measured for a three dimensional Laplace problem (e.g. the 3D heat conduction problem) with full Dirichlet boundary conditions. In this test, the stopping criterion of $\epsilon = 10^{-6}$ has been chosen and analogously to the parallel performance of the algebraic operations, the different partitioning configurations have been tested in different sizes of problems: $20 \times 20 \times 20$, $40 \times 40 \times 40$, $60 \times 60 \times 60$, $80 \times 80 \times 80$, $100 \times 100 \times 100$. For each solver, size of problem number of processors and partitioning configurations, the test is repeated several times and the timings are averaged.

The test is executed at both computers PC cluster and the Cray T3E. The results are summarized by means of the speed-ups. Figures and tables for each solver are reported below.

- For Jacobi solver see Figs. 4.29, 4.30 and tables 4.8,4.9.
- For Gauss-Seidel solver see Figs. 4.31, 4.32 and tables 4.10,4.11.
- For MSIP solver with $\alpha = 0.5$ see Figs. 4.33, 4.34 and tables 4.12,4.13.
- For preconditioned BiCGSTAB solver see Figs. 4.35, 4.36 and tables 4.14,4.15.
- For preconditioned GMRESR(10) solver see Figs. 4.37,4.38 and tables 4.16,4.17.

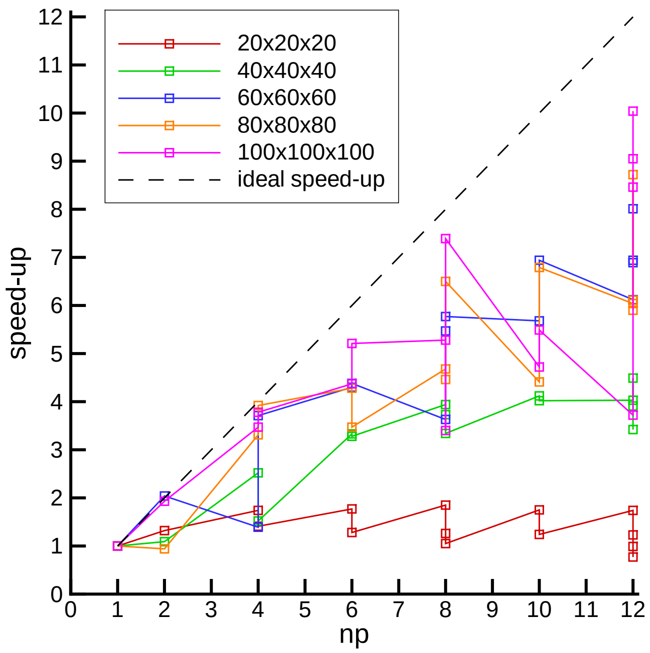

Figure 4.29: Speed-up for the Jacobi in PC cluster.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$	$100 \times 100 \times 100$
1	1x1y1z	1.00	1.00	1.00	1.00	1.00
2	1x1y2z	1.32	1.09	2.04	0.94	1.93
4	1x1y4z	1.74	2.52	1.39	3.31	3.47
4	1x2y2z	1.41	1.52	3.71	3.92	3.78
6	1x1y6z	1.77	3.33	4.30	4.28	4.37
6	1x2y3z	1.28	3.28	4.38	3.47	5.21
8	1x1y8z	1.85	3.94	3.63	4.68	5.28
8	1x2y4z	1.26	3.73	5.47	4.46	3.40
8	2x2y2z	1.05	3.34	5.77	6.50	7.39
10	1x1y10z	1.75	4.12	5.68	4.41	4.72
10	1x2y5z	1.24	4.02	6.94	6.79	5.49
12	1x2y12z	1.74	4.03	6.12	6.04	3.72
12	1x2y6z	1.23	4.49	8.01	6.11	8.46
12	1x3y4z	0.99	3.91	6.94	8.72	9.05
12	2x2y3z	0.77	3.42	6.89	5.90	10.04

Table 4.8: Speed-up of Jacobi solver in PC cluster.

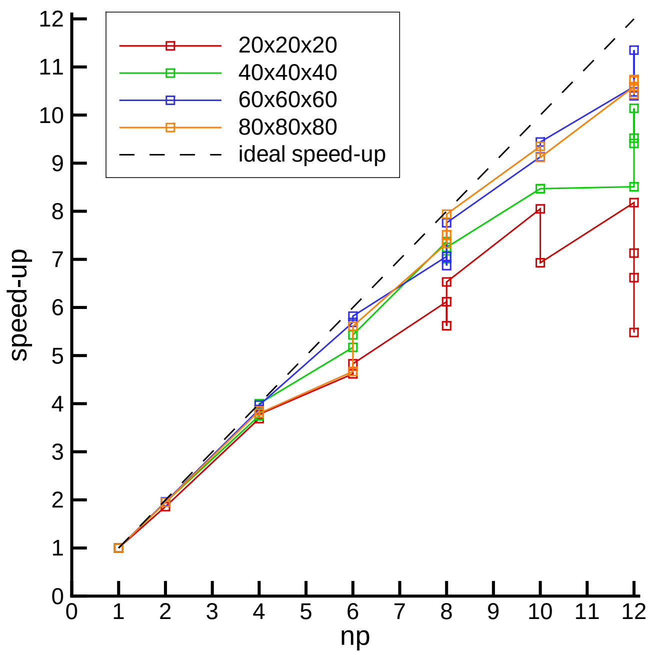

Figure 4.30: Speed-up for the Jacobi in the Cray T3E.

np	partition	20 × 20 × 20	40 × 40 × 40	60 × 60 × 60	80 × 80 × 80
1	1x1y1z	1.00	1.00	1.00	1.00
2	1x1y2z	1.86	1.95	1.96	1.95
4	1x1y4z	3.69	3.75	3.88	3.85
4	1x2y2z	3.78	4.00	3.97	3.80
6	1x1y6z	4.62	5.17	5.69	4.67
6	1x2y3z	4.83	5.43	5.82	5.61
8	1x1y8z	6.12	7.37	7.06	7.33
8	1x2y4z	5.62	7.01	6.87	7.51
8	2x2y2z	6.53	7.25	7.76	7.94
10	1x1y10z	8.05	8.47	9.13	9.35
10	1x2y5z	6.93	8.47	9.44	9.12
12	1x2y12z	8.18	8.51	10.59	10.59
12	1x2y6z	7.13	9.41	10.48	10.43
12	1x3y4z	6.62	10.14	11.35	10.71
12	2x2y3z	5.48	9.52	10.40	10.74

Table 4.9: Speed-up of Jacobi solver at Cray T3E.

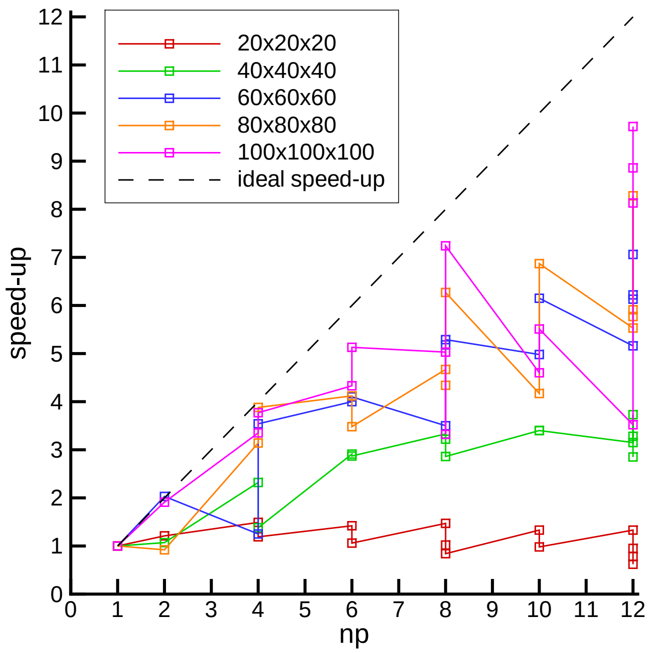

Figure 4.31: Speed-up for the Gauss Seidel in PC cluster.

np	partition	20 × 20 × 20	40 × 40 × 40	60 × 60 × 60	80 × 80 × 80	100 × 100 × 100
1	1x1y1z	1.00	1.00	1.00	1.00	1.00
2	1x1y2z	1.21	1.07	2.03	0.92	1.91
4	1x1y4z	1.49	2.32	1.25	3.14	3.35
4	1x2y2z	1.19	1.38	3.54	3.88	3.77
6	1x1y6z	1.42	2.91	4.00	4.12	4.33
6	1x2y3z	1.06	2.87	4.10	3.48	5.13
8	1x1y8z	1.47	3.33	3.50	4.67	5.03
8	1x2y4z	1.02	3.22	5.18	4.34	3.32
8	2x2y2z	0.84	2.86	5.29	6.27	7.24
10	1x1y10z	1.33	3.40	4.98	4.17	4.60
10	1x2y5z	0.98	3.40	6.15	6.87	5.51
12	1x2y12z	1.33	3.15	5.16	5.53	3.52
12	1x2y6z	0.95	3.73	7.06	5.91	8.13
12	1x3y4z	0.78	3.28	6.22	8.28	8.86
12	2x2y3z	0.62	2.85	6.13	5.77	9.72

Table 4.10: Speed-up of Gauss-Seidel solver in PC cluster.

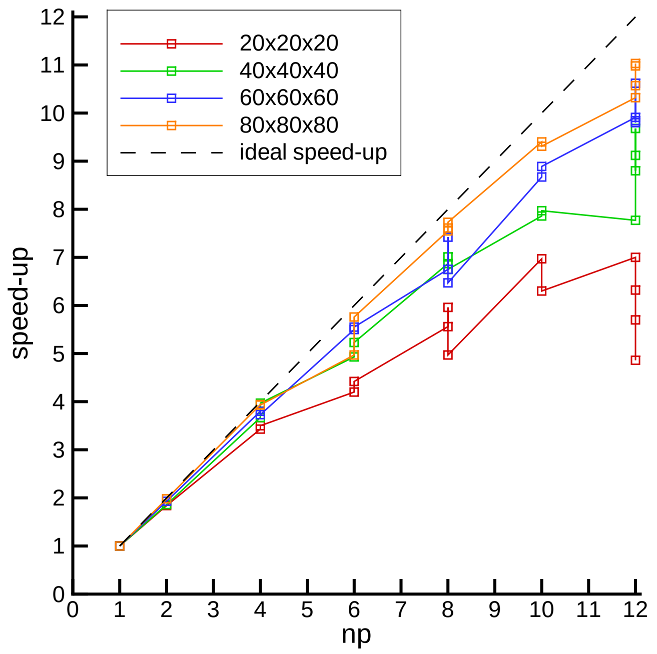

Figure 4.32: Speed-up for the Gauss Seidel in the Cray T3E.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$
1	1x1y1z	1.00	1.00	1.00	1.00
2	1x1y2z	1.84	1.86	1.93	1.98
4	1x1y4z	3.43	3.67	3.80	3.94
4	1x2y2z	3.50	3.97	3.73	3.93
6	1x1y6z	4.20	4.93	5.50	4.97
6	1x2y3z	4.42	5.23	5.55	5.76
8	1x1y8z	5.56	6.87	6.75	7.55
8	1x2y4z	5.96	7.01	7.42	7.61
8	2x2y2z	4.97	6.75	6.47	7.73
10	1x1y10z	6.97	7.86	8.67	9.40
10	1x2y5z	6.30	7.97	8.89	9.31
12	1x2y12z	7.00	7.77	9.91	10.32
12	1x2y6z	6.32	9.12	9.84	10.57
12	1x3y4z	5.70	9.68	10.62	10.98
12	2x2y3z	4.86	8.80	9.80	11.03

Table 4.11: Speed-up of Gauss-Seidel solver at Cray T3E.

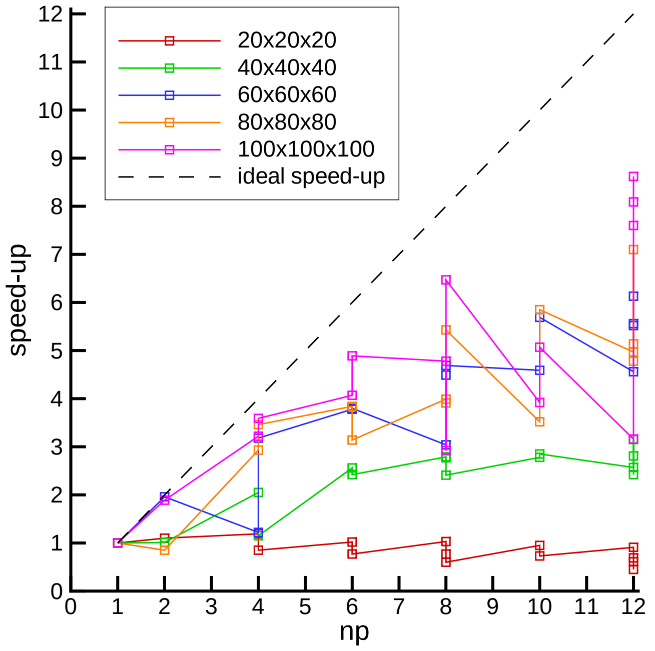

Figure 4.33: Speed-up for the MSIP in PC cluster.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$	$100 \times 100 \times 100$
1	1x1y1z	1.00	1.00	1.00	1.00	1.00
2	1x1y2z	1.10	1.01	1.96	0.85	1.89
4	1x1y4z	1.19	2.05	1.22	2.93	3.22
4	1x2y2z	0.85	1.15	3.18	3.46	3.59
6	1x1y6z	1.02	2.56	3.78	3.84	4.07
6	1x2y3z	0.77	2.42	3.80	3.14	4.89
8	1x1y8z	1.03	2.79	3.04	3.99	4.78
8	1x2y4z	0.77	2.77	4.49	3.91	2.92
8	2x2y2z	0.60	2.41	4.69	5.43	6.47
10	1x1y10z	0.95	2.78	4.59	3.52	3.92
10	1x2y5z	0.73	2.85	5.69	5.85	5.07
12	1x2y12z	0.91	2.57	4.56	4.97	3.16
12	1x2y6z	0.69	3.15	6.13	5.14	7.60
12	1x3y4z	0.61	2.81	5.52	7.10	8.09
12	2x2y3z	0.45	2.42	5.56	4.78	8.62

Table 4.12: Speed-up of MSIP solver with $\alpha = 0.5$ in PC cluster.

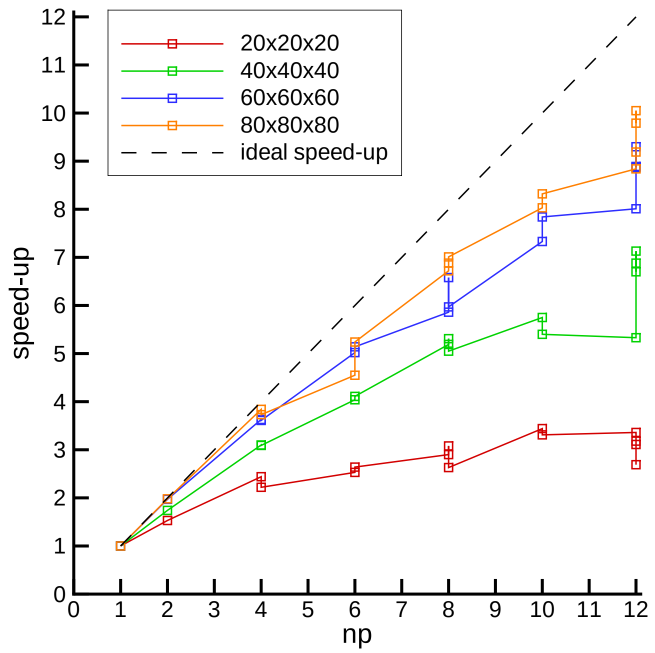

Figure 4.34: Speed-up for the MSIP in the Cray T3E.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$
1	1x1y1z	1.00	1.00	1.00	1.00
2	1x1y2z	1.53	1.74	1.97	1.98
4	1x1y4z	2.44	3.10	3.63	3.84
4	1x2y2z	2.22	3.09	3.61	3.73
6	1x1y6z	2.53	4.04	5.02	4.55
6	1x2y3z	2.64	4.11	5.14	5.24
8	1x1y8z	2.90	5.19	5.86	6.72
8	1x2y4z	3.08	5.31	6.58	6.88
8	2x2y2z	2.63	5.05	5.97	7.01
10	1x1y10z	3.44	5.75	7.33	8.03
10	1x2y5z	3.31	5.40	7.84	8.32
12	1x2y12z	3.36	5.33	8.01	8.84
12	1x2y6z	3.18	6.88	8.89	9.19
12	1x3y4z	3.11	7.13	9.30	10.05
12	2x2y3z	2.69	6.70	8.87	9.79

Table 4.13: Speed-up of MSIP solver with $\alpha = 0.5$ in the Cray T3E.

