POLITECNICO DI MILANO
DEPARTMENT OF MATHEMATICS
DOCTORAL PROGRAMME IN PH.D. COURSE IN MATHEMATICAL
MODELS AND METHODS FOR ENGINEERING

HPC SIMULATION OF SEDIMENTARY BASIN

Doctoral Dissertation of:
Nur Aiman Fadel

Supervisor:
Prof. Luca Formaggia

Tutor:
Prof. Luca Formaggia

Chair of the Doctoral Program:
Prof. Roberto Lucchetti

2013 – XXV

Acknowledgements

I would like to acknowledge the support of ENI SpA and CINECA and in particular I wish to thank Dr. Carlo Cavazzoni and Prof. Giovanni Erbacci and their staff for their help and fruitful discussions.

I also wish to gratefully thank Dr. Michel Kern and his staff for their collaboration and their kindness during my stay in Maison de la Simulation, Paris.

A special thanks to all the people of MOX, in particular past and present Tender's crew: Antonio, Anna, Franco, Ilaria, Guido, Marianna, Alberto, Domenico, Anwar, Mirko, Minou, Bianca, Davide, Mattia P., Matteo, Marco V., Andrea M. for their friendship.

I am grateful to Dr. Silvetris, Dr. Filippini, Dr. Bossi and his staff who gave me the opportunity to be here. Finally, I wish to thank my parents and my brother for their support through out the last three years.

Last but not least I thank my advisor Prof. Luca Formaggia for his help, advice and encouragement.

Abstract

THE upper part of the Earth crust is modelled as superimposed regions of homogeneous rock with constant physical properties. On geological time frames, rocks can be considered fluids with very high viscosity and their evolution can be described by the Stokes equations.

To solve the problem in a reasonable time it is important to develop a parallel implementation. In this work, the solution of the Stokes problem is coupled with a suitable algorithm to track the interfaces between rock layers. The parallel implementation of this solver presents some challenges: we need to find suitable parallel preconditioner for the Stokes problem and to devise an efficient strategy for the set of hyperbolic equations governing the interface tracking. Simulations have been performed on several HPC architectures to test and optimize the proposed solutions. Details on performance and scalability are given.

Summary

NUMERICAL simulations of large scale sedimentary basins in their geological evolution are a topic of great interest in oil industry. The concern is a direct consequence of the strong correlation, assessed by geological studies, between salt domes and the formation of oilfields in their proximity.

Mathematical models and numerical tools in this field have seen a steady development in the last years [23, 24, 30, 32]. Full three-dimensional simulations are required to reproduce the evolution of realistic portions of the upper part of the Earth crust, that are of the size of a few kilometres in each direction. Under these constraints the number of unknowns of the resulting discretized problem is typically very large (from 40 million degree of freedom up to 1-2 billion) and therefore parallel techniques become a necessity to achieve reasonable simulation times.

Our work is indeed strongly based on [24, 30], where a serial version of the algorithm for the simulation of sedimentary basins modelled as stratified fluids has been presented and analysed. This work will focus on the parallel implementation. It tackles the problem of suitable parallel preconditioning techniques and the porting and testing on High Performance Computing machines.

The work is organized as follows: in chapter 1 we introduce the geological problem, in chapter 2 we detail the mathematical problem and its numerical formulation. In chapter 3 we illustrate the tracking algorithm, while in chapter 4 and 5 we introduce respectively the preconditioning techniques, the performance metrics and the machines used. Then in chapter 6

we discuss the results, while in the chapter 7 we explain the parallel implementation. Finally in chapter 8 we show an applicative example.

Contents

CHAPTER 1

Geological model of a sedimentary basin

1.1 Introduction

The term **sedimentary basin** is used to refer to any geological feature exhibiting subsidence and consequent infilling by sedimentation, as shown in Figure 1.1 where some geological features of sedimentary basin are outlined. Indeed sedimentary basins consist of stockpiles of gravel, sand, rocks and biological remains that have been transported by natural agents like wind, rivers, glaciers and sea. A typical example of a sedimentary basin is an alluvial plain where the erosion of the surrounding mountain ranges provides most of the deposited sediments.

The physical characteristics of the layers evolve with the geological eras that can last tens of millions of years. Typically the type of deposited sediments changes suddenly in the geological time scale framework, creating well defined sedimentary layers of almost homogeneous material. This stratified configuration is clearly visible in Figure 1.2. Consequently, the mixing between layers is very limited and the interfaces are called **horizons**. The sedimentary basin is delimited by the top surface, the basement and a lateral contour which conventionally limits the area of interest. The basement is a solid compacted layer and can be either continental or oceanic

(many sedimentary basins develop under the surface of the oceans). It is the bed over which the other sediments (called overburden) lie and it has better mechanical characteristics than the overburden. The surface is the upper part of the basin. The lateral contour is usually an arbitrary boundary which delimits the area of interest and in many cases it has no physical meaning. The size of typical basins is of the order of 100 by $100\ km$ in the horizontal plane and $10\ km$ in depth.

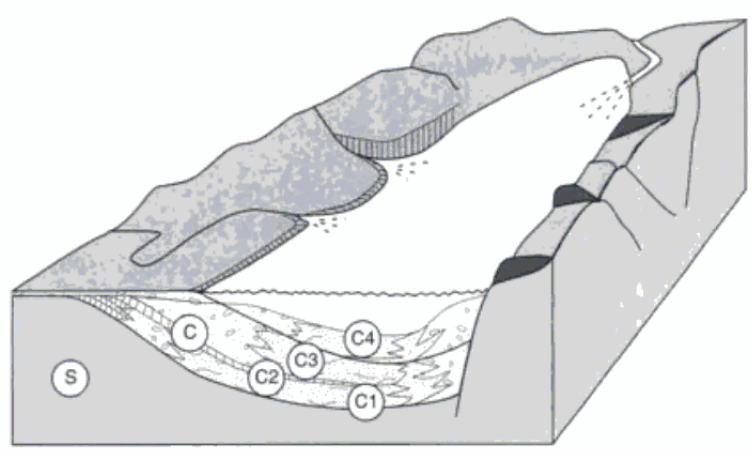

Figure 1.1: *An example of a sedimentary basin. A sedimentary basin is a hollow in the relief. Erosion products accumulates in it and gradually fill it up. Its size varies considerably from lake to ocean. The bottom is called basement of substratum (S). The sedimentary fill, or cover (C) is a sequence of layers of different kinds (C1, C2, C3, C4). The deepest layers are the oldest, lining the bottom of the initial hollow, figure from [4].*