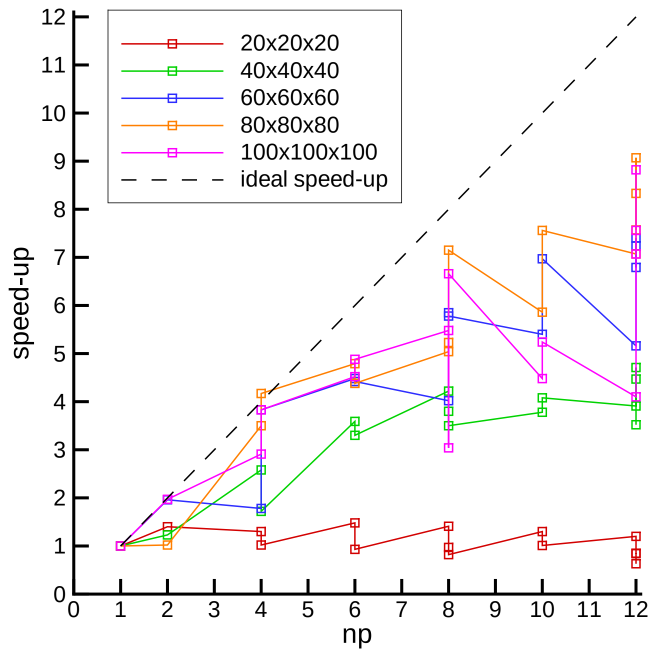

Figure 4.35: Speed-up for the BiCGSTAB preconditioned with MSIP in PC cluster.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$	$100 \times 100 \times 100$
1	1x1y1z	1.00	1.00	1.00	1.00	1.00
2	1x1y2z	1.40	1.23	1.96	1.02	1.97
4	1x1y4z	1.30	2.58	1.78	3.50	2.91
4	1x2y2z	1.02	1.72	3.83	4.17	3.83
6	1x1y6z	1.48	3.59	4.48	4.79	4.52
6	1x2y3z	0.93	3.30	4.42	4.38	4.88
8	1x1y8z	1.41	4.22	4.02	5.04	5.48
8	1x2y4z	0.97	3.80	5.85	5.23	3.04
8	2x2y2z	0.82	3.50	5.78	7.15	6.66
10	1x1y10z	1.30	3.78	5.40	5.86	4.48
10	1x2y5z	1.01	4.08	6.97	7.56	5.24
12	1x2y12z	1.20	3.91	5.16	7.07	4.10
12	1x2y6z	0.84	4.71	7.40	8.33	7.07
12	1x3y4z	0.85	4.47	7.24	9.07	7.56
12	2x2y3z	0.63	3.52	6.79	7.57	8.82

Table 4.14: Speed-up of BiCGSTAB solver preconditioned with MSIP in PC cluster.

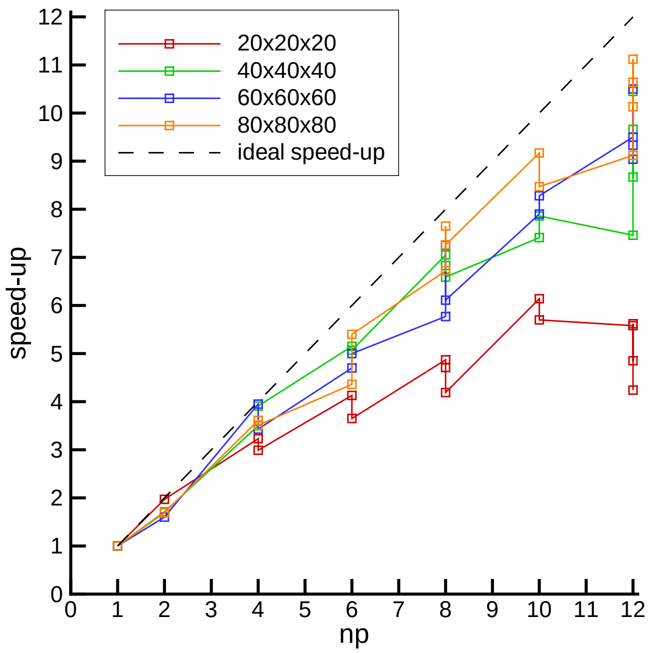

Figure 4.36: Speed-up for the BiCGSTAB preconditioned with MSIP in the Cray T3E.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$
1	1x1y1z	1.00	1.00	1.00	1.00
2	1x1y2z	1.97	1.71	1.60	1.69
4	1x1y4z	3.23	3.50	3.95	3.61
4	1x2y2z	2.99	3.91	3.43	3.51
6	1x1y6z	4.13	5.15	4.70	4.36
6	1x2y3z	3.65	5.07	5.00	5.40
8	1x1y8z	4.87	7.05	5.77	6.72
8	1x2y4z	4.71	6.83	7.25	7.65
8	2x2y2z	4.19	6.59	6.11	7.26
10	1x1y10z	6.14	7.41	7.90	9.17
10	1x2y5z	5.70	7.86	8.28	8.47
12	1x2y12z	5.58	7.46	9.50	9.12
12	1x2y6z	4.85	9.66	9.33	11.12
12	1x3y4z	5.62	10.45	10.50	10.13
12	2x2y3z	4.24	8.67	9.04	10.64

Table 4.15: Speed-up of BiCGSTAB solver preconditioned with MSIP in the Cray T3E.

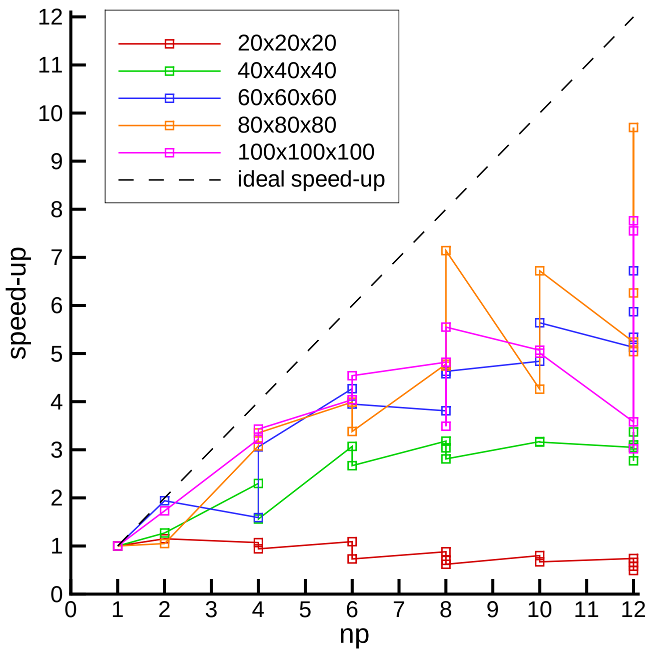

Figure 4.37: Speed-up for the GMRESR preconditioned with MSIP in PC cluster.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$	$100 \times 100 \times 100$
1	1x1y1z	1.00	1.00	1.00	1.00	1.00
2	1x1y2z	1.15	1.27	1.94	1.05	1.73
4	1x1y4z	1.07	2.30	1.59	3.08	3.22
4	1x2y2z	0.94	1.56	3.06	3.35	3.43
6	1x1y6z	1.09	3.07	4.27	3.99	4.04
6	1x2y3z	0.73	2.67	3.95	3.38	4.54
8	1x1y8z	0.88	3.18	3.81	4.78	4.82
8	1x2y4z	0.70	3.04	4.58	4.74	3.49
8	2x2y2z	0.62	2.81	4.63	7.14	5.55
10	1x1y10z	0.80	3.17	4.84	4.26	5.07
10	1x2y5z	0.67	3.16	5.64	6.72	5.01
12	1x2y12z	0.74	3.05	5.13	5.24	3.58
12	1x2y6z	0.66	3.37	6.72	6.26	7.55
12	1x3y4z	0.58	3.10	5.87	9.70	7.76
12	2x2y3z	0.49	2.77	5.34	5.04	3.02

Table 4.16: Speed-up of GMRESR solver preconditioned with MSIP in PC cluster.

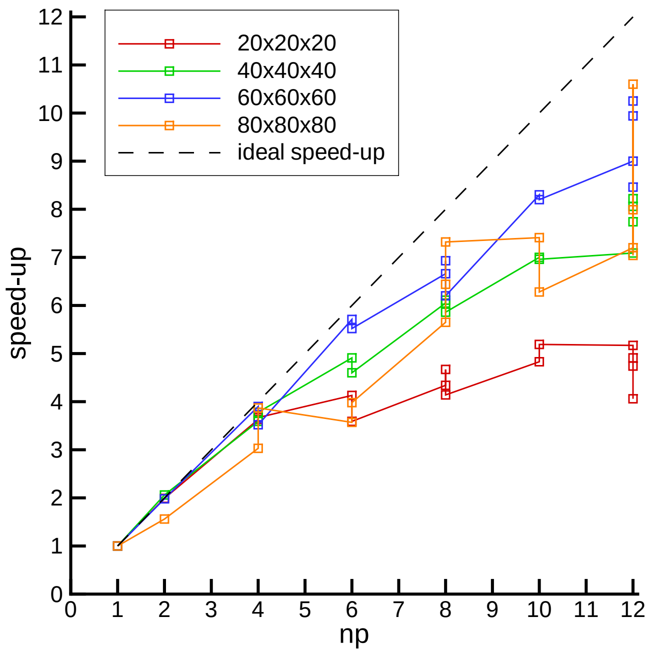

Figure 4.38: Speed-up for the GMRESR preconditioned with MSIP in the Cray T3E.

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$	$80 \times 80 \times 80$
1	1x1y1z	1.00	1.00	1.00	1.00
2	1x1y2z	1.99	2.06	1.98	1.56
4	1x1y4z	3.63	3.59	3.90	3.03
4	1x2y2z	3.67	3.78	3.52	3.87
6	1x1y6z	4.13	4.91	5.71	3.57
6	1x2y3z	3.59	4.60	5.52	3.98
8	1x1y8z	4.34	6.05	6.66	5.65
8	1x2y4z	4.67	6.11	6.93	6.44
8	2x2y2z	4.14	5.86	6.20	7.32
10	1x1y10z	4.83	7.00	8.30	7.41
10	1x2y5z	5.19	6.96	8.20	6.28
12	1x2y12z	5.17	7.09	9.00	7.20
12	1x2y6z	4.91	8.06	10.25	7.99
12	1x3y4z	4.74	8.22	9.94	10.60
12	2x2y3z	4.06	7.74	8.46	7.04

Table 4.17: Speed-up of GMRESR solver preconditioned with MSIP in the Cray T3E.

It is shown in these figures that both facilities PC cluster and the Cray T3E have reported similar behaviours in spite of the differences on their respective communication networks, being the speed ups in the PC cluster as good as those of the Cray T3E.

An unexpected result observed in these figures is that matrix-vector product's speed-ups are worst than Jacobi or Gauss-Seidel's speed-ups for a given problem size. The reason may be due to the better computation with communication ratio in solvers. I.e. the computational load in solvers embeds other operations apart of the matrix-vector product (e.g. the difference of vectors for the computation of the residual and its norm) while the additional communication load, (e.g. communication in the inner product) doesn't increase as the computational load.

Apart from the Jacobi and Gauss-Seidel solvers, there is a similar degradation of MSIP solver and MSIP preconditioner based solvers, i.e. BiCGSTAB and GMRESR. The degradation increases for the one partitioning direction while the multiple partitioning direction reduce this degradation. This seems in contradiction with a bigger number of communication messages for a 3D partitioned domain respect to the 1D partitioned domain. The reason is that the global ILU factorization, e.g. MSIP, is better represented by local ILU factorizations in multiple directions.

This fact is also observed in the increment of the number of iterations when the the number of processors in one direction increases. These numbers have been reported in the table 4.18 for the case of $100 \times 100 \times 100$, i.e. a linear system of 10^6 unknowns.

np	partition	Jacobi	Gauss	MSIP	biCGSTAB+prec	GMRESR(10)+prec
1	1x1y1z	6040	3021	634	35	32
2	1x1y2z	6040	3041	666	34	36
4	1x1y4z	6040	3063	700	41	35
4	1x2y2z	6040	3060	712	35	36
6	1x1y6z	6040	3084	731	33	37
6	1x2y3z	6040	3072	730	38	38
8	1x1y8z	6040	3105	763	34	37
8	1x2y4z	6040	3083	745	41	35
8	2x2y2z	6040	3080	755	38	40
10	1x1y10z	6040	3127	794	41	34
10	1x2y5z	6040	3093	761	41	40
12	1x2y12z	6040	3149	826	35	40
12	1x2y6z	6040	3103	776	41	38
12	1x3y4z	6040	3094	770	41	38
12	2x2y3z	6040	3092	772	39	40

Table 4.18: Iterations for the solution of a linear system with 1000000 unknowns.

It is worth noting that the Jacobi number of iterations remains constant and in the case of the BiCGSTAB it also decreases for some partitioning configurations. In order to represent these numbers, the increment of the number of iterations respect to the sequential computation have been calculated and given in terms of the percentage in figure 4.39.

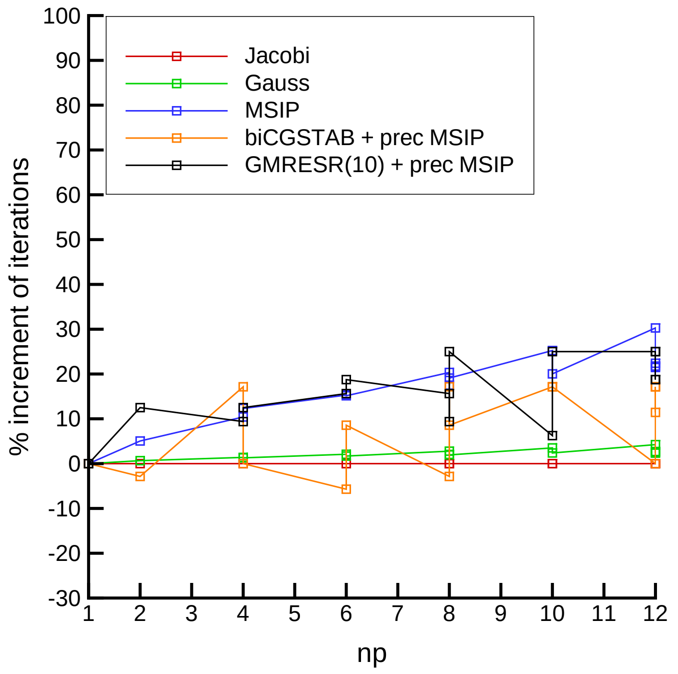

Figure 4.39: Percentage of increment of iterations for the solution of a linear system with 10^6 unknowns when increasing number of processors.

4.9 Nomenclature

A	discretization matrix
a	coeff. in A
b	right hand side
bl	block lenght
d	block of data
E	efficiency
I	unknowns in x-direction
i	index for x-direction
J	unknowns in y-direction
j	similar to i
K	unknowns in z-direction
k	similar to i
N	number of unknowns
n	number of
np	number of processors
ngb	neighbour processors
o	operations of
ov	overlapping area
p	processor identification number
Re	Reynolds number
r	residual
S	speed up
str	stride
t	measure of time
x	unknown
y	auxiliar vector, buffer vector
z	auxiliar vector
$\{x, y, z\}$	cartesian coordinates

Greek symbols

α	network latency
β	network bandwidth
ϵ	precission
ρ	scalar value

Other symbols

$7-PF$	seven point formulation
$<,>$	inner product
$\|\|.\|\|_2$	2-norm of a vector
$\oplus$	general algebraic operation

Superscripts

(k)	k-th iteration

Chapter 5

Parallel CFD in PC clusters

5.1 Implementation details of the SIMPLE-like in parallel

From chapter 2 to chapter 4, we have introduced the different ingredients needed for the computation in parallel of CFD problems by means of the domain decomposition technique in MIMD machines.