1.2 Geology of sedimentary basin

During geological time scales sedimentary basins can experience strong deformations and also topological changes of the geometry of the layers. One of the main driving forces is the sedimentation that is the **continuous deposition of debris**. Nevertheless, there are many other forces applied from the inside and outside of the basin. For instance the movement of the basement and of the lateral edges has a great impact on the morphology of the basin.

From a tectonic viewpoint the basin is located on the upper part of the crust. Crust movements and the continental drift affect the movement of the boundary of the basin. This basin evolution could, in turn, modify the local

evolution of the crust: in fact the steady deposition of sediments causes the sinking of the basement by several kilometres in tens of millions of years. This phenomenon is also known as **flexural lithostasis**.

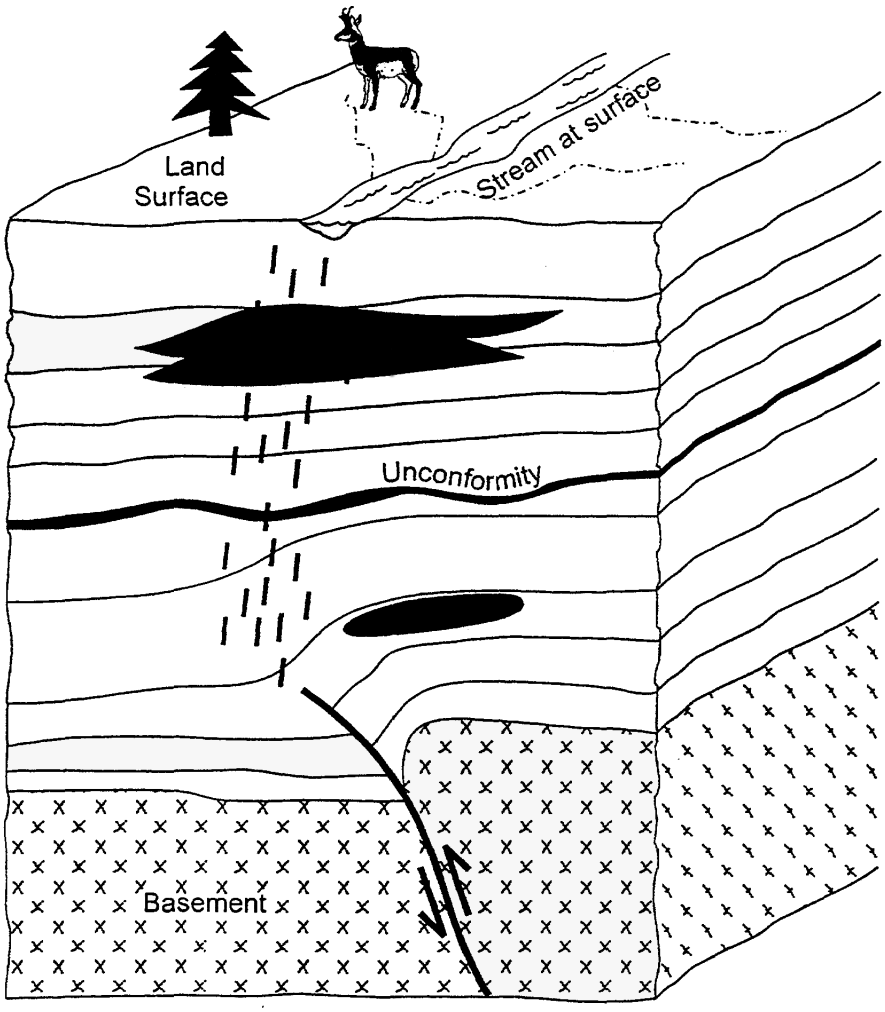

Figure 1.2: *A schematic view of a sedimentary basin illustrating relationship between topographic surface, subsurface reservoirs and deeper basement level structures.*

Among the several internal phenomena that could trigger the evolution of the basin, of particular importance is the **buoyancy** of the lighter layers over the denser ones. For example, salt is less compressible than other materials. Thus, rock-salt may become less dense than the surrounding rocks and turn into one of the primary internal driving forces of the sedimentary basin dynamics. The sediments compressed by the overburden instead ex-

pel most of their water content and become heavier than the salt layer. Other types of rocks such as lightweight shale can be also affected by buoyancy effects.

The sediments and the rocks behave, on geological time scales, as a viscous fluid. In Table 1.1 are described the main physical characteristics of different types of sediments.

Table 1.1: *Viscosities and densities of various rocks found in sedimentary basins.*

Type of sediment or rock	Density (Kg/m^3)	Viscosity ($Pa \cdot s$)
Shale	2300	10^{21}
Under-compacted Shale	2200	10^{20}
Sandstone	2400	10^{21}
Limestone	2500	10^{22}
Rock-salt	1800	10^{19}
Rock	2500	10^{21}

The fluid behaviour of the rocks could be explained at least by two physical arguments:

1. the first one deals with the crystalline structure of the rocks and the movement of voids and dislocations.

   The rock structure contains many defects in the crystalline lattice, thus their position evolves in geological time scales and their distribution can be statistically determined.

   The movement of the voids is equally probable in all the directions without applied loads, but if a load is applied, their movement becomes more probable in some specific directions, [7] and the macroscopic net effect is a fluid behaviour associated to a Newtonian rheology.