- Governing equations, discretization, SIMPLE-like and time marching algorithms in chapter 2.
- Computation of the solution of linear systems of algebraic equations in single processor architectures in chapter 3.
- Parallel computation of the solution of these systems of equations in a PC cluster and the Cray T3E by means of the domain decomposition technique, in chapter 4.

In this chapter, all of these ingredients are reused and applied to an incompressible fluid flow problem with a three dimensional and complex flow structure that requires a fine grid discretization. The solution of a large size problem such as the presented later on, cannot be carried out by a single processor architecture due to high demand of computational resources, i.e., it requires both a high speed of processor and large RAM storage devices.

However, if the original problem is split in sub domains small enough to be fitted and distributed among processors of a MIMD machine and a communication system is provided for sharing data, then each processor can solve independently and in parallel their own part being feasible the solution of the overall problem.

To do so, we re-write the SIMPLE-like algorithm embedded in the time marching method to be used in parallel with the communication steps explicitly remarked. The flowchart and its implementation are described in Fig. 5.1 and Alg. 40 respectively

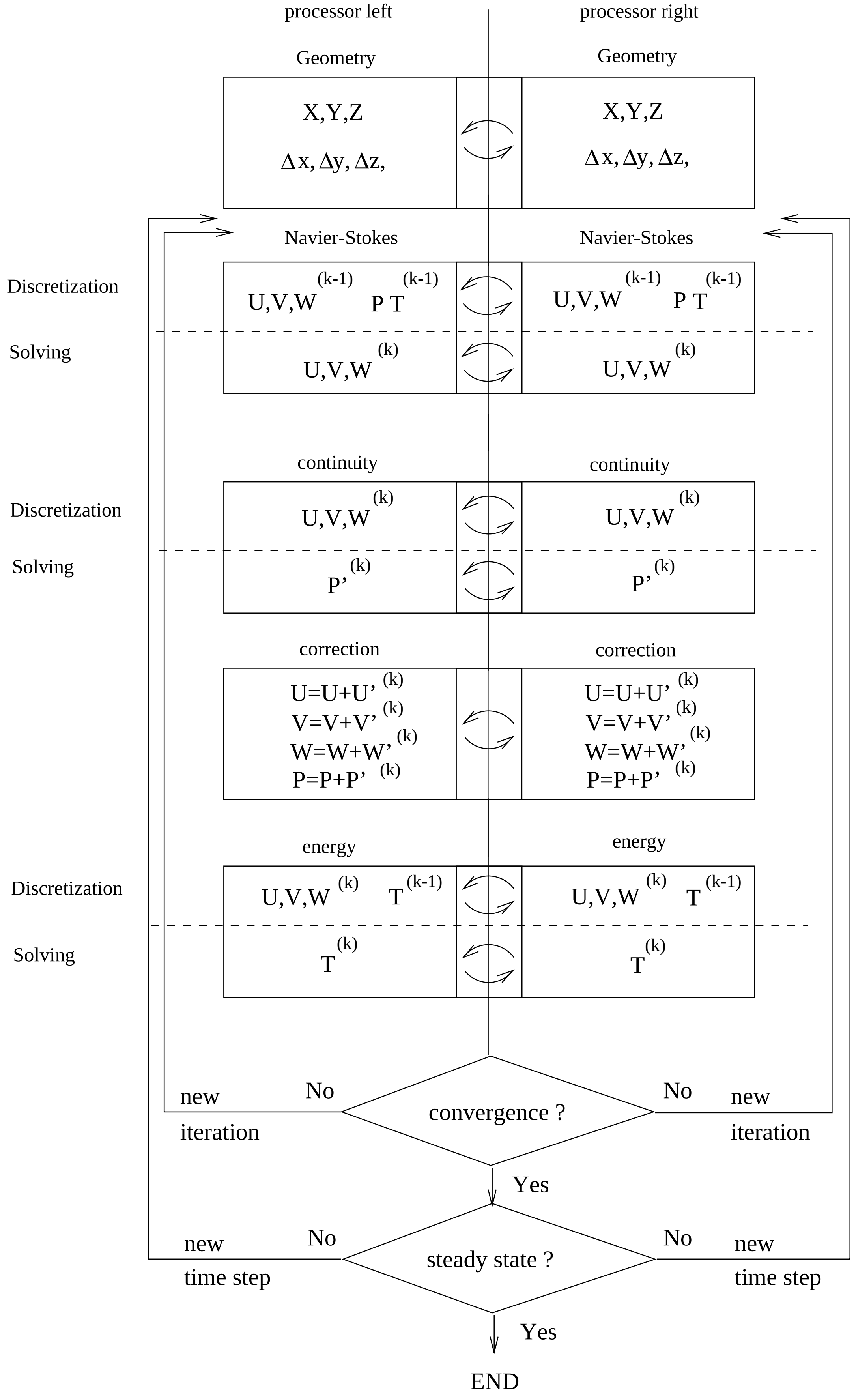

Figure 5.1: parallel SIMPLE plus time marching flowchart.

Algorithm 40: Parallel SIMPLE-like plus time marching

```
***start*** *with* $\tau = 0$
    $\vec{V}^{\tau},\ P^{\tau},\ T^{\tau}$
**do**
    ***new*** *time step* $\tau = \tau + \Delta\tau$
    ***start*** *with* $k = 0$
        $\vec{V}^{(k)} = \vec{V}^{\tau-\Delta\tau},\ P^{(k)} = P^{\tau-\Delta\tau},\ T^{(k)} = T^{\tau-\Delta\tau}$
    **do**
        ***new*** *iteration* $k = k + 1$
        ***guess*** *fluid flow variables*
            $\vec{V}^{*} = \vec{V}^{(k-1)},\ P^{*} = P^{(k-1)},\ T^{*} = T^{(k-1)}$
        ***evaluate*** *coefficients of Navier-Stokes equations*
        ***solve*** $a_{P,u}u_P^{(k)} + \sum_{NGB} a_{NGB,u}u_{NGB}^{(k)} = b_{P,u}$
        ***solve*** $a_{P,v}v_P^{(k)} + \sum_{NGB} a_{NGB,v}v_{NGB}^{(k)} = b_{P,v}$
        ***solve*** $a_{P,w}w_P^{(k)} + \sum_{NGB} a_{NGB,w}w_{NGB}^{(k)} = b_{P,w}$
        ***communication*** *of du, dv, dw*
        ***evaluate*** *coefficients of continuity equation*
        ***solve*** $a_{P,P'}P_P'^{(k)} + \sum_{NGB} a_{NGB,P'}^{(k)}P_{NGB}'^{(k)} = b_{P,P'}$
        ***correct*** *the pressure*
            $P^{(k)} = P^{(k)} + \alpha_P P'^{(k)}$
            ***communication*** *of* $P^{(k)}$
        ***correct*** *the velocity*
            $\vec{V}^{(k)} = \vec{V}^{(k)} + \alpha_V \vec{V}'^{(k)}$
        ***evaluate*** *coefficients of energy equation*
        ***solve*** $a_{P,T}^{(k)}T_P^{(k)} + \sum_{NGB} a_{NGB,T}^{(k)}T_{NGB}^{(k)} = b_{P,T}^{(k)}$
        ***communication*** *of* $\vec{V}^{(k)},\ P^{(k)},\ T^{(k)}$
    **until** (*mass, momentum and energy conservation*)
    $\vec{V}^{\tau} = \vec{V}^{(k)},\ P^{\tau} = P^{(k)},\ T^{\tau} = T^{(k)}$
**until** (*steady state of all variables*)
```

By comparison with the sequential one, it is worth noting that only few new steps have been introduced while the remaining steps, and in general, the algorithm has been kept unchanged. From the implementation point of view of this algorithm, it is better to start with a description of the grid generation and the grid point indexation of centered and staggered grids for each subdomain.

For simplicity, a two dimensional rectangular domain is considered and divided by two in x direction obtaining two parts the left and the right (see Fig. 5.2). The division or

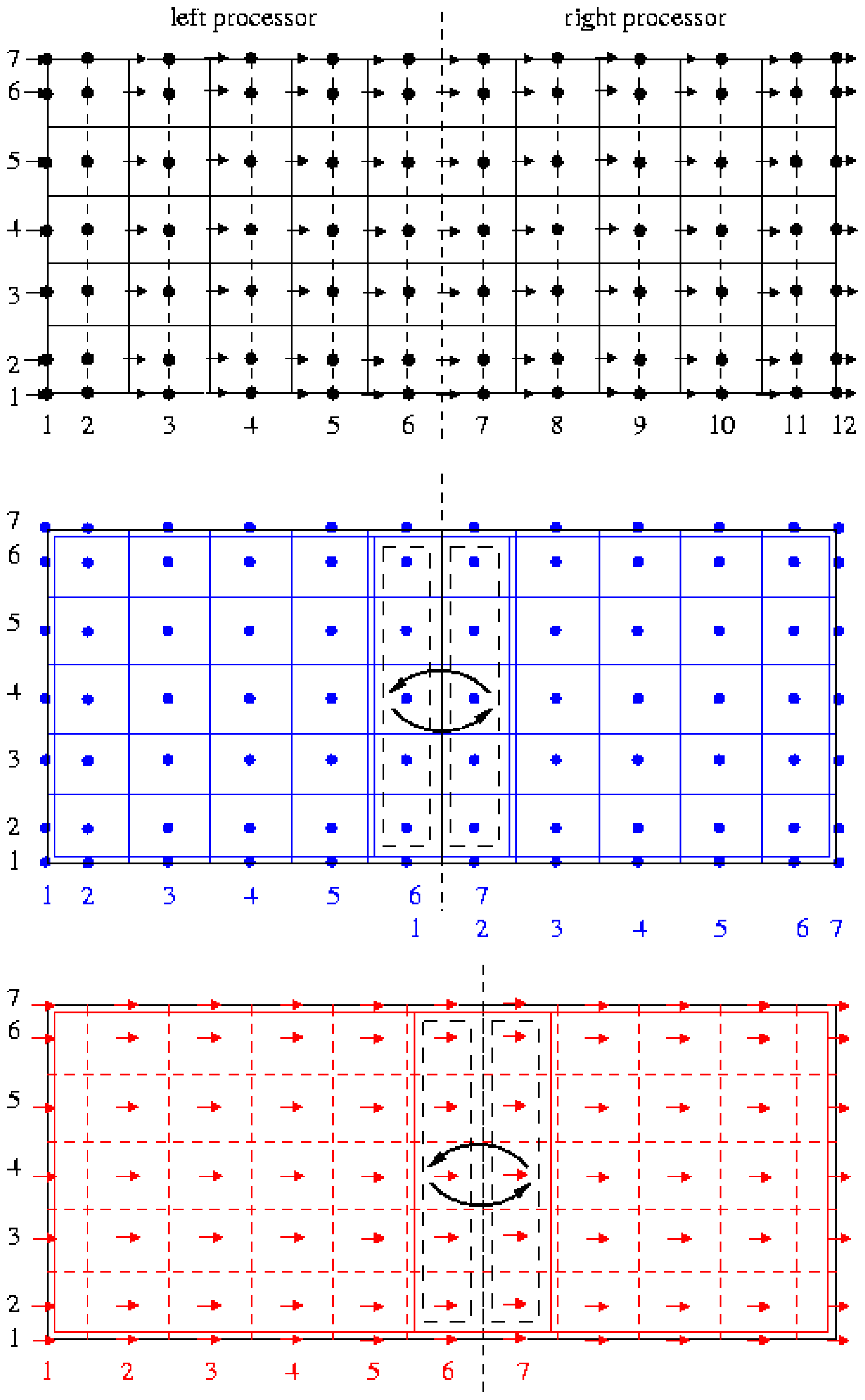

Figure 5.2: Top, centered (solid line) and staggered (dashed line) grids in x direction for the overall domain. Center, centered grid in blue split in x direction. Bottom, staggered grid in red in x direction.

partition is done on the centered grid (see grid painted in black for the original and in blue for the partitioned) as balanced as possible. Once we have the centered grids associated to each processor, the staggered grids can be build independently at each processor (shown here in red for the x direction).

It is shown in the figure the index of each grid point for both partitions left and right. The zones enclosed with dashed lines contain the grid points that are involved in the communication processes between both processors. For the centered grids, these points stand for the pressure and pressure correction variables while for the staggered grid ones in x direction the zone enclose the velocity variable in x direction that will be involved in the communication processes.

These zones define the boundaries in contact between the processors for each subdomain. If the boundaries of a subdomain do not keep in contact with other subdomains, then we have to fix them with the boundary conditions explained in chapter 2, i.e., the Dirichlet, Neumann or periodic boundary conditions in order to close the problem for each subdomain.

Following with the explanation and related with the communication issues there are two points inside the algorithm where the communications are required: the communication of the set of variables before the evaluation of the coefficients of each system of the SIMPLE algorithm and the communication of a single variable inside the solver when needed. The first type of communications, say update of variables, are needed before the evaluation of coefficients near the boundaries in contact with neighbour processors. For instance, in order to evaluate the convective term of the Navier-Stokes equations at such boundaries, it may be necessary the velocity placed at the neighbour processor. The amount of velocity points from the neighbour processor depends on the scheme used. For our purpose, i.e., the Upwind and the Central Difference Schemes only one line of points is required. For higher order schemes, two or even more lines of points are needed. If that is extrapolated to the three dimensional case, a thin slice of one point is needed for Upwind and Central Difference Schemes.

The Navier-Stokes equations require also the gradient of the pressure variable so it is also necessary to obtain those points placed at the neighbour processors. It is worth noting that each variable has its own indexation system per processor and the communication of the variables of these systems must be considered carefully in order to place at the right location these variables on the receiver processor. Conversely it is also important to give the right location of the variables to be sent to the neighbour processors.

Fig. 5.3 show the communications needed from the processor right to the processor left before the evaluation of the Navier Stokes coefficients in x direction. The velocities U, V (figured with straight arrows) and the pressure P (figured with filled dots) are sent (figured with curved arrows) from the enclosed zones of the right processor to the left processor.

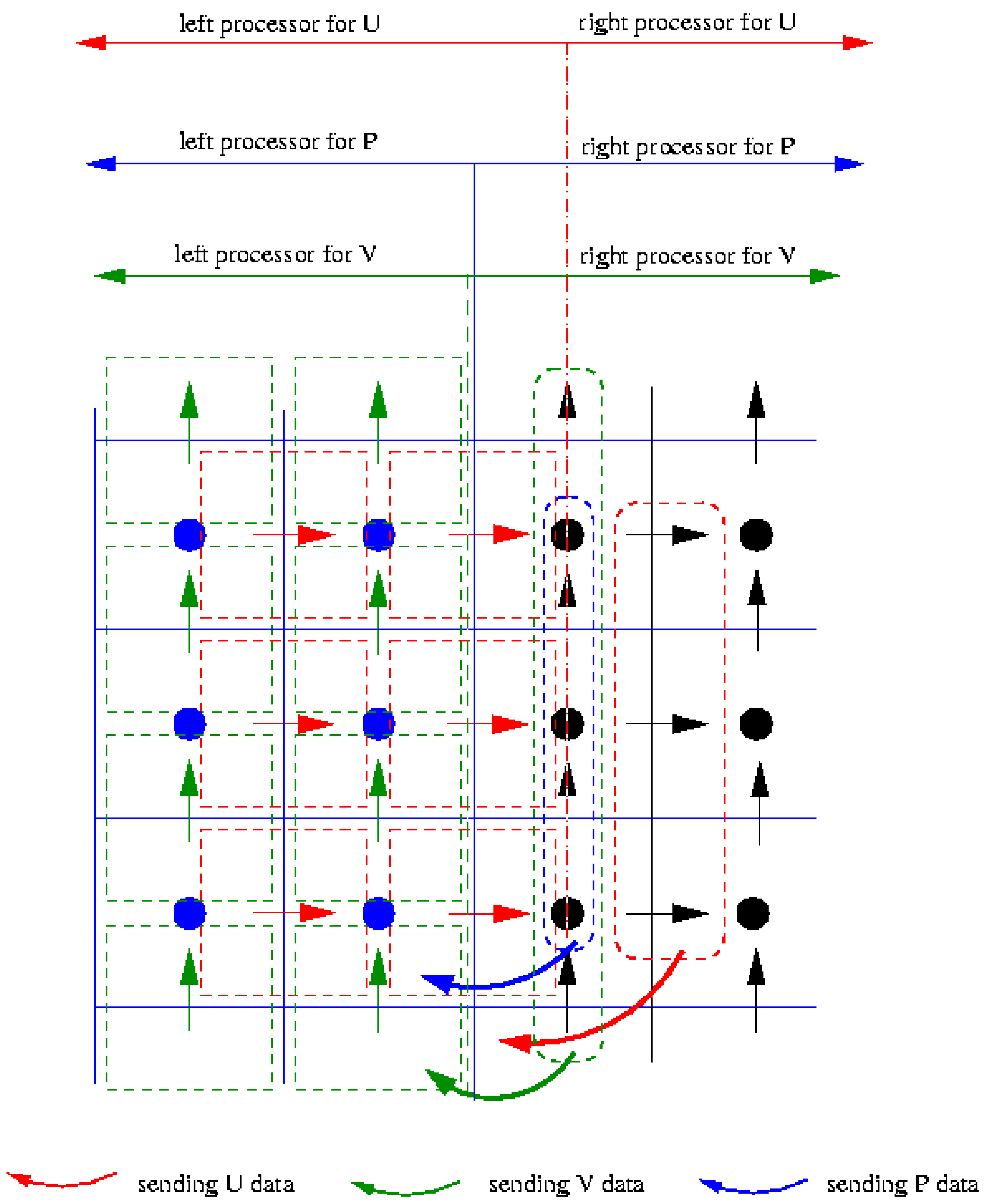

Figure 5.3: Communications from the right processor to the left processor before the evaluation of the coefficients of the Navier Stokes equation in x direction.