2. The second one is based on the solubility of rock components in water. It is known that pressure affects the solubility of solids in water, so where a load is applied, the solubility increases. In the contact regions among grains the rock dissolves, and later deposits on unstressed areas. This phenomenon is represented macroscopically by a shear flow of a Newtonian fluid, see [7].

Nevertheless not all the effects can be explained by viscous fluid models. In fact also plasticity plays a key role: the rocks, under the pressure applied by the overburden, tend to fracture and to modify their reciprocal position. These effects are usually modelled with a plastic-type rheological law, see [42].

The fault formation can be seen as a plastic effect: in the faulted regions the soil is highly damaged and it cannot sustain the applied stress. All these phenomena are active in a sedimentary basin and none of them alone can fully explain the rheological behaviour of the sediments. So far, a comprehensive model that links the stress to the strain and the strain to the velocity is missing, therefore semi-empirical relations are widely used like [7], [31], [42], [44].

Another important phenomenon is **compaction**: superficial layers of sediments can have up to 50% of void space filled by air/water (sea water if the sedimentary basin is the sea bed). Indeed the pore spaces are saturated by water below few hundred meters. As the layers are progressively buried by the accumulation of sediments, the overburden pressure rises. This triggers the reduction of the pore spaces and the liquid phase is expelled from the porous media.

In some cases the water can be trapped by impermeable traps: in this case the fluid pressure rises as part of the overburden is supported by the liquid phase. This case is also known as **overpressure** and it has strong consequences on the safety of oilfield exploitation.

The chemical reactions are another key element, in fact they can modify the chemical composition of the rocks, an example is the **cementification**. Chemistry is involved also in the rock formation (the **diagenesis**): older sediment layers are made of compacted rock with much stronger mechanical characteristics than the shallow under-compacted sediments. Also the formation of natural oil and gas is affected by chemical reactions.

Another important feature to be considered is the temperature distribution inside the sedimentary basin. The thermal gradient is generally about $30^{\circ}/km$, therefore the basement could reach a temperature of three hundred degrees Celsius.

The main effect is the heat diffusion from the lower layers and from the upper mantle, but also the transport of heat carried by the water is an important aspect. Temperature greatly affects the chemical reactions and the rheology. Finally, salt is a good heat conductor and this has many implications in the petroleum formation.

1.2.1 Salt tectonics

Salt tectonics is concerned with the geometries and processes associated with the presence of significant thicknesses of evaporites containing rock salt within a stratigraphic sequence of rocks. The formation of salt domes is due both to the low density of salt, which does not increase with burial,

and its low strength.

A salt dome is a type of structural dome formed when a thick bed of evaporite minerals (mainly salt, or halite) found at depth intrudes vertically into surrounding rock strata, forming a diapir. It is important in petroleum geology because salt structures are impermeable and can lead to the formation of a stratigraphic trap.

The formation of a salt dome begins with the deposition of salt in a restricted marine basin. The restricted flow of salt-rich seawater into the basin allows evaporation to occur, resulting in the precipitation of salt, with the evaporites being deposited. The rate of sedimentation of salt is significantly larger than the rate of sedimentation of clastics [41], but it is recognised that a single evaporation event is rarely enough to produce alone the vast quantities of salt needed to form a salt diapir. This indicates that a sustained period of episodic flooding and evaporation of the basin happened, as can be seen from the example of the Mediterranean Messinian salinity crisis. At the present day, evaporite deposits can be seen accumulating in basins that have restricted access, but do not completely dry out.

Over time, the layer of salt is covered with deposited sediment, and buried under an increasingly large overburden. The overlying sediment undergo compaction, causing an increase in density and therefore a decrease in buoyancy. Unlike clastics, pressure has a significantly smaller effect on the salt density due to its crystal structure and this eventually leads to it becoming more buoyant than the sediment above it.

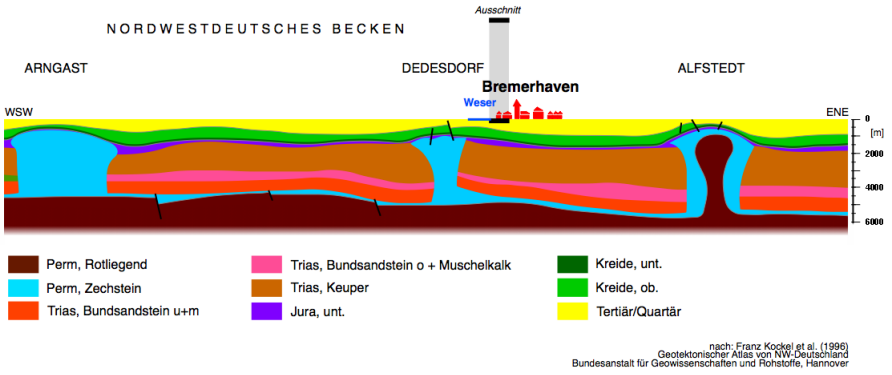

Figure 1.3: *Examples of salt domes: the geological profile through northern Germany with salt domes in blue (taken from wikipedia).*

The ductility of salt initially allows plastic deformation and lateral flow, decoupling the overlying sediment from the underlying sediment. Since the

salt has a larger buoyancy than the sediment above, if a significant faulting event affects the lower surface of the salt, it can be enough to cause the salt to begin to flow vertically, forming a salt pillow.

The vertical growth of these salt pillows creates pressure on the upper surface, causing extension and faulting. Eventually, over millions of years, the salt will pierce and break through the overlying sediment, first as a dome-shaped and then a mushroom-shaped, the fully formed *salt diapir*. If the rising salt diapir reaches the surface, it can become a flowing salt glacier like the salt glacier of Lüneburg Kalkberg in Germany. In cross section, these large domes can measure from 1 to 10 km in diameter, and extend as deep as 6.5 km.