Once, all of these communications are done, i.e., the U, V, W and P the algorithm proceeds to the evaluation of the coefficients of Navier-Stokes equations for each direction.

Another point within the algorithm that requires such communications is placed before the evaluation of the coefficients of the continuity equation. In this case, the evaluation of these coefficients require the values of du, dv and dw that have been evaluated in the Navier-Stokes equations. Since these values are placed on staggered grids and the continuity equation is placed on the centered grid, we have take carefully the indexation of each original and destination location for each value.

Finally, the correction step and for instance, the correction of the velocities requires the gradients of the correction pressure so it is necessary to receive the correction pressures that are placed in neighbour processors. Again, the index of the centered grid associated to the pressure correction is different to the index of the staggered grids. Therefore, we have to consider the locations from the centered grid of the neighbour processors to the staggered grids of the receiver processors.

Once the systems of equations are defined the solver can compute the solution for each subdomain in parallel. Since these systems, as explained in chapter 3, are featured by non symmetric coefficient matrices, we have adopted the incomplete factorization solver MSIP for the three systems of Navier-Stokes equations and the Krylov space based method BiCGSTAB preconditioned with MSIP for the computation of the solution of the remaining equations, i.e. the continuity equation and the energy equation.

5.2 Benchmarking the parallel implementation

The lid driven cavity flow is one of the well known benchmark test for a CFD code. It has been chosen among other benchmark tests due to its geometric simplicity but complex flow structure. As was shown by Koseff and Street (60), the flow structure in a 3D cavity becomes more complex than a 2D cavity. Their experiments demonstrated that the presence of side-walls produced a three-dimensional structure that significantly altered the primary flow in the central plane, behaviour that 2D simulations simply could not capture. In Fig. 5.4 the experimental setup and a sketch of the flow structure are shown. The flow structure is composed by a primary eddy, a downstream secondary eddy, an upstream tertiary eddy and a corner eddy.

For laminar flows ($Re < 6000$) they found two sources of 3D structure. Examining a plane parallel to the downstream wall, corner eddies were caused at the juncture of the side-walls and the ground. Downstream secondary vortices caused as well by centrifugal forces along the downstream eddy separation surface were found along the span. These came to be known as Taylor-Gortler-Like vortices (61) in reference to their curvature-induced origins. The number and location of these vortices were function of Reynolds number.

Simulations were performed for cavities of SAR (Span Aspect Ratio) 1:1 and 3:1 for a wide range of Reynolds numbers: $10^3 - 10^4$. Koseff and Street reported on one hand the nondimensionalized height of the downstream secondary eddy with the Reynolds number and on the other hand the central and end-wall plane velocity profiles for a Reynolds number of 3200.

These experimental results are compared with a 3D simulation based on the SIMPLE and time marching algorithms onto a $120 \times 120 \times 120$ grid of points. In order to reduce the computational effort, the domain has been divided among 8 processors with a partitioning configuration of $2x \times 2y \times 2z$ (see Fig. 5.4), such that each processor contains $60 \times 60 \times 60$ grid points.

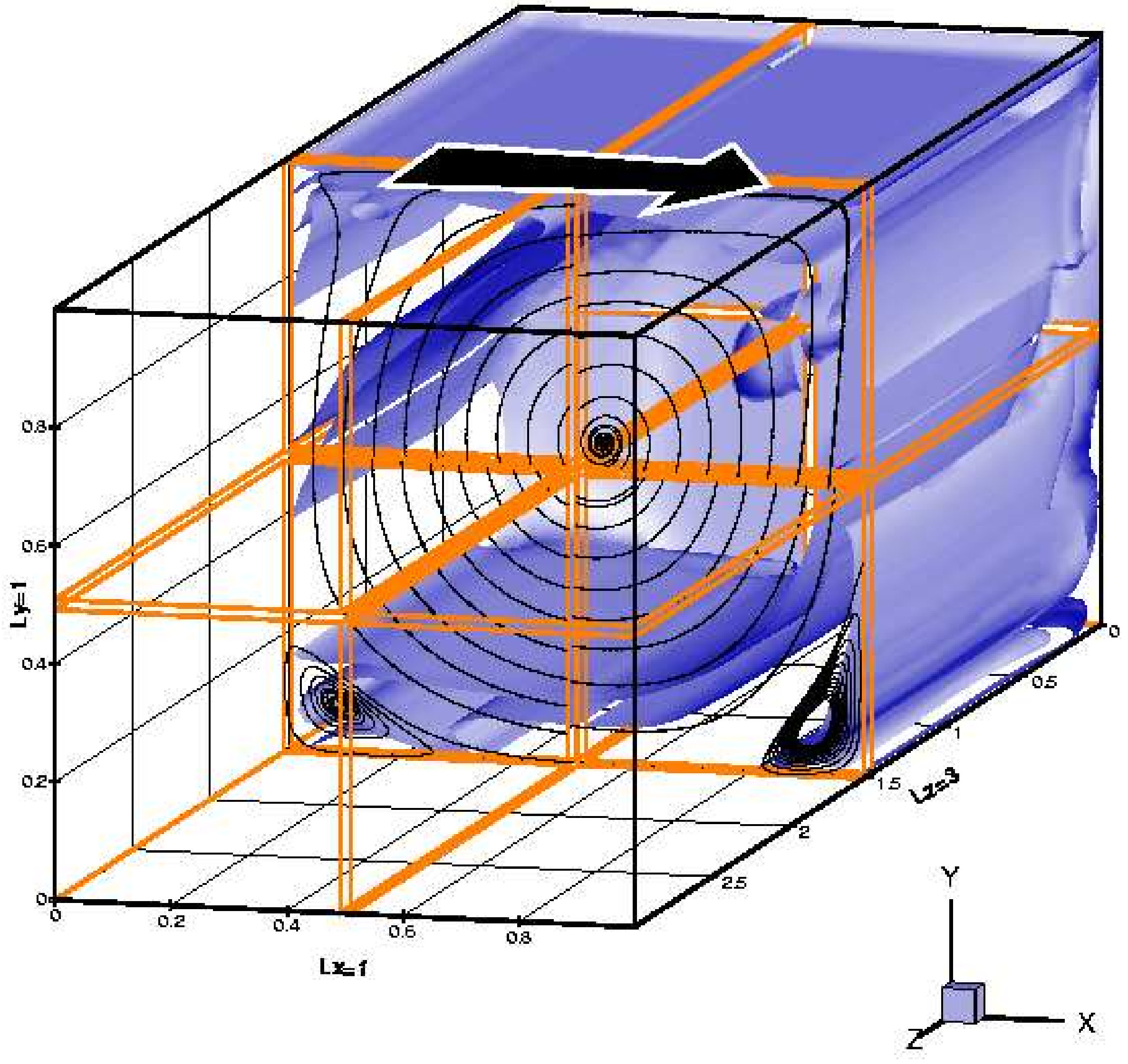

Figure 5.4: Domain partitioned in three directions $2x \times 2y \times 2z$ in order to be simulated with 8 processors

The first results provided by the simulation refer to the Reynolds number 3200. The fluid-flow variables are depicted in figure 5.5 with stream lines at few locations inside the domain and they are conducted to show the 3D structure of the flow. The three orthogonal central planes X, Y and Z are also shown for clarity of the structure of flow. The velocity vectors have been scaled for a better visualization.

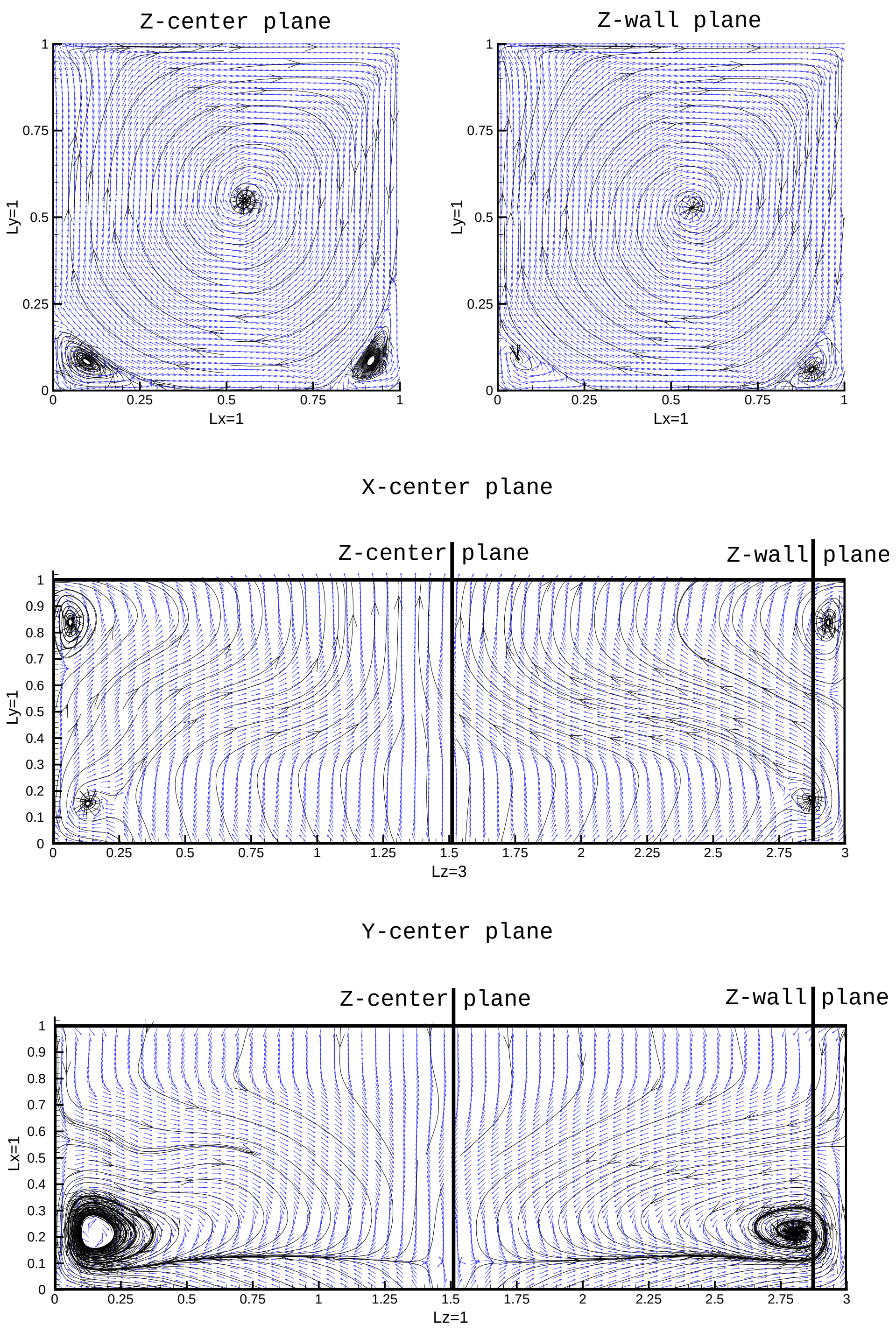

Figure 5.5: Left-top, Z central plane. Right-top Z end-wall plane. Center, Y central plane. Bottom X central plane.

The U and V velocity profiles for the Z central plane have been compared with the experimental data of Koseff and Street (see Fig. 5.6). The figure shows the good agreement for two simulations with grids $60 \times 60 \times 60$ and $120 \times 120 \times 120$ executed in parallel with 8 processors at PC cluster.

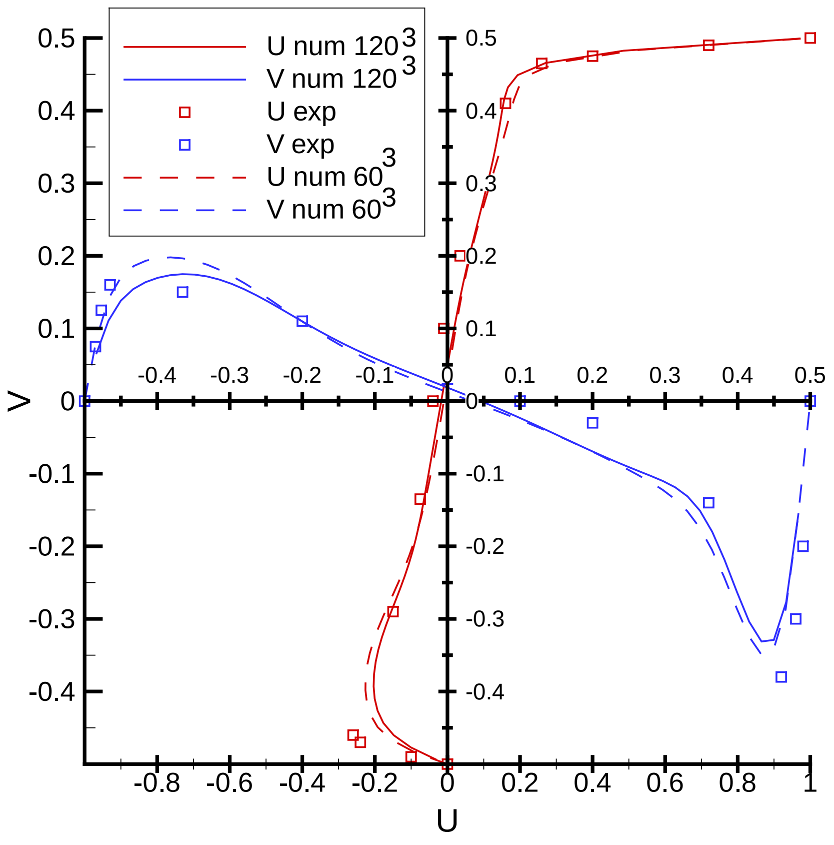

Figure 5.6: Numerical and experimental velocity profiles for Reynolds number of 3200 at Z central plane.

It is worth noting the main differences produced at the vivicinity of walls that could be reduced by refining the grid, concentrating grid points onward walls and using high order schemes.

Moreover, numerical results provided by DPC code developed at CTTC (see DPC and CTTC in acronyms page) have been added to the validation of the current code. The DPC code is also based on the finite volume method with SIMPLE-like coupling algorithm, and hence, the numerical results at same grid should be identical.

For DPC code, a maximum of $60 \times 60 \times 60$ regular grid points with upwind scheme may be fitted in a single processor. Concentration grid factors at the vicinity of the walls as well as higher order schemes such those based on the deferred correction, produce several instabilities for this Reynolds number.

The agreement of the numerical results with the experimental data give small discrepancies at maximun values of center lines (see table 5.1).

data source	experimental	DPC	present code	present code
grid	–	$60 \times 60 \times 60$	$60 \times 60 \times 60$	$120 \times 120 \times 120$
V_{max}	0.19	0.18	0.20	0.25
x location V_{max}	-0.41	-0.31	-0.39	-0.35
V_{min}	-0.38	-0.32	-0.35	-0.45
x location V_{min}	0.46	0.45	0.45	0.45
U_{min}	-0.28	-0.20	-0.23	-0.29
y location U_{min}	-0.40	-0.33	-0.39	-0.40

Table 5.1: Discrepancies of results.

The second results presented refer to the height of the downstream secondary eddy when varying the Reynolds number from 1000 to 8000 (see Fig. 5.7).