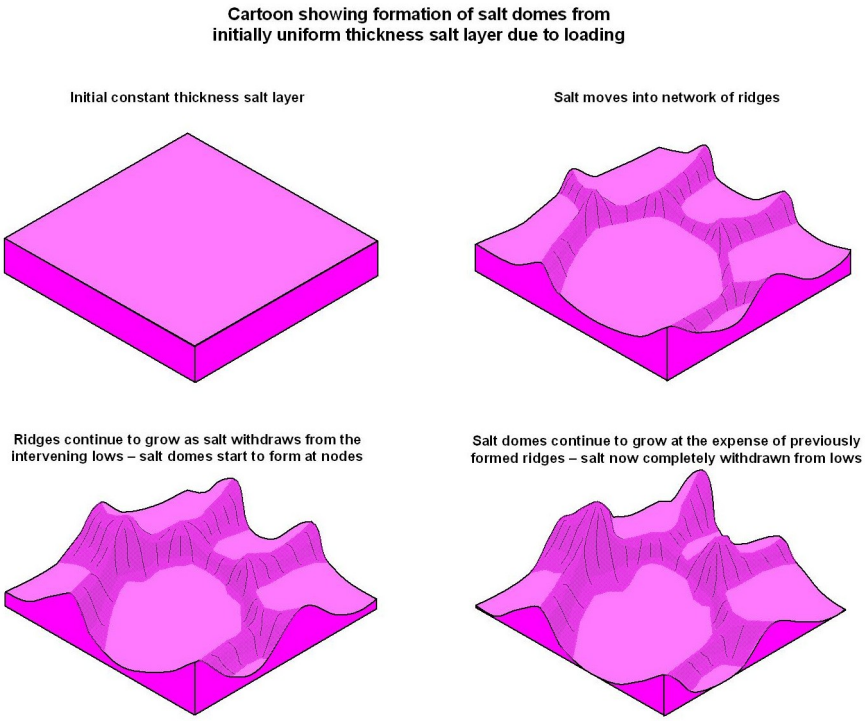

Figure 1.4: *Schematic view of the process of the growth of salt domes (taken from wikipedia).*

The rock-salt that is found in salt domes is mostly impermeable. As the salt moves up towards the surface, it can penetrate and/or bend layers of existing rock with it. As these strata are penetrated, they are generally bent slightly upwards at the point of contact with the dome, and can form pockets where petroleum and natural gas can collect between impermeable layers of rock and the salt. The strata immediately above the dome that

are not penetrated are pushed upward, creating a dome-like reservoir above the salt where petroleum can also gather. These oil pools can eventually be extracted, and indeed they form a major source of the petroleum produced along the coast of the Gulf of Mexico.

1.3 Importance of numerical simulation

Sedimentary basins, in particular salt basins, are among the best places to find petroleum, natural gas and to store nuclear waste material. In fact the low permeability of salt guarantees low water leakages that are the main concern for the safety of nuclear waste storage. Precise data regarding the basin evolution on geological timescales are required to solve the problems related to these two applications.

The history of the basin has a deep impact on the characteristics of the generated oil: in particular the geometrical evolution and the temperature experienced by the sediments determine localization, quantity and quality of the oil. For example temperature is a key aspect that controls the petroleum-gas ratio, the latter being less valuable than the former. Another example is the geometry of the cap-rock which is the sealing layer of the oilfield. Oil usually floats and collects near the cap-rock. In other terms, to have detailed information about an oilfield, we must have information about the past history of the basin.

Until now sedimentary basin studies have been based on the geological interpretation of experienced specialists. Geologists can usually outline several evolution scenarios of the basin. Therefore we must choose among them the ones which are coherent from a physical viewpoint. Numerical simulation could provide a tool for choosing the right scenario. Moreover, it can provide quantitative information (for instance the stress field) which are difficult to estimate by other means.

The great interest in numerical tools is boosted by the technical difficulties to carry out analogical experiments. Indeed it is difficult to scale correctly all the physical quantities in a relatively small model.

Sandbox experiments [12] provide useful information regarding the brittle behaviour of grains but can not represent all the viscous creeping mechanisms which require millions of years to produce a measurable effect. The experiments devoted to investigate the sediment rheology are difficult to carry out too. As a matter of fact it is necessary, at the same time, to reach extreme values of pressure and to measure very small displacements.

Also the nuclear industry is interested in structural geology simulations. Here the main interest is oriented to simulate future evolution of the basins

and, in particular, the stability of a deposit in a time frame comparable to the half life of the isotopes to be stored.

CHAPTER 2

Mathematical Problem

2.1 Introduction

In the description of the evolution of sedimentary basins and of the diapir growth the typical time scales are of the order of millions of years. At this time scale sedimentary rocks can be modelled as Newtonian fluids characterized by high viscosity. Thanks to this assumption the description of the diapir growth can be cast into the framework of the Rayleigh-Taylor theory describing the evolution of gravitational instability between fluid layers. Thus, let us consider sedimentary rocks and halite as incompressible Newtonian fluids having variable and possibly discontinuous density and viscosity, possibly discontinuous. In this case the Navier-Stokes equations read:

$$\begin{cases} \rho \frac{\partial \mathbf{u}}{\partial t} + \rho \left(\mathbf{u} \cdot \nabla \right) \mathbf{u} - \nabla \cdot \left[\mu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^{\top} \right) \right] + \nabla p = \rho \mathbf{g} \\ \nabla \cdot \mathbf{u} = 0 \end{cases} \tag{2.1}$$

in an open set $\Omega \in \mathbb{R}^d$ and $t > 0$; ρ is the density, μ the viscosity, $\mathbf{u}$ the velocity field, p the pressure and $\mathbf{g} = (0, 0, -g)^{\top}$ the gravity acceleration.

Note that the domain Ω evolves in time because of the presence of a free surface (the top layer).