The differences respect to the experimental data may be due to different factors. From the numerical point of view, we do not know if the grid mesh is fine enough to simulate accurately the fluid flow phenomena. Furthermore, since the upwind scheme has been used for the treatment of the convective terms, it introduces a numerical diffusion of the velocity values and hence it affect to the global result. A more detailed simulation by means of a finer grid would reveal the tendency of the numerical solution. However, the storage capacity of computers is limited even in distributed machines so this study has been stopped at this stage. From the experimental point of view, Koseff et al. reported a measured error of $1-10\%$ at all ranges of velocities. Therefore, the agreement between numerical and experimental results has been considered satisfactory.

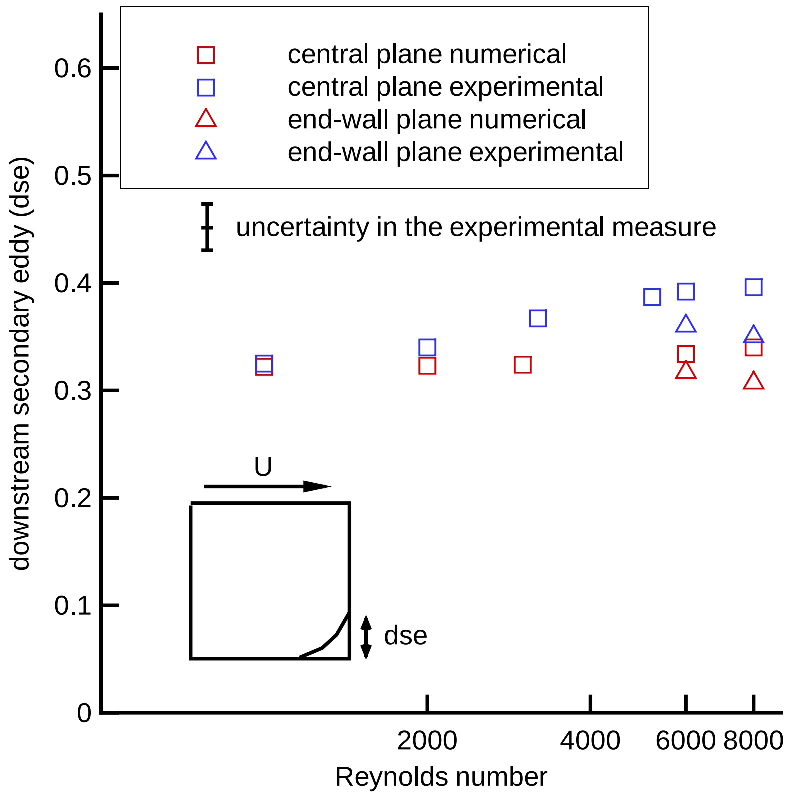

Figure 5.7: Height of the downstream secondary eddy for different Reynolds numbers

Finally, if the simulation is carried out for Reynolds numbers larger than 6000, the structure of the flow becomes unstable (60) being more difficult the convergence. At those Reynolds, the Taylor-Gortler-like vortices and the corner vortices are clearly visible. The longitudinal Taylor-Gortler-like vortices are formed because the surface of separation between the primary eddy and the downstream secondary eddy is effectively a concave wall. The curved separation surface promotes the instability caused by the centrifugal forces on this wall, leading to the formation of Taylor-Gortler-like structures. The corner vortex originates from the adjustment of the shear and pressure forces acting on the recirculating fluid to the non-slip condition imposed by the presence of the end-wall.

These structures are the most evident manifestation of three-dimensionally in this flow and they have been simulated at Reynolds number 8000 onto a 120x120x120 grid of points with 8 processors. The streamlines of the flow near the corner composed by the end-wall, ground and downstream-wall are represented in Fig. 5.8.

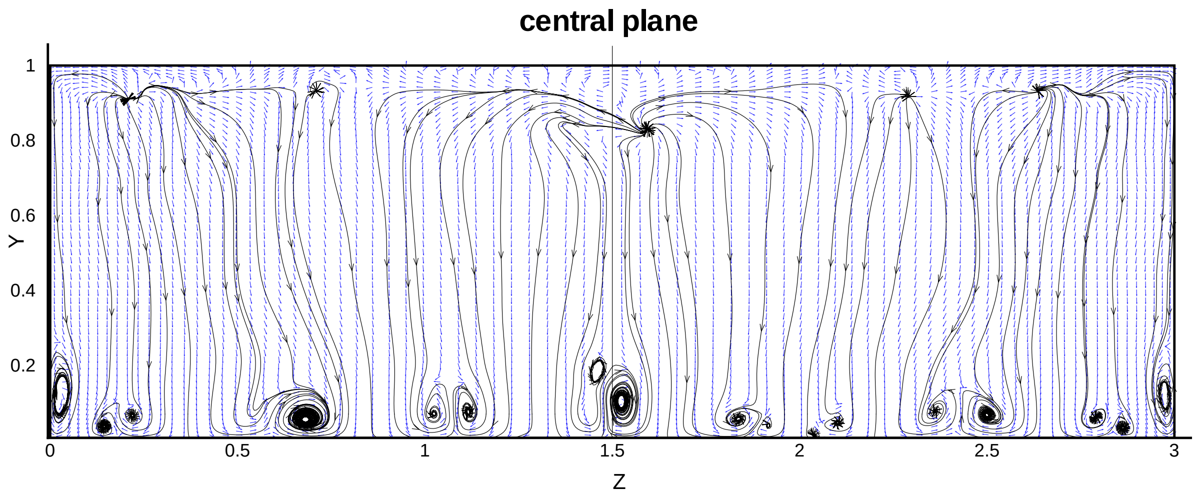

Figure 5.8: Pairs of Taylor-Gortler-like vortex along the span wise direction for a Reynolds number of 8000. There are also shown the corner vortex at both end walls.

5.3 Performance measure of the parallel implementation

In previous chapter, we have done an analysis of the parallel performance of several solvers once given a model problem and submitted to different number of processors, partitioning configurations and varying the size of the problem. Following this scheme, the 3D lid driven cavity for a Reynolds number of 1000 in a $1 \times 1 \times 1$ cubic box has been chosen for the parallel performance of the SIMPLE and time-marching algorithm in parallel. The measured wall clock time catches the time elapsed from the initial state up to the steady state. Several grid sizes $20 \times 20 \times 20, 40 \times 40 \times 40, 60 \times 60 \times 60, 80 \times 80 \times 80 i 100 \times 100 \times 100$ have been used in order to see the effect of the grid size per processor and to give an idea of the scalability. The measures of the speed-up have been done within a range of 1-12 processors.

Analogously to the results presented in the previous chapter, the speed-ups are given in tables 5.2, 5.3 and represented in Figs. 5.9, 5.10 for both machines, i.e., PC cluster and the Cray T3E. It is worth noting that in the case of the Cray T3E, the maximum grid size that can be hold in RAM by a single processor is limited to a $60 \times 60 \times 60$.

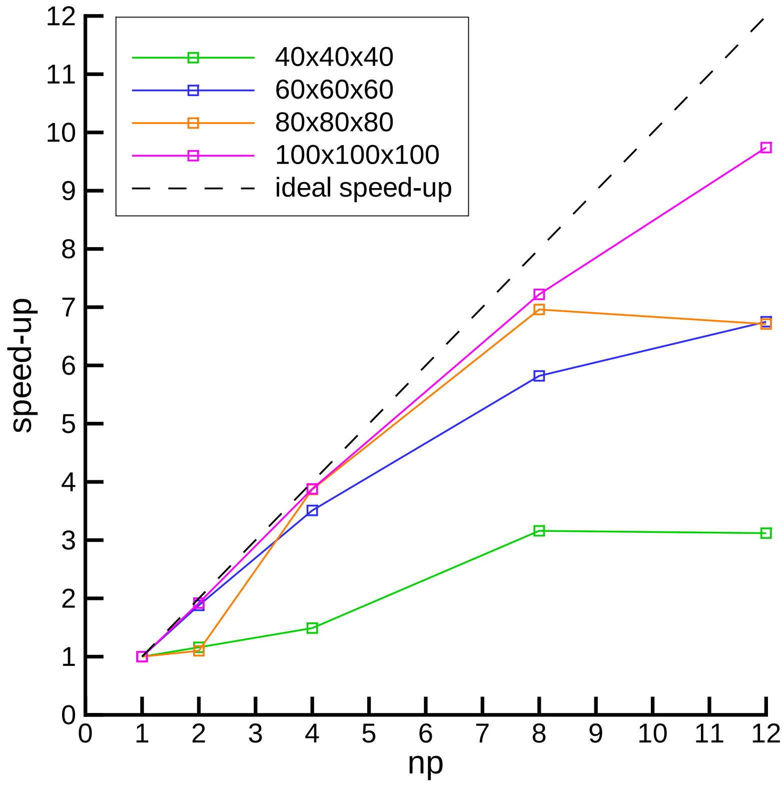

Figure 5.9: Speed-up in PC cluster.

np	partition	40 × 40 × 40	60 × 60 × 60	80 × 80 × 80	100 × 100 × 100
1	1x1y1z	1.00	1.00	1.00	1.00
2	2x1y1z	1.16	1.88	1.10	1.92
4	2x2y1z	1.49	3.51	3.87	3.88
8	2x2y2z	3.16	5.82	6.96	7.22
12	3x3y2z	3.12	6.75	6.71	9.74

Table 5.2: Speed-up in PC cluster.

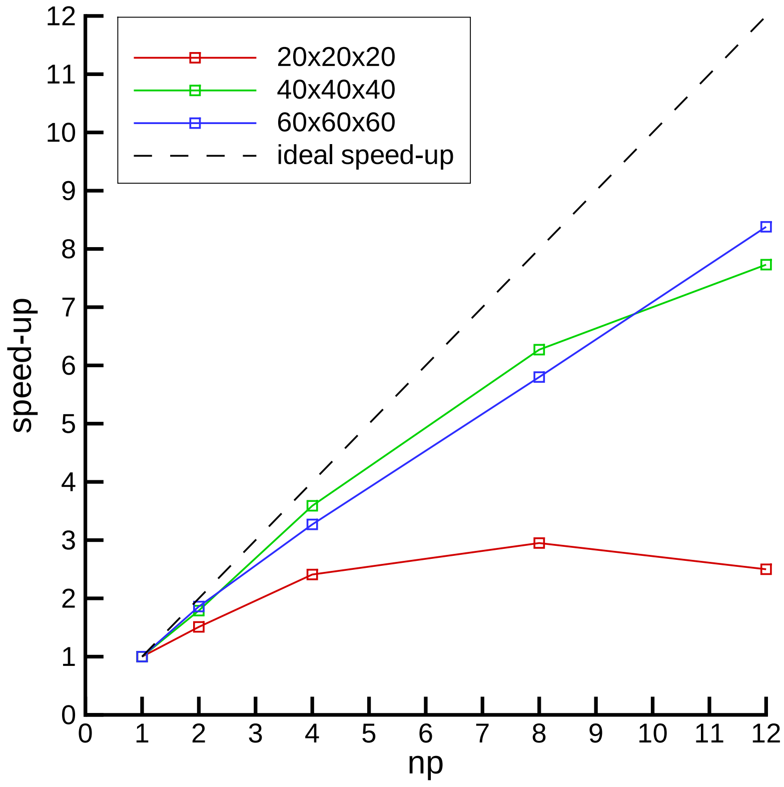

Figure 5.10: Speed-up in the Cray T3E

np	partition	$20 \times 20 \times 20$	$40 \times 40 \times 40$	$60 \times 60 \times 60$
1	1x1y1z	1.00	1.00	1.00
2	2x1y1z	1.51	1.79	1.86
4	2x2y1z	2.41	3.59	3.27
8	2x2y2z	2.95	6.27	5.80
12	3x3y2z	2.50	7.73	8.38

Table 5.3: Speed-up in the Cray T3E.

From the results presented above, we state that the performance at both machines is similar and the configurations of 8 and 12 processors have reported good speed-ups of 7 and 10 respectively when the grid size of the problem increase. This behaviour is due to the two factors mentioned in previous chapter. The minimum grid size per processor constrains the efficiency of the parallel computation to large grid sizes. Conversely, given a problem with a fixed grid size, there is a loose of efficiency when increasing the number of processors in one direction. This fact is due to the degradation of the solver and more precisely to the degradation of the incomplete factorizations used within the solver or within the preconditioner.

This loose of efficiency is much more significative within the solution of the continuity equation than the solution of the Navier-Stokes equations. In order to show this, an analysis of the timings of each step of the SIMPLE-like algorithm are provided. The computation and the communication times have been expressed in terms of the percentages respect to the overall wall clock time of the simulation.

The two bar-graphs (see Figs. 5.11, 5.12), one for PC cluster and the other for the Cray T3E, show this percentages from one of the examples with a grid size of $80 \times 80 \times 80$ and executed with 8 processors with a partitioning configuration of $2x \times 2y \times 2z$.

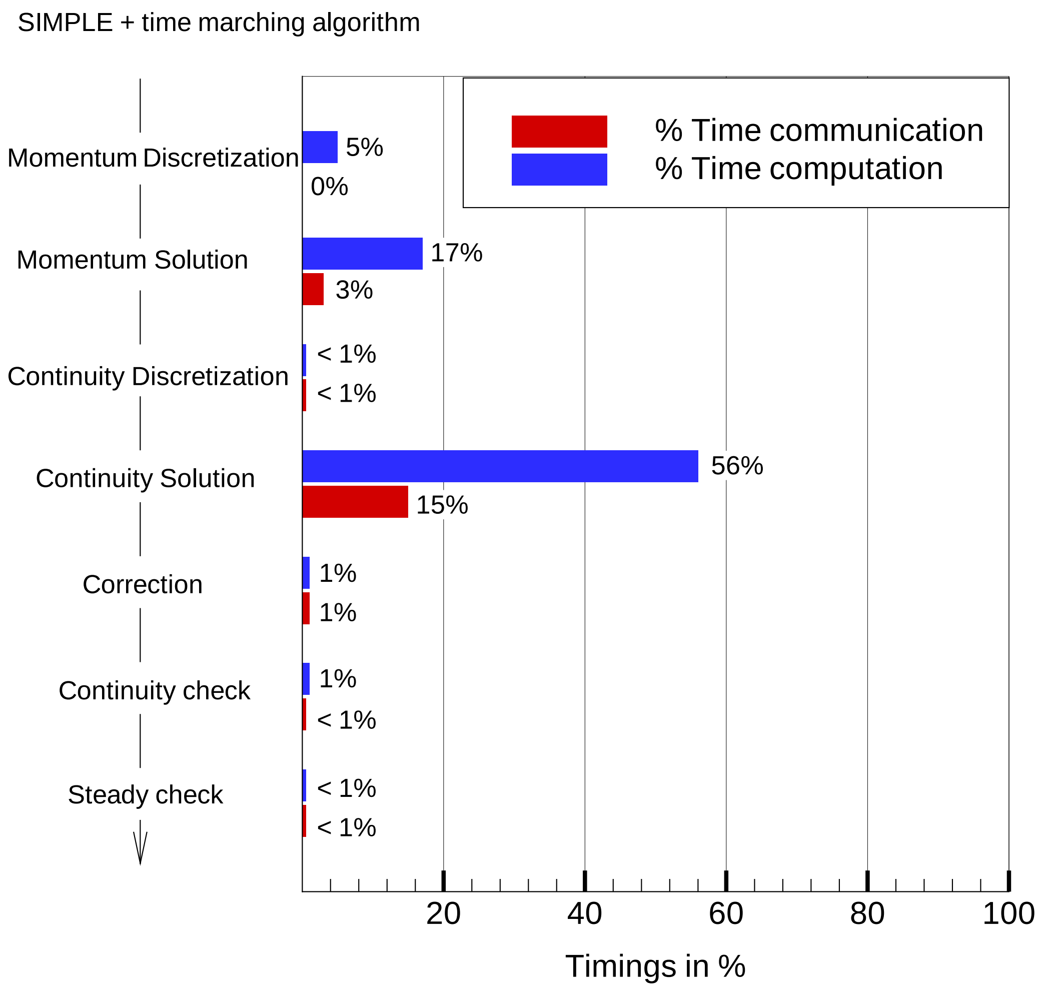

Figure 5.11: Timings in % of SIMPLE at PC cluster

At glance, there are no significative differences between both machines. The results show that roughly the $60-70\%$ from the overall wall clock time of the simulation is carried out at the solution of the continuity equation, meanwhile the remaining time is distributed among the solution of the Navier-Stokes equations, the correction of fluid flow variables, and in less percentage, in the continuity and the steady checkings.

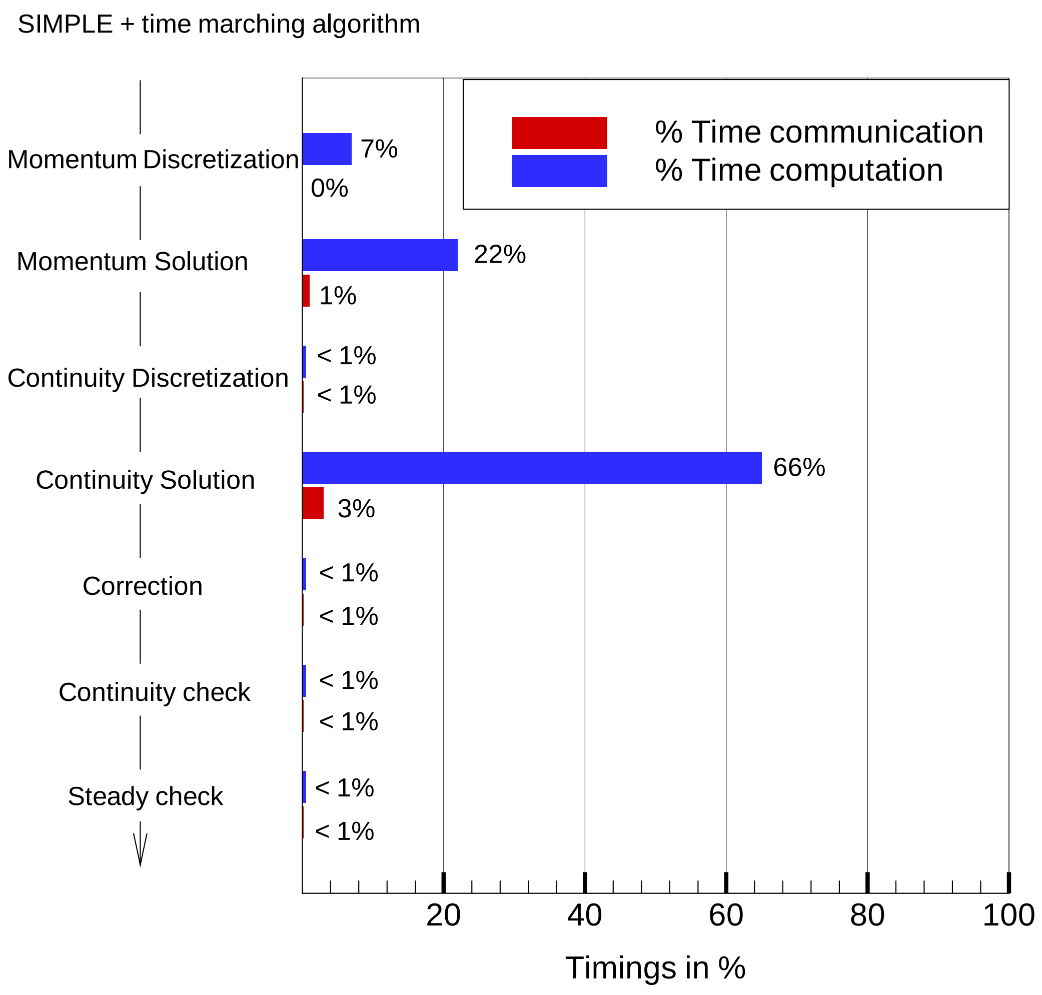

Figure 5.12: Timings in % of SIMPLE in the Cray T3E

This $60-70\%$ of time is composed by a percentage of time of computation and a percentage of time of communication that differ from one to the other machine. On PC cluster, the 56% of time is taken in the computation and the 15% is taken in the communication. On the Cray T3E, the 66% of time is taken in the computation and only the 3% of the time is taken in the communication. Clearly, this is the main difference between both machines, i.e., the ratio of the time of computation with the time of communication.

From the optimization point of view of the algorithm, it is remarkable that a reduction of the times of computation and communication of the solution of the continuity equation would reduce globally the overall wall clock time of the simulation.

Therefore, an analysis more detailed of the operations involved within the BiCGSTAB preconditioned with MSIP enable us to evaluate the bottlenecks that affect to the overall time.

The two bar-graphs 5.13,5.14 show the percentages of time of computation and communication of each algebraic operation involved within the solver respect to the wall clock time of the solver, i.e., the $60 - 70\%$ of the overall time of the simulation.

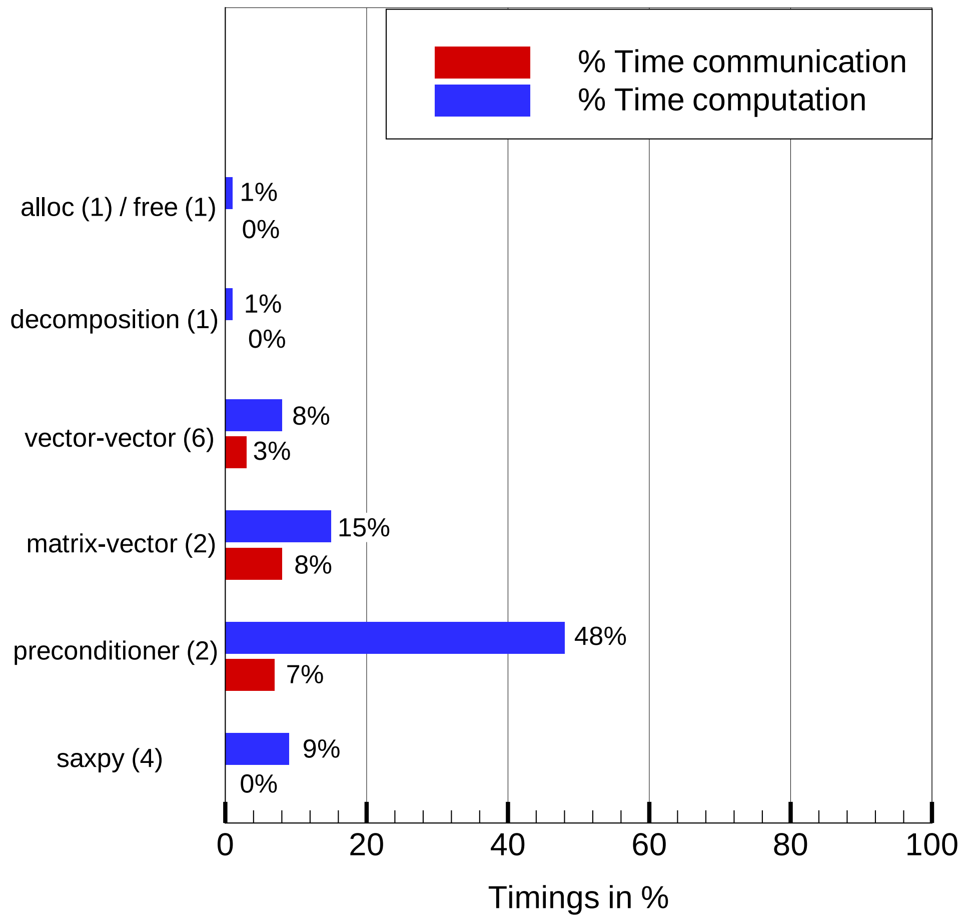

Figure 5.13: Timings in % of BiCGSTAB at PC cluster

Both graphs show analogous percentages for each algebraic operation. Clearly, the largest percentage is related with the preconditioner. The 60% of the time is taken within the preconditioner, the 20% in matrix-vector product operations and the 10% of time in the vector-vector product. All of them require communications. It is worth noting that the time of computation of the incomplete decomposition is small compared to the above timings.

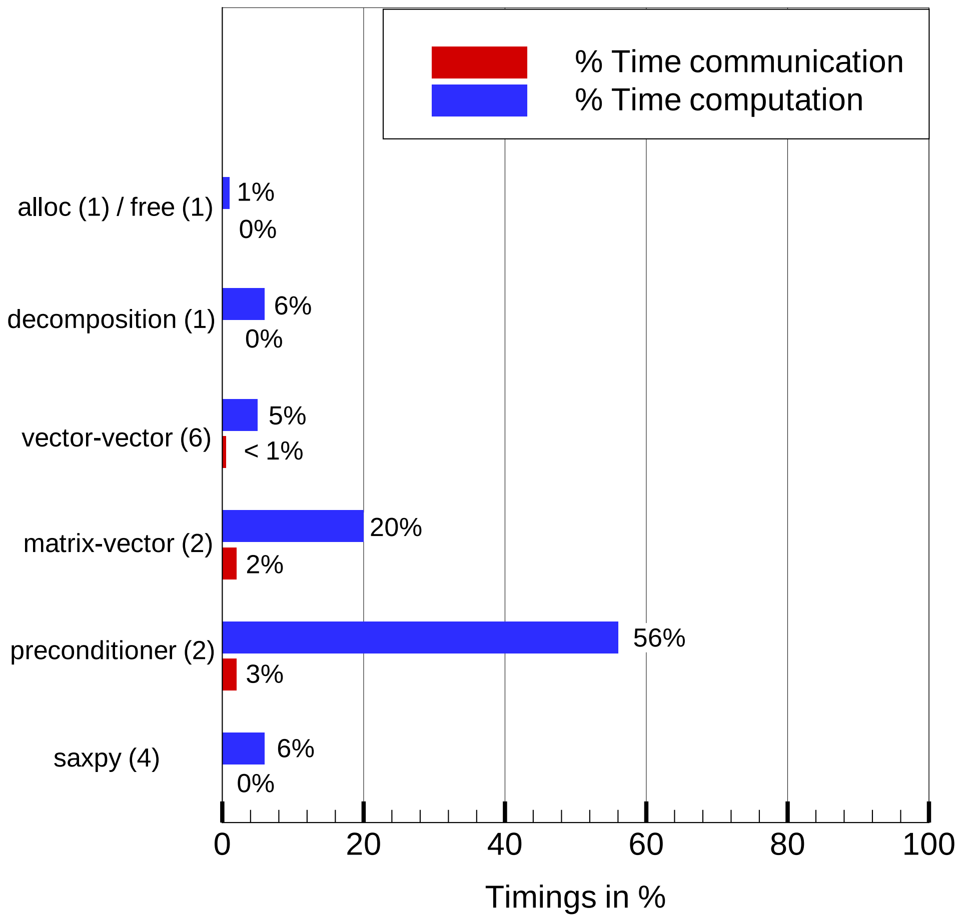

Figure 5.14: Timings in % of BiCGSTAB at Cray T3E

Therefore, the bottleneck arises in the preconditioner that not only it takes the longest time of computation and communication but also, since its algorithm is inherently sequential, there is a degradation of the efficiency when the number of processors is increased at each direction.

5.4 Other alternatives not based on Krylov solvers

As seen in previous section, the time of computation needed for the solution of the continuity equation is critical for the overall time of computation of the simulation. Research efforts in parallel preconditioners such as SPAI and polynomial preconditioners (see chapter 3, preconditioners section) are contributing to the improvement of the performance of Krylov solvers.

Other alternatives not based on Krylov solvers are the Schur Complement (SC) and the algebraic multigrid (ACM). In this work, BiCGSTAB has been compared with DDACM (Domain Decomposed Additive Correction Multigrid) described in (39) and SC described in (24; 62).

- SC is a direct parallel method, based on the use of non-overlapping subdomains with implicit treatment of interface conditions. Its main feature is that each processor has to solve only twice its own subdomain plus an interface problem in order to obtain the exact solution of the subdomain, with just one global communication operation.

- DDACM is a multigrid algorithm based on ACM, with both pre and post smoothing for the finest levels and the Schur Complement solver for the coarsest level. Although ACM is formulated with a recursive formulation, a non-recursive formulation is used for better control of the communications of DDACM.

5.4.1 Results of the alternatives

Since the available implementations of SC and DDACM are two dimensional and the implementation of BiCGSTAB is only three dimensional, only a qualitative comparison is possible. The extension of SC and DDACM to three dimensions is not straightforward.

The execution times compared are:

- BiCGSTAB: 3D lid driven cavity problem (described in this chapter), ϵ=10^{-4}, np=8, solved with a PC cluster described in the Appendix.

- DDACM: 2D model problem (described in (39)), ϵ=10^{-6}, np=9, solved with a 700 MHz PC cluster with a fastethernet (100Mb/sec) network.

- SC: 2D natural convection problem (described in (63)), $\epsilon \approx 10^{-11}$, np=8, solved with a PC cluster described in the Appendix.

The results of the times of computation are shown in Fig. 5.15.

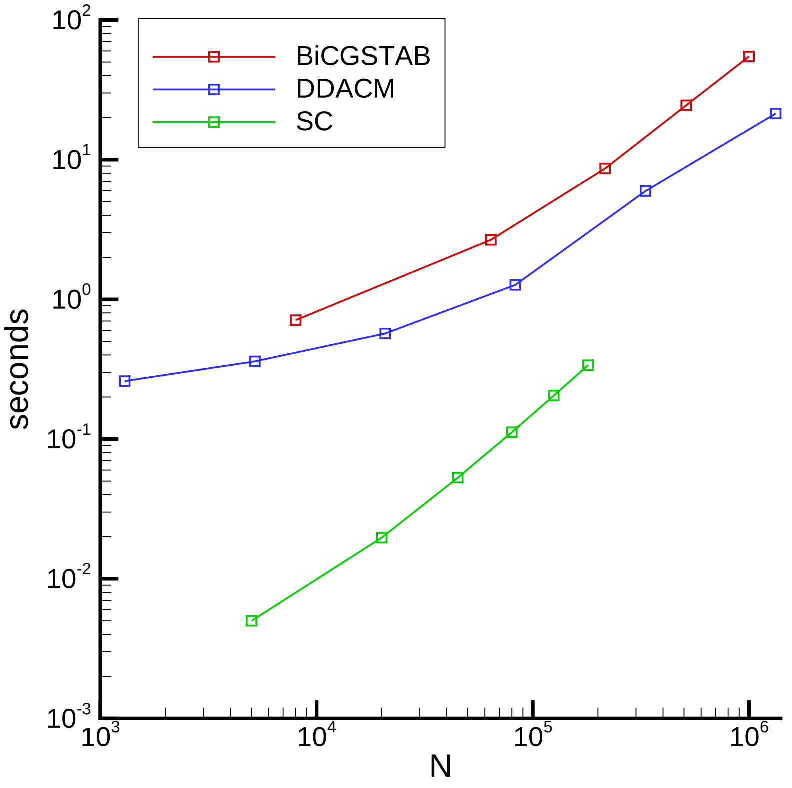

Figure 5.15: Times of computation of the continuity equation solved with the BiCGSTAB, SC and DDACM. N is the problem size.

Computation times of SC and DDACM are notably smaller than the BiCGSTAB ones. A part of this difference might be due to the differences between the problems solved. Additionally, in order to compare the different solvers, other factors have to be considered, such as preprocess time, storage requirements and algorithmic degradation with the number of processors *np*. The differences of these factors for the SC and BiCGSTAB are outlined below:

- With respect to the preprocess time, the SC takes a preprocess much higher than the solution time (on the order of hours), and hence, this solver is only feasible for constant coefficient matrices, i.e. for incompressible fluid flow problems. Therefore, SC is not applicable to compressible fluid flow problems. However, the BiCGSTAB takes a meaningless preprocess so it is applicable to both incompressible and compressible fluid flow problems, where the coefficients of the continuity equation change at each iteration.

- With respect to the storage, the requirements of memory are not a limiting factor for the BiCGSTAB, and hence, it enables the solution of large three dimensional problems. For the SC, the requirements of memory are large in two dimensional problems and even much more in three dimensional problems. See (62) for an estimation of the memory requirements with the problem size N and number of processors np.

- With respect to the algorithmic degradation with np, the preconditioner of the BiCGSTAB based on the incomplete factorization degrades with np. For the SC, the efficiency is kept for any np (but at the expense of an increase of memory).

With respect to DDACM, the preprocess, storage requirements and efficiency of DDACM are strongly dependent on the number of levels. More levels implies a reduction of both preprocess time and storage. Therefore, this reduction enables the application of DDACM to incompressible three dimensional fluid flow problems. However, the solver becomes more iterative and the number of communications per processor increases. Moreover, the high latency in networks of PC clusters reduce the efficiency with the number of levels.