2.2 Stokes equation

Let us perform a dimensional analysis of (2.1) by considering the typical values of the dynamic parameters involved in the diapir growth (see Table 1.1). According to literature data [23] we choose the following reference values for density and viscosity:

$$\overline{\mu} = 10^{20}\, Pa \cdot s\,, \quad \overline{\rho} = 10^{3}\, \frac{kg}{m^3} \tag{2.2}$$

Since the time scale is of order of $1\, Ma$, while the characteristic length of a sedimentary basin is about $1\, km$ vertically, let introduce the following scaling factors for the space and time:

$$T = 1\, Ma = 3.1536 \cdot 10^{13}\, s, \quad L = 1000\, m \tag{2.3}$$

while the pressure will be scaled by

$$\overline{P} = \rho g L = 9.81 \cdot 10^{6}\, Pa \tag{2.4}$$

Therefore, we adopt the following non-dimensional quantities:

$$\begin{gathered} \bar{t} = \frac{t}{T}, \quad \overline{x} = \frac{x}{L}, \quad \overline{p} = \frac{p}{P}, \\ \overline{\mathbf{u}} = \frac{\mathbf{u}}{U} = \mathbf{u}\frac{T}{L} \end{gathered} \tag{2.5}$$

So, time and space derivatives can be rescaled as:

$$\frac{\partial(\cdot)}{\partial t} = \frac{1}{T}\frac{\partial(\cdot)}{\partial \bar{t}} \quad \nabla(\cdot) = \frac{1}{L}\overline{\nabla}(\cdot) \quad \nabla \cdot (\cdot) = \frac{1}{L}\overline{\nabla \cdot}(\cdot) \tag{2.6}$$

where ∇ and $\nabla\cdot$ are respectively the gradient and divergence operators with respect to the set of non-dimensional variables. Then, it is possible to rewrite equations (2.1) in terms of the non-dimensional quantities (2.5) where it is omitted the $\bar{\cdot}$ from now.

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} - \frac{1}{Re}\nabla \cdot \left[\mu\left(\nabla \mathbf{u} + \nabla \mathbf{u}^{\top}\right)\right] + \frac{1}{Fr}\nabla p = \frac{1}{Fr}\mathbf{g} \\ \nabla \cdot \mathbf{u} = 0 \end{cases} \tag{2.7}$$

The Reynolds and Froude non-dimensional numbers defined respectively as come out from the previous assumption:

$$Re = \frac{\rho L^2}{\bar{\mu} T}, \quad Fr = \frac{L}{T^2 g} \tag{2.8}$$

represent respectively the ratio between the inertial and viscous forces, and the ratio between buoyancy and inertial forces.
Since in the application of our interest:

$$Re = 3.174 \cdot 10^{-25}, \quad Fr = 1.027 \cdot 10^{-25} \tag{2.9}$$

the inertial terms can be dropped from the momentum equation of the Navier-Stokes system since they are extremely low.
The evolution of a sedimentary basin can therefore be described by the Stokes problem, namely

$$\begin{cases} -\frac{Fr}{Re} \nabla \cdot \left[\mu \left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^\top \right) \right] + \nabla p = \boldsymbol{g} \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases} \tag{2.10}$$

Then considering also the equation to track the physical properties (that will be presented in section 2.3) it is possible to summarize the closed system of equations

$$\begin{cases} -\frac{Fr}{Re} \nabla \cdot \left[\mu (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^\top) \right] + \nabla p = \rho \boldsymbol{g} & \text{in } \Omega \times (0, T] \\ \nabla \cdot \boldsymbol{u} = 0 & \text{in } \Omega \times (0, T] \\ \dfrac{\partial \rho}{\partial t} + \boldsymbol{u} \cdot \nabla \rho = 0 & \text{in } \Omega \times (0, T] \\ \dfrac{\partial \mu}{\partial t} + \boldsymbol{u} \cdot \nabla \mu = 0 & \text{in } \Omega \times (0, T] \\ \rho = \rho_0, \quad \mu = \mu_0 & \text{in } \Omega \times \{0\} \\ \boldsymbol{u} = \tilde{\boldsymbol{u}} & \text{on } \Gamma \end{cases} \tag{2.11}$$

The effect of stress in the fluid is represented by the left hand side of the first equation (2.11). The term ∇p is called the *pressure gradient* and arises from the isotropic part of the Cauchy stress tensor. This part is given by normal stresses that turn up in almost all situations. The anisotropic part of the stress tensor gives rise to $\nabla \cdot \left[\mu (\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^\top) \right]$, which conventionally describes viscous forces; for incompressible flow, this is only a shear effect.

The second equation represents the incompressibility constrain while the last two equations are balance equations of respectively density ρ and viscosity μ.

2.2.1 Boundary conditions

Let us consider the model sketched in Figure 2.1, with the boundary conditions summarized in (2.12): on Γ_B , the bottom of the domain, the velocity is set to zero as the rocks are considered static, on Γ_S and Γ_L the condition for the velocity is $\boldsymbol{u} \cdot \boldsymbol{n} = 0$, so there is no velocity in the normal direction, while the upper part Γ_S behaves like a free surface.

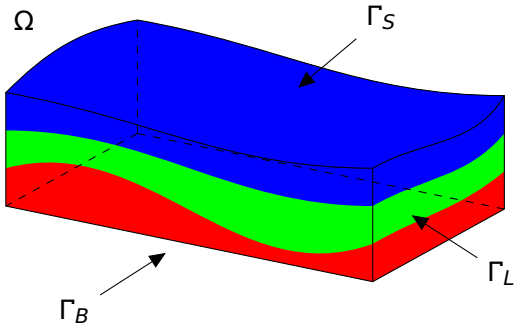

Figure 2.1: *External shape of the domain Ω. The external boundary Γ is divided into three parts: the basement Γ_B, the free surface Γ_S and the lateral contour Γ_L.*

The existence and uniqueness of the solution has been already proved in [17].

$$\begin{cases} \rho = \rho_0, \quad \mu = \mu_0 & \text{in } \Omega \times \{0\} \\ \boldsymbol{u} = \tilde{\boldsymbol{u}} & \text{on } \Gamma_B \\ \boldsymbol{u} \cdot \boldsymbol{n} = 0, \quad \boldsymbol{\sigma} \cdot \boldsymbol{n} - \boldsymbol{n}^\top \boldsymbol{\sigma} \boldsymbol{n} = 0 & \text{on } \Gamma_L \\ \boldsymbol{\sigma} \cdot \boldsymbol{n} = 0 & \text{on } \Gamma_S \\ \boldsymbol{\sigma} = \mu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^\top) - \nabla p \end{cases} \tag{2.12}$$

2.2.2 Weak problem

Finite elements formulations of the Stokes problem are based on the weak form of (2.12). Details may be found for instance in [34]. Here we recall only the basic steps. Now we multiply the momentum balance equation by suitable test function $\boldsymbol{v} \in V \subset [H^1(\Omega)]^d$ and we integrate by parts to obtain:

$$\int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} + \int_\Omega p \nabla \cdot \boldsymbol{v} = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \quad \forall \boldsymbol{v} \in V \tag{2.13}$$

Finally, we multiply the continuity equation by a test function

$$q \in L^2(\Omega) \tag{2.14}$$

to obtain:

$$\int_\Omega q \nabla \cdot \boldsymbol{u} = 0 \quad \forall q \in L^2(\Omega) \tag{2.15}$$

The final weak formulation is as follows.
Find $(\boldsymbol{u}, p) \in V_B \times L^2(\Omega)$ such that:

$$\begin{cases} \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} + \int_\Omega p \nabla \cdot \boldsymbol{v} = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} & \forall \boldsymbol{u} \in V_B \\ \int_\Omega q \nabla \cdot \boldsymbol{u} = 0 & \forall q \in L^2(\Omega) \end{cases} \tag{2.16}$$

where

$$V = \{\boldsymbol{v} \in \left[H^1(\Omega)\right]^d : \quad \boldsymbol{v} \mid_{\Gamma_B} = 0, \quad \boldsymbol{v} \cdot \boldsymbol{n} \mid_{\Gamma_S \cup \Gamma_L} = 0\} \tag{2.17}$$

and

$$V_B = \{\boldsymbol{v} \in \left[H^1(\Omega)\right]^d : \quad \boldsymbol{v} \mid_{\Gamma_B} = \tilde{\boldsymbol{u}}, \quad \boldsymbol{v} \cdot \boldsymbol{n} \mid_{\Gamma_S \cup \Gamma_L} = 0\} \tag{2.18}$$

It is clear from the previous derivation that a classical solution of the Stokes problem is also a weak solution. If we define:

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega \nabla \boldsymbol{u} : \nabla \boldsymbol{v} \tag{2.19}$$

$$b(\boldsymbol{v}, p) = \int_\Omega p \nabla \cdot \boldsymbol{v} \tag{2.20}$$

$$F(\boldsymbol{v}) = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \tag{2.21}$$

the weak problem can be cast in the form:

$$\begin{cases} a(\boldsymbol{u}, \boldsymbol{v}) + b(\boldsymbol{v}, p) = F(\boldsymbol{v}) & \forall \boldsymbol{v} \in V \\ b(\boldsymbol{u}, q) = 0 & \forall \boldsymbol{q} \in S = L^2(\Omega) \end{cases} \tag{2.22}$$

2.3 Hyperbolic tracking equation

Given the incompressibility constraint stated in the second of (2.11), the pure advection equation for the density ρ is equivalent to the mass conservation law. Following [46], we introduce a set of characteristic functions λ_α, each one associated to the sub-domain Ω_α, that identify a region with homogeneous material properties. More precisely,

$$\lambda_\alpha(\boldsymbol{x}, t) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \Omega_\alpha(t) \\ 0 & \text{if } \boldsymbol{x} \notin \Omega_\alpha(t) \end{cases} \tag{2.23}$$

where the dependency on time is kept explicit here to remark that the sub-domain Ω_α evolves in time. We denote by $\boldsymbol{\lambda}$ the vector of values λ_α where the subscript α denotes the layer that is tracked. If we consider that the physical properties vary sharply across the interfaces $\Gamma_{\alpha\beta} = \partial\Omega_\alpha \cap \partial\Omega_\beta$, we can rewrite the density ρ and the viscosity μ as

$$\rho = \sum_{\alpha=1}^{s} \lambda_\alpha \, \rho_\alpha, \qquad\qquad \mu = \sum_{\alpha=1}^{s} \lambda_\alpha \, \mu_\alpha \tag{2.24}$$

where s denotes the total number of sub domains Ω_α. We can now similarly rewrite the initial conditions for the physical properties as

$$\rho_0 = \sum_{\alpha=1}^{s} \lambda_{\alpha 0} \, \rho_\alpha, \qquad\qquad \mu_0 = \sum_{\alpha=1}^{s} \lambda_{\alpha 0} \, \mu_\alpha$$

Therefore we can drop the transport equations in (2.11) and replace them with an evolution equation for the set of λ_α

$$\begin{cases} \dfrac{\partial \boldsymbol{\lambda}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{\lambda} = 0 & \text{in } \Omega \times (0, T] \\ \boldsymbol{\lambda} = \boldsymbol{\lambda}_0 & \text{in } \Omega \times \{0\} \end{cases} \tag{2.25}$$

where $\boldsymbol{\lambda}_0 = \{\lambda_{\alpha 0}\}$. Finally, the complete model for the basin becomes

$$\begin{cases} \nabla \cdot (\mu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^\top)) - \nabla p = -\rho \boldsymbol{g} \\ \nabla \cdot \boldsymbol{u} = 0 \\ \dfrac{\partial \boldsymbol{\lambda}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{\lambda} = 0 \end{cases} \tag{2.26}$$

with suitable boundary and initial conditions, and complemented with the algebraic relations (2.24).

CHAPTER 3

Discretized Problem and Parallel Implementation

This chapter illustrates the discretized problem and the method to track interfaces between immiscible fluids when several fluids are present. In the first part are given the algorithmic details while the second part deals with the parallelization.

3.1 Tracking Algorithm

The aim is to construct an efficient and robust method, effective even when the interfaces experience a strong deformation, with good mass conservation properties, that can be used on 2D and 3D unstructured meshes. The basic methodology, which we briefly recall in the following, has been taken from [46].

3.1.1 Characteristic function discretization

To solve (2.25) we adopt a coupled level set/volume tracking technique that can deal with many fluid components. The two representations are in direct

correspondence with a two-fold interpretation of the discretized problem, as explained in the cited reference.

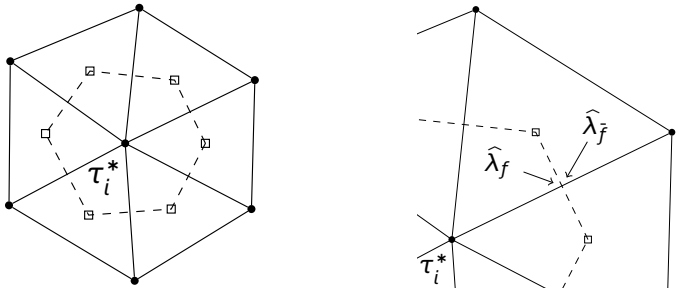

Figure 3.1: *A cell τ_i^* of the dual mesh and the position of the fluxes $\widehat{\lambda}_f$ and $\widehat{\lambda}_{\bar{f}}$.*

Let's denote with $\mathcal{T}_h^*$ the dual mesh of $\mathcal{T}_h$ and with τ_i^* a generic element of $\mathcal{T}_h^*$. A two-dimensional example of the relationship between meshes is shown in Figure 3.1. Mesh $\mathcal{T}_h^*$ has a number of cells equal to the number of points of $\mathcal{T}_h$. Let

$$\boldsymbol{\lambda}_h^n = [\lambda_{h,1}^n, \dots \lambda_{h,s}^n]^T$$

be the vector containing the discrete solution of (2.25) at time step t^n, where s is the number of components. We take each $\lambda_{h,\alpha}^n$, $\alpha = 1, \dots s$, in the space V_0^* defined as

$$V_0^* = \{\phi_h \in L^2(\Omega) : \phi_h|_{\tau_i^*} \in \mathbb{P}^0(\tau_i^*)\}, \tag{3.1}$$

where $\mathbb{P}^0(\tau_i^*)$ is the space of constant polynomials in τ_i^*. If χ_i is the characteristic function of τ_i^* we can expand $\boldsymbol{\lambda}_h^n$ as

$$\boldsymbol{\lambda}_h^n = \sum_{i=1}^{n_p} \boldsymbol{\lambda}_i^n \chi_i,$$

where $\boldsymbol{\lambda}_i^n \in \mathbb{R}$ is the value of $\boldsymbol{\lambda}_h^n$ on τ_i^*.
Let

$$V_1 = \{\psi_h \in \mathbb{C}^1(\Omega) : \psi_h|_{\tau_k} \in \mathbb{P}^1(\tau_k)\}$$

be the space of piecewise linear functions on τ_h and $\Pi_h : V_0^* \to V_1$ be the interpolant such that $\psi = \Pi_h^n \lambda$ satisfies

$$\psi(x_i) = \lambda(x_i), \quad i = 1, \dots n_p.$$

From now on, the subscript h, that identifies the discrete solution, will be discarded to ease notation and we set the level set function $\boldsymbol{\psi}^n$ as $\boldsymbol{\psi}^n =$

$\Pi_h \boldsymbol{\lambda}^n$. This way we link the volume tracking and the level set representations together.
We define a finite volume advection scheme for $\boldsymbol{\lambda}$ of the form

$$\boldsymbol{\lambda}_i^{n+1} = \left(1 + \sum_{j \in \mathcal{C}_i} \nu_{i,j}^n\right) \boldsymbol{\lambda}_i^n - \sum_{j \in \mathcal{C}_i} \boldsymbol{F}_{i,j}^n.$$

Here, $\mathcal{C}_i$ is the set of elements of $\mathcal{T}_h^*$ adjacent to τ_i^*, where $\nu_{k,j}^n$ are the interface Courant numbers defined as

$$\nu_{i,j}^n = \frac{\Delta t^n}{|\tau_i^*|} \int_{l_{ij}^*} \boldsymbol{u}^n \cdot \boldsymbol{n},$$

where $l_{ij}^* = \partial\tau_i^* \cap \partial\tau_j^*$ and $\boldsymbol{n}$ the normal to $\partial\tau_i^*$.

The interface fluxes $\boldsymbol{F}_{i,j}^n$ are defined as $\boldsymbol{F}_{i,j}^n = \nu_{i,j}^n \Phi_{ij}(\boldsymbol{\lambda}_{i,j}^{n,+}, \boldsymbol{\lambda}_{i,j}^{n,-})$ where

$$\Phi_{ij}(\boldsymbol{\lambda}^+, \boldsymbol{\lambda}^-) = \begin{cases} \boldsymbol{\lambda}^+ & \text{if } \nu_{i,j}^n \geq 0 \\ \boldsymbol{\lambda}^- & \text{if } \nu_{i,j}^n < 0 \end{cases}$$

is the upwind function. Here, $\boldsymbol{\lambda}_{i,j}^{n,\pm}$ are suitably reconstructed states on the two sides of l_{ij}^*: that facing τ_i^* and that facing τ_j^*, respectively. The reconstruction is carried out by a particular MUSCL-type algorithm, detailed in [46] and based on the minimization of a local constrained problem that guarantees mass conservation as well as the positivity of the solution. This reconstruction requires the knowledge of the value of the level set function ψ in a neighbourhood of each element of the dual grid.