5.5 Nomenclature

A	discretization matrix
a	coeff. in A
b	discretization right hand side
np	number of processors
P	pressure
Re	Reynolds number
r	residual
T	temperature
$\vec{V}$	fluid flow velocity vector
$\{u, v, w\}$	fluid flow velocity components
$\{x, y, z\}$	cartesian coordinates

Greek symbols

ϵ	precission
τ	time

Subscripts

NGB	general neighbour grid point
P	central grid point under consideration

Superscripts

(k)	k-th iteration of
τ	new value at time $\tau + \Delta\tau$ of
$*$	guessed value of
$'$	correction value of

Chapter 6

Conclusions

The detailed simulation of complex heat and mass transfer phenomena (i.e. detailed CFD simulation) requires a fine discretization of the governing equations (conservation of mass, momentum and energy) in both time and space. This leads to a set of strongly coupled non linear systems of equations with a large number of unknowns per system. Although the use of high order schemes is intended for the reduction of the size of these systems without loosing the accuracy of results, the number of unknowns per system is over several millions. E.g., a detailed three dimensional simulation of an incompressible and isothermal laminar flow within a $200 \times 200 \times 200$ grid points, leads to 4 coupled systems of equations (three velocity components $\{u, v, w\}$ and pressure P) with 8 millions of unknowns per system.

Dealing with such problems means computing the solution of these systems by combining the different numerical algorithms with the current hardware and software within a reasonable time. Hence, we have focussed our attention on solvers, parallelization by domain decomposition and computation in distributed memory machines (for instance, in clusters of personal computers).

It has been done, with respect to the solvers, a revision of those iterative methods for non symmetric matrices (hence, the symmetric matrices are considered a particular case) suitable for CFD problems: the incomplete factorizations or ILUs such as MSIP and SIS, Krylov space based solvers such as BiCGSTAB and GMRESR with their preconditioners and accelerating techniques such as the algebraic multigrid (AMG) and multiresolution analysis (MRA) with wavelets. These solvers exploit the matrix sparcity of these systems and reduce the storage in memory. Hence, GMRESR and BiCGSTAB have also been preconditioned with ILUs.

Furthermore, each of them may be used as black box and even combined among them because the operations are purely algebraic and there are no dependencies with the geometry of the problem, boundary conditions (Dirichlet, Neumann and periodic) and phenomena (the discretization of the advection and diffusion terms leads to non symmetric and symmetric coefficients within the matrices respectively).

This revision is completed with a comparative study of both the computing time in sequential and memory storage for a two dimensional problem with few thousands of unknowns. For different boundary conditions and for a wide range of convective with diffusive ratios the results show, with independence of the case, that the acceleration with either AMG or MRA improves the convergence rates but at expense of an slightly increase in memory storage. However, these techniques are unable to bear problems with several millions of unknowns in single processor machines since their processing speed (i.e. number of floating point operations per second) and storage capacity are limited.

To overcome these difficulties, the path of the parallel computation in distributed memory multiprocessors, and for instance, in PC clusters has been adopted. The data exchange among processors is performed by means of message passing software such as MPI. By comparison with conventional distributed memory supercomputers, PC clusters have powerful processors at lower cost and larger scalabilities that compensate the slower communication throughputs (i.e. high latency and low bandwidth).

A comparative study, from the parallel performance point of view, carried out on two different machine architectures, the Cray T3E and a PC cluster (see appendix), has strengthened this position. The evaluation of the parameters affecting to this performance has shown, for both machines, similar processor performances but different network througputs, being at Cray T3E 10 times faster than PC cluster.

This implies, from the design and implementation points of view of the parallel algorithms that the computational load per processor must be large and balanced, and the communication among processors must be minimized (e.g. nonblocking point to point communications), thus reducing the penalty in the performance. Hence, larger computation with communication ratios will provide better performances, i.e. better speed-ups.

For this reason, given the same computation load (number of flops) and communication load (number of communications and data to be transferred) for both computers, the performance is greater at computers with faster networks, and for instance, at Cray T3E. This fact oblige to work at PC clusters with larger problems per processor in order to keep a similar performance. E.g. in order to have a minimum efficiency of 50% in a parallel matrix-vector product, having the matrix a 7-point stencil, the estimation of the minimum size per processor is $20 \times 20 \times 20$ at Cray T3E and $30 \times 30 \times 30$ at PC cluster.

Indeed, this strategy is concerned with the domain decomposition of the problem and the application of the numerical algorithms at each subdomain. Thus, the discretization of the governing equations over the decomposed domain leads to a set of coupled systems of equations composed by matrices and vectors distributed among processors in block matrix and block vector structures respectively.

Krylov space based solvers are suitable algorithms for the parallel solution of these systems since the involved algebraic operations (matrix-vector and vector-vector products) may be computed efficiently in parallel. However, these solvers need a preconditioner for better convergence behaviour. In this work, it has been used an incomplete factorization wich exploits the sparsity of block matrices. The drawback of the preconditioned solvers is the loss of efficiency or degradation of such preconditioners when the number of processors is increased. This is due to the evaluation whithout communications of local factorizations (at each subdomain), and hence, losing global convergence rate.

In this sense, a numerical experiment has been performed by partitioning a large three dimensional model problem (i.e. a large linear system of equations) into several directions. The results have pointed out, with independence of the computer, the degradation of the parallel algorithms with the number of processors and their limited performance. The best results have been obtained with BiCGSTAB preconditioned with MSIP.

Finally, these techniques have been applied to the solution of large coupled systems of equations derived from a well known CFD problem: the three dimensional isothermal lid driven cavity flow for laminar Reynolds numbers. The detailed simulation with a $120 \times 120 \times 120$ grid has been carried out at PC cluster. It has been successfully benchmarked with both experimental and numerical simulation (in single processor) data provided by other authors. By using MSIP for the momentum equations and BiCGSTAB preconditioned with MSIP for the continuity equation, the simulation has been computed for several processors (from 1 up to 12 processors) and partitioning configurations.

The most significative results are speed-ups of 7 with 8 processors and 10 with 12 processors. Slightly better speed-ups have been obtained at Cray T3E with differencies of 15% at most of the overall time of simulation.

From the timing analysis of SIMPLE algorithm it is shown, as expected, that the most time consuming part (roughly the 60-70%) is wasted in the solution of the continuity equation. Therefore, any improvement on numerical algorithms aimed to the computing time reduction in the solution of the continuity equation, will represent a notably reduction in the overall time of simulation.

In this sense, alternative methods such as Algebraic Multigrid and Schur Complement have been qualitatively compared with BiCGSTAB. The comparison has shown that these alternatives provide better speed-ups. However, they present several constraints such as the application only to incompressible fluid flow problems, the need of a large preprocess step and the requirement of a large storage capacity.

Hence, further improvements may be stressed in several in combinations of the above mentioned methods. Following this working line, the research should be focused on efficient parallel preconditioners with the number of processors, and following the cited alternatives, on the reduction of the preprocess step and storage requirements.

Appendix A

Facilities

A.1 JFF PC cluster at CTTC

The JFF PC cluster is sited at CTTC (Centre Tecnològic de Transferència de Calor) in Terrassa (Spain). This acronym has been adopted in memorial of Joan Francesc Fernandez, computer science professor of the Polytechnic University of Catalonia. The JFF cluster is a network of 35 commodity processors connected with fast ethernet NICs (Network Interface Cards) and two full duplex switches. Each processor runs the operating system Linux with a minimal installation with a source Message Passing library.

Details of the cluster configuration are given in table A.1.

element	type	model
main board	35×ASUS a7V	AMD-751 chipset
processor (32bits)	5×600MHz + 28×900MHz	AMD-K7
cache (L1+L2)	128KB of L1 + 512KB of off-chip L2	
RAM	2×1GB + 33×512MB	SDRAM 133MHz
HD	25GB on server + 34 × 18GB	IDE
SWAP	double RAM per processor	
NICs (100Mb/s)	2 on server + 34 × 1	3C905B-TX-NM
switch (100Mb/s)	24 full duplex ports, scalable	3C16980
matrix cable	1 matrix cable connecting 2 switches	3C16965
OS	Linux	Debian v2.2.17
batch system	none	
MP library	MPI	LAM 6.1
compilers	C,C++,F77	gcc v2.7, g77 v0.5.24

Table A.1: Cluster configuration

Few pictures A.1,A.2 and A.3 of the JFF cluster have been attached to show the distribution and connection of the commodity PCs and switches on shells.

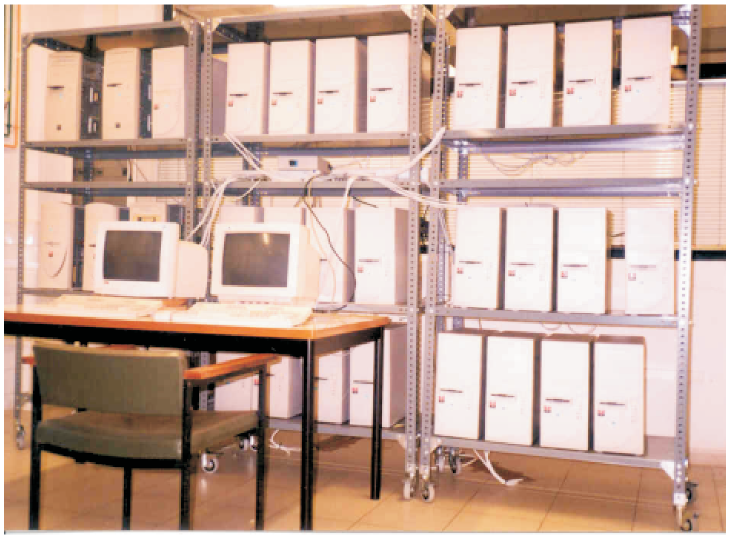

Figure A.1: Front view of the JFF cluster

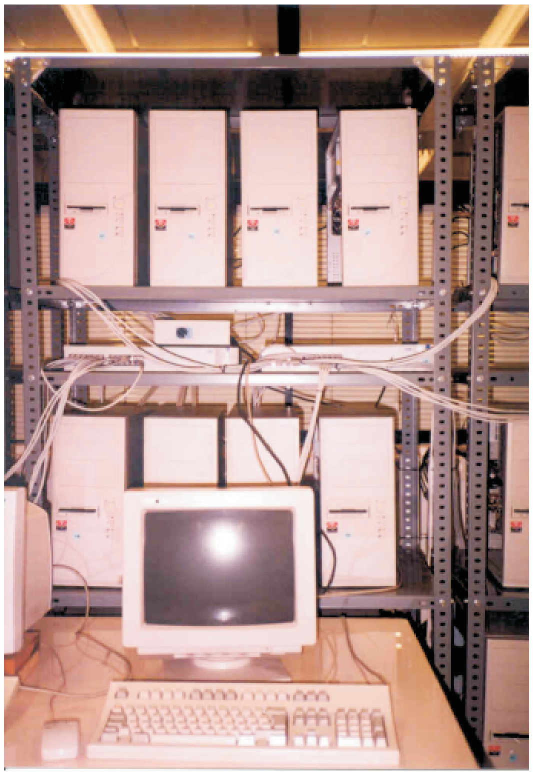

Figure A.2: Front of the switches and connections

Figure A.3: Back of the switches and connections

In order to manage the cluster, there is a server node with x-server and equipped with two NICs (Network Interface Cards), one connected to the 34 nodes of the private net and the other connected to the public network. Users access to the cluster on this second NIC. Tasks of compilation and debugging may be done in this node. The compiler and loader for sequential jobs is the `gcc`, with the following flags and mathematic library:

```
> gcc -O3 -w -Wall -pedantic source-files.c -o object-files.o
> gcc object-files.o -lm executable-file
```

For parallel jobs, the compiler and loader is the `hcc`,with the MPI library:

```
> hcc -O3 -w -Wall -pedantic source-files.c -o object-files.o
> hcc object-files.o -lm -lmpi executable-file
```

After that, directories and executable files needed for execution are copied to nodes with the remote commands `rsh` and `rcp`.

Since we are in the beginnings of the JFF cluster, there is not yet a batch system. Therefore the execution of sequential and parallel jobs is done either in interactive mode or in background with the command `nohup`.

For sequential jobs the following command line is used:

```
> nohup /work-diretory/executable-file &
```

For parallel jobs with LAM MPI library three steps must be followed:

```
> lamboot -v nodes
> nohup mpirun -v -c np -O -w /work-directory/executable-file &
> wipe -v nodes
```

The first step initialize the lam daemon process and 'awakes' the np node-processors listed in the nodes file. The second step runs with the `mpirun` command np copies of the executable file in np processors placed in the work directory. The `-O` flag tells to the lam daemond that the network of processors is homogeneous (it does not data conversion among nodes), and the `-w` flag waits for all process to complete before exiting `mpirun` and reports abnormal exit codes. Once the execution has ended, the daemon process is killed with the `wipe` command.

Although Linux is a multiprocess operating system it is desirable to minimize the number of processes for performance measures (i.e. timings of subroutines). During these measures, each process whether it is originated by the operating system (i.e. the daemons) or by the executable file (i.e. the output of results) produce perturbations in these measures. For this reason every experiment must be repeated many times to give an averaged measure.

A.2 SGI/Cray T3E at CIEMAT

The SGI/Cray T3E supercomputer of 32 air cooled nodes is sited at CIEMAT (Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas) in Madrid (Spain). Details of the configuration are given in table A.2.

element	type	model
processor (64bits)	32×300MHz, super scalar RISC	DEC 21164
cache (L1+L2)	8KB of L1 + 96KB 3-way of L2	
RAM	32×128MB	DRAM
HD	130GB distributed for every 8np	Giga Ring, SCSI
SWAP	none	
net (1Gb/s)	3-D torus of low latency	
OS	Unix	Unicos/mk v2.0.4.34
batch system	Network Queuing Environment	NQE v3.3
MP library	PVM,MPI	optimized for Cray
compilers	C,C++,F77,F90	optimized for Cray

Table A.2: SGI/Cray T3E configuration

The SGI/Cray T3E has two processors dedicated to the surveillance and maintenance of the global system (i.e. the Global Resource Manager) and file server for the rest of nodes. Four processors for interactive tasks such as user sessions, compiling and debugging. Only they support multiprocess (time sharing). The rest of processors (i.e. 26 processors) do parallel applications throughout the command `mpprun`. Each processor accepts only one process and the total amount of memory available without swap is 128MB.

Sequential and parallel jobs are submitted to a batch system (for instance, the Network Queuing Environment) using the following script, called `job.sh`:

```
# QSUB -l mpp_p=np limit
# QSUB -l mpp_t=hh:mm:ss limit
# QSUB -eo
# QSUB -o job.sh
cd /work-directory
mpprun -n np executable-file
```

Further details may be found at the Ciemat's web site www.ciemat.es.

Appendix B

Software layers in CFD code

B.1 Description of layers

The CFD code is structured in four layers: the user layer, the solver layer, the algebra layer and the communication layer. These layers are linked and integrated to build an structure which looks like an iceberg (see Fig. B.1).

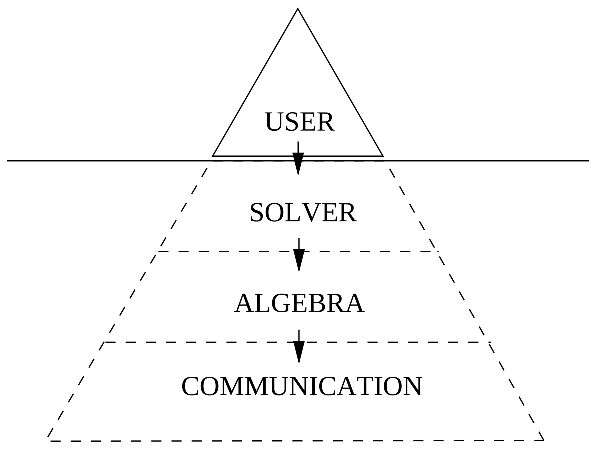

Figure B.1: Iceberg built by layers: user layer, solver layer, algebra layer and communication layer.

The concept of the iceberg comes from the idea that only the top of the iceberg or the last layer is visible while the rest of the iceberg or layers are hidden below the water. This top end is called the user layer, the only layer visible to the end user of the CFD code. The rest of layers are hidden from the user, i.e. the user does not access to the remaining layers. In order to build the CFD code in layers it is very important to specify clearly which are the subroutines of each layer and which layer is supported by another, i.e, to define the dependencies between layers in only one direction. This dependency goes from the top to the base as shown with the arrows in the in figure B.1.