3.1.2 Time discretization

To solve (2.26) numerically we proceed with a time discretization of the time interval $[0, T]$ in discrete times $0 = t^0, t^1, \ldots$ with variable time step size $\Delta t^n = t^{n+1} - t^n$. We adopt a first order accurate splitting algorithm [32] in order to segregate the evolution of the fluid variables $\boldsymbol{u}$ and p from the evolution of the characteristic functions λ_α. We denote the quantities at t^n with the superscript n. At each time step $n+1$ we therefore adopt the following scheme:

i) we advance the characteristic vector function $\boldsymbol{\lambda}$ using $\boldsymbol{u}$ at time step n in an explicit manner

$$\frac{\boldsymbol{\lambda}^{n+1} - \boldsymbol{\lambda}^n}{\Delta t^n} + \boldsymbol{u}^n \cdot \nabla \lambda^n = 0 \tag{3.2}$$

ii) we use $\boldsymbol{\lambda}^{n+1}$ to compute ρ^{n+1} and μ^{n+1},

iii) we solve the Stokes system

$$\begin{cases} \nabla \cdot (\mu^{n+1}(\nabla \boldsymbol{u}^{n+1} + \nabla \boldsymbol{u}^{n+1^\top})) - \nabla p^{n+1} = -\rho^{n+1}\boldsymbol{g} \\ \nabla \cdot \boldsymbol{u}^{n+1} = 0 \end{cases} \tag{3.3}$$

to obtain the new velocity and pressure fields $\boldsymbol{u}^{n+1}$ and p^{n+1}.

The time derivatives are discretized with a first order scheme since the splitting already reduces the order of convergence of the global algorithm to first-order in time. While the Stokes system is solved implicitly, so there is no restriction on the time step, the set of equations for $\boldsymbol{\lambda}$ is discretized explicitly, leading to the well known Courant condition [25]. For this reason a single time step for the Stokes system is usually matched by multiple sub-steps for the hyperbolic solver.

3.2 Spatial discretization of the Stokes problem

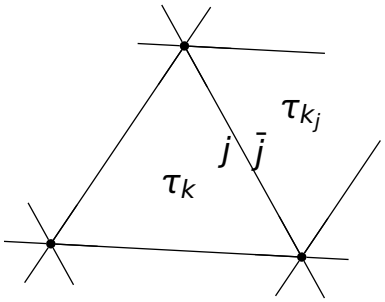

Figure 3.2: *An element τ_k of the mesh and a facing element τ_{k_j}.*

Problem (2.16) can be discretized with a suitable finite element approach [11]. Let $\mathcal{T}_h$ be a simplicial tetrahedral grid on Ω with n_e elements and n_p points, and let h denote the maximum diameter of the grid elements. Let also τ_k be a generic element of $\mathcal{T}_h$, i.e. $\bigcup_k \tau_k = \mathcal{T}_h$. If f_k is the number of faces of τ_k, we denote by τ_{k_j} the element that faces τ_k on the j-th face, while $\bar{j}$ is the index of the face of τ_{k_j} that faces τ_k, as depicted in Figure 3.2 for the two-dimensional case.

We denote with $\boldsymbol{u}_h^n$ and p_h^n the discrete variables for the velocity and the pressure fields at the n-th time step, respectively. The spaces that contain

these functions are respectively $X_h \subset [H^1(\Omega)]^d$ and $S_h \subset L^2_0(\Omega)$, that are two sets of finite dimensional subspaces parametrized by h. To satisfy the Ladyzenskaja-Babuska-Brezzi condition we choose X_h to be a subspace formed by first order polynomials with an additional cubic bubble function at the barycentre of the element (denoted by $\mathbb{P}^1_b$), while S_h is the subspace of first order polynomials $\mathbb{P}^1$, so

$$\begin{aligned} \boldsymbol{u}_h^n \in X_h, \quad X_h &= \{\boldsymbol{v}_h \in \left[H^1(\Omega)\right]^d : \boldsymbol{v}_h|_{\tau_k} \in \mathbb{P}^1_b\} \\ p_h^n \in S_h, \quad S_h &= \{q_h \in L^2_0(\Omega) : q_h|_{\tau_k} \in \mathbb{P}^1\} \end{aligned}$$

Albeit Stokes problem is stationary, we need to treat the time for the hyperbolic tracking equation (see section 2.3). It is now possible to write (2.10) in its discretized weak form, dropping the superscript $n+1$ from all variables for simplicity, as

$$\begin{cases} a(\boldsymbol{u}_h, \boldsymbol{v}_h) + b(p_h, \boldsymbol{v}_h) = f(\boldsymbol{v}_h) & \forall \boldsymbol{v}_h \in X_h \\ b(q_h, \boldsymbol{u}_h) = 0 & \forall q_h \in S_h \end{cases} \tag{3.4}$$

We expand the discrete solutions $\boldsymbol{u}_h$ and p_h on the test functions v_h and q_h, as

$$\boldsymbol{u}_h = \sum_{i=1}^{n_p+n_e} \boldsymbol{u}_i v_i \qquad\qquad p_h = \sum_{i=1}^{n_p} p_i q_i$$

so we can finally write (2.10) in the algebraic form that reads

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^\top \\ \boldsymbol{B} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{V} \\ \boldsymbol{P} \end{pmatrix} = \begin{pmatrix} \boldsymbol{F} \\ \boldsymbol{0} \end{pmatrix} \tag{3.5}$$

where $\boldsymbol{V} = \{u_i\}$, $\boldsymbol{P} = \{p_i\}$ and

$$\boldsymbol{A} = \{a(v_i, v_j)\} \qquad \boldsymbol{B} = \{b(q_i, v_j)\} \qquad \boldsymbol{F} = \{f(v_i)\}$$

Further details on the discretization can be found in [46].

The size of these matrices can be quite large when dealing with realistic cases, where unstructured meshes with hundred of thousands up to million of points are used. Sparse matrix memorization and iterative solvers are therefore a necessity. The conditioning number of the matrix is greatly influenced by the jumps in the values of the viscosity. Suitable preconditioning