The description of these layers is given below.

- The user layer can be structured into a set of modules, each of them with different functions. For example the graphic module, the data module and the model module. The graphic module represents the front-end of the CFD code. The data module contains all information needed for running a case. These, a specific information of the case (e.g. geometry, boundary conditions, initial conditions, properties of fluid and flow), and a specific information about how to run the case (e.g. algorithm or procedure, solver, iterations, convergence criteria) must be included. The model module performs the translation from the real case to the set of algebraic systems of equations (e.g. finite volume methods, schemes, set of physic hypotheses). This translation is the discretization of the partial differential equations involved in the case. Since this work has been focused on how to solve such algebraic systems by using solvers and parallel computing, the above modules are compacted in the concept of the user layer.

- The solver layer performs the evaluation of the solution of the algebraic systems of equations. This layer specifies a wide range of solvers based on classic methods (e.g. Jacobi, TDMA, Gauss), decompositions (e.g. complete and incomplete LU decompositions), Krylov space based methods helped by preconditioners (e.g. CG, BiCGSTAB, GMRESR) and acceleration techniques (e.g. Algebraic MultiGrid and MultiResolution Analysis with wavelets).

- Each of these solvers contains some common basic operations which have been integrated in a more basic layer so called algebra layer. By this, we refer mainly to the algebraic operations between vectors and matrices (e.g. matrix-vector product, inner product between vectors, addition and subtraction between vectors, norm of vectors), and some transformations over vectors, matrix and maps of scalars (e.g. discrete Fourier transform, discrete wavelet transform).

- The above layers are supported by the communication layer. It is also called the communication layer because its main task is the communication inside a layer between processors: mainly the domain decomposition at the user, solver and algebra layers . If the CFD code is thought in sequential, this layer can be eliminate at all, but in parallel, part of the task are done at the communication layer. For example, the parallel algebraic operations are performed in the algebra layer except the exchange of data among the processors that is performed in the communication layer (e.g. the matrix-vector product, the norm of a vector and the maximum or the minimum value of a map distributed among the processors). The subroutines embedded in this layer contain calls to the parallel library MPI. It is worth noting that the from the programming point of view of the CFD code it is based on the SPMD (Single Program and Multiple Data) paradigm.

Most of these subroutines have been summarized for each layer in Fig. B.2.

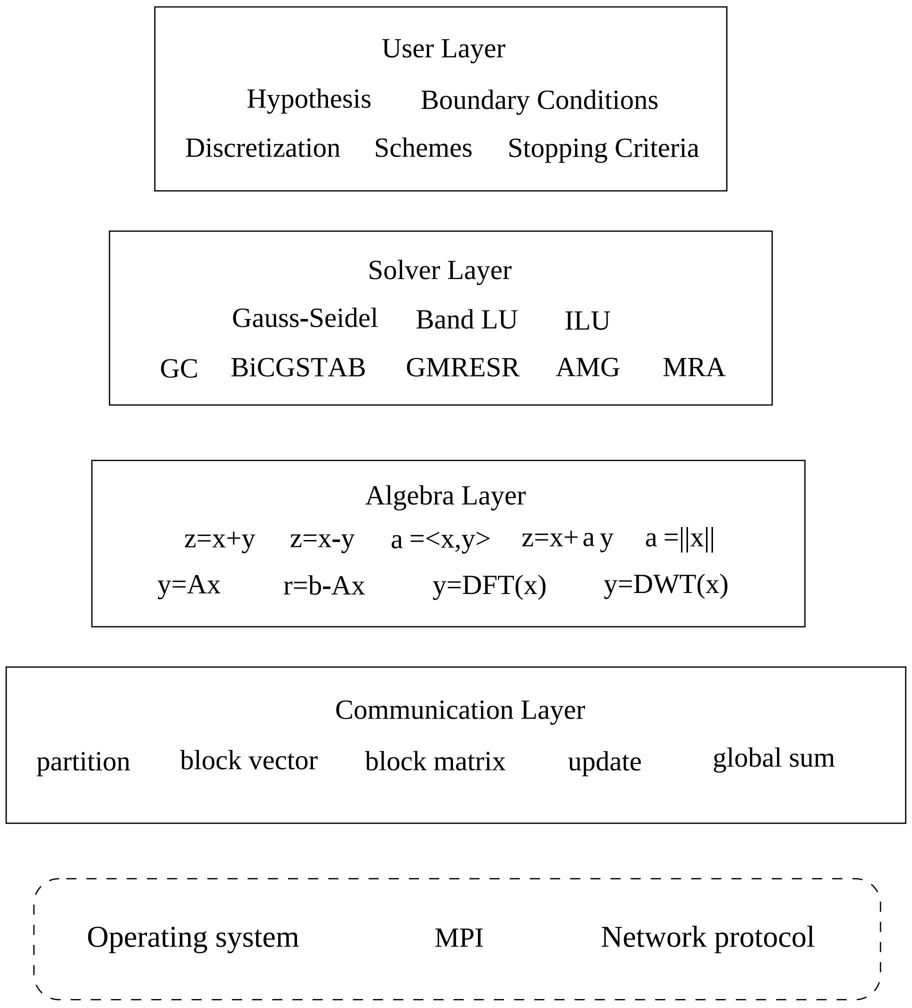

Figure B.2: Main subroutines of the CFD code grouped by layers.

Bibliography

[1] M. Costa, A. Oliva, and C. Pérez Segarra, "A three-dimensional numerical study of melting inside a heated horizontal cylinder," *Numerical Heat Transfer*, vol. Part A, no. 32, pp. 531–553, 1997.

[2] H. Schweiger, A. Oliva, M. Costa, and C. Pérez Segarra, "Numerical experiments on laminar natural convection in rectangular cavities with and without honeycomb-structures," *International Journal of Numerical Methods for Heat and Fluid Flow*, vol. 5, no. 5, pp. 423–445, 1995.

[3] M. Soria, H. Costa, M. Schweiger, and A. Oliva, "Design of multifunctional ventilated facades for mediterranean climates using a specific numerical simulation code," in *Eurosun98 Conference, Slovenia*, 1998.

[4] C. Pérez Segarra, J. Cadafalch, J. Rigola, and A. Oliva, "Numerical study of turbulent fluid-flow through valves," in *International Conference on Compressors and Their Systems, London*, 1999.

[5] A. Ivancic, A. Oliva, C. Pérez Segarra, and M. Costa, "Heat transfer simulation in vertical cylindrical enclosures for supercritic rayleigh number and arbitrary side-wall conductivity," *International Journal of Heat and Mass Transfer*, vol. 42, no. 2, pp. 323–343, 1999.

[6] Myricom, "Creators of myrinet." http://www.myri.com.

[7] M. P. I. Forum, "Mpi forum, mpi: A message-passing interface standard." http://www.mpi-forum.org.

[8] M. Zedan and G. Schneider, "A coupled strongly implicit procedure for velocity and pressure computation in fluid flow problems," *Numerical Heat Transfer, part B*, vol. 8, pp. 537–557, 1985.

[9] X. Chuan and D. Keyes, "Nonlinear preconditioned inexact newton algorithms," 2000. Technical Report, Departement of Computer Science, University of Colorado, Boulder.

[10] H. Stone, "Iterative solution of implicit approximation of multidimensional partial differential equations," *SIAM Journal of numerical analysis*, vol. 5, pp. 530–558, 1968.

[11] M. Zedan and G. Schneider, "A three-dimensional modified strongly implicit procedure for heat conduction," *AIAA*, vol. 21, pp. 295–303, Feb 1983.

[12] M. Peric, "An efficient semi-implicit solving algorithm for nine-diagonal coefficient matrix," *Numerical Heat Transfer*, vol. 11, no. Part B, Fundamentals, pp. 251–279, 1987.

[13] S. Lee, "A strongly implicit solver for two dimensional elliptic differential equations," *Numerical Heat Transfer*, vol. 16, no. Part B, Fundamentals, pp. 161–178, 1989.

[14] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices.* New York: Oxford University Press, 1986.

[15] C. Craig, "Mgnet." Web site www.mgnet.org.

[16] P. Wesseling, *An Introduction to Multigrid Methods.* Pure and Applied Mathematics, John Wiley and Sons, 1992.

[17] P. Zeeuw, *Acceleration of Iterative Methods by Coarse Grid Corrections.* PhD thesis, Amsterdam University, 1997.

[18] C. Wagner, "Introduction to algebraic multigrid," 1999. Course Notes of an Algebraic Multigrid Course, Heidelberg University.

[19] J. Mora, M. Soria, and A. Oliva, "Uso de multigrid algebráico para la resolución de los sistemas de ecuaciones discretos obtenidos en transferencia de calor y dinámica de fluidos," in *XIII Congreso Nacional de Ingeniería Mecánica*, 1998.

[20] J. Williams and K. Amaratunga, "Introduction to wavelets in engineering," *I. J. Numerical Methods in Engineering*, vol. 37, pp. 2365–2388, 1994.

[21] R. Wells and X. Zhou, "Wavelet interpolation and approximate solutions of elliptic partial differential equations," 1992. Technical Report 92-03, Computational Mathematics Laboratory, Rice University.

[22] A. Rieder, R. Wells, and X. Zhou, "A wavelet approach to robust multilevel solvers for anysotropic elliptic problems," 1992. Technical Report 93-07, Computational Mathematics Laboratory, Rice University.

[23] R. DeVore and B. Lucier, "Wavelets," 1992. Technical Report, Preprint 92-026, University of Minnesota.

[24] E. Simons, "Domain decomposition methods for separable elliptic equations suitable for les of complex flows," 1995. Technical Report, von Karman Institute for Fluid Dynamics, Belgium.

[25] Y. Saad, "Krylov subspace method for solving large unsymmetric linear systems," *Math. Comput.*, vol. 37, pp. 105–126, 1981.

[26] Y. Saad and J. Zhang, "Diagonal threshold techniques in robust multi-level ilu preconditioners for general sparse linear systems," *Numerical Linear Algebra*, vol. 6, no. 4, pp. 257–280, 1999.

[27] A. Basermann, B. Reichel, and C. Schelthoff, “Preconditioned cg methods for sparse matrices on massively parallel machines,” *Parallel Computing*, vol. 26, pp. 381–398, 1997.

[28] M. Grote and T. Huckle, “Parallel preconditioning with sparse approximate inverses,” *SIAM J. Scientific Computing*, vol. 18, no. 3, pp. 838–853, 1997.

[29] M. Benzi and M. Tuma, “A sparse approximate inverse preconditioner for nonsymmetric linear systems,” *SIAM J. Sci. Comput.*, vol. 19, pp. 968–994, 1998.

[30] S. Zweben, S. Edwards, B. Weide, and J. Hollingsworth, “The effects of layering and encapsulation on software development and quality,” *IEEE Transactions on software engineering*, vol. 21, no. 3, pp. 200–208, 1995.

[31] W. Rohsenow, J. Hartnett, and E. Ganic, *Handbook of Heat Transfer Fundamentals.* Mc. Graww-Hill Book Company, 1985.

[32] S. Patankar, *Numerical Heat Transfer and Flow.* New York: Hemisphere, 1980.

[33] J.H.Ferziger and M.Peric, *Computational Methods for Fluid Dynamics.* Springer-Berlag, 1996.

[34] M. Darwish and F. Moukalled, “The normalized variable and space formulation methodology for high resolution schemes,” *Numerical Heat Transfer*, vol. Part B, fundamentals, no. 26, pp. 79–96, 1994.

[35] J. Van Doormal and G. Raithby, “Enhancements of the simple method for predicting incompressible fluid flows,” *Numerical Heat Transfer, part B*, vol. 7, pp. 147–163, 1984.

[36] K. Aksevol and P. Moin, “Large eddy simulation of turbulent confined coanular jets and turbulent flow over a backward facing step,” 1992. Technical Report TR-63, Research Center of Turbulence,U.S.

[37] J. Mora, “A nine point formulation of poisson equation with a fourth order scheme,” 2000. Technical Report of CTTC, Polytechnical University of Catalonia.

[38] M. Quispe, J. Cadafalch, M. Costa, and M. Soria, “Comparative study of flow and heat transfer periodic boundary conditions,” in *ECCOMAS*, 2000.

[39] M. Soria, *Parallel Multigrid Algorithms for Computational Fluid Dynamics and Heat Transfer.* PhD thesis, Polytechnical University of Catalonia, 2000.

[40] R. Barrett *et al.*, “Templates for the solution of linear systems: Building blocks for iterative methods.” Downloadable document at ftp.netlib.org/templates/templates.ps.

[41] S. Rump, “Ill conditioned matrices are component wise near to singularity,” *SIAM Review*, vol. 41, no. 1, pp. 102–112, 1999.

[42] W. Spotz and G. Carey, “A high-order compact formulation for the 3d poisson equation,” *Numerical Methods for Partial Differential Equations*, vol. 12, pp. 235–243, 1996.

[43] W. Press *et al.*, *Numerical Recipes in C. The art of Scientific Computing.* Cambridge University Press, 1994.

[44] C. Vuik, "Solution of the discretized icompressible navier-stokes equations with the gmres method," *I. J. for Numerical Methods in Fluids*, vol. 16, pp. 507–523, 1993.

[45] H. Vorst, "Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 13, pp. 631–644, 1992.

[46] M. Saad, Y. ahd Schultz, "Gmres: A generalized minimum residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, pp. 856–869, 1986.

[47] H. Vorst and C. Vuik, "Gmresr: A family of nested gmres methods," 1991. Technical Report, Delft University of Technology.

[48] M. Trummer, "Iterative methods in linear algebra," 1997. Technical Report, Dept. Mathematics and Statistics. Simon Fraser University.

[49] H. Vorst and G. Sleijpen, "The effect of incomplete decomposition preconditioning on the convergence of conjugate gradients.," 1992. Incomplete Decompositions, Proceedings of the Eight GAMM Seminar, Delft University of Technology.

[50] S. Dupont and J. Marchal, "Preconditioned conjugate gradients for solving the transient boussinesq equations in three-dimensional geometries," *I. J. for Numerical Methods in Fluids*, vol. 8, pp. 283–303, 1988.

[51] G. Larrazábal and J. Cela, "Study of spai preconditioners for convective problems," 1999. Technical Report, Dept. Computer Architecture. Polytechnical University of Catalonia.

[52] B. Hutchinson and G. Raithby, "A multigrid method based on the additive correction strategy," *Numerical Heat Transfer*, vol. Part B, no. 9, pp. 511–537, 1986.

[53] I. Daubechies, *Ten Lectures on Wavelets.* Pure and Applied Mathematics, Philadelphia: SIAM, 1992.

[54] S. Mallat, "A theory for multiresolution signal decomposition the wavelet representation," *Communications in Pure and Applied Mathematics*, vol. 41, pp. 674–693, 1988.

[55] J. Xu and W. Shann, "Galerkin-wavelet methods for two-point boundary value problems," *Numerical Mathematics*, vol. 63, pp. 123–144, 1992.

[56] J. Dongarra and T. Dunigan, "Message passing performance of various computers," 1996. Technical Report, University of Tennessee. Oak Ridge National Laboratory.

[57] G. Editorial, "Parallel computing on clusters of workstations," *Parallel Computing*, vol. 26, pp. 295–303, 2000.

[58] J. Cadafalch *et al.*, “Domain decomposition as a method for the parallel computing of laminar incompressible flows,” 1996. Proceedings of the Third ECCOMAS Computational Fluid Dynamics Conference.

[59] H. Gilbert *et al.*, “Sparse matrices in matlab: design and implementation,” *SIAM J. Matrix Analysis Applications*, vol. 13, no. 1, pp. 333–356, 1992.

[60] J. Koseff, “On end wall effects in a lid driven cavity flow,” *Journal of Fluid Engineering*, vol. 106, pp. 385–389, 1984.

[61] H. Gortler, “On the three-dimensional instability of laminar boundary layers on concave walls,” *NACA Techinical mem.*, no. 1375, 1954.

[62] M. Soria, C. Perez-Segarra, and A. Oliva, “A direct algorithm for the efficient solution of the pressure-correction equation of incompressible flow problems using loosely coupled parallel computers,” *(Submitted) Numerical Heat Transfer*, vol. Part B, Fundamentals, 2001.

[63] S. Paolucci, “Direct numerical simulation of two-dimensional turbulent natural convection in an enclosed cavity,” *Journal of Fluid Mechanics*, vol. 215, pp. 229–262, 1990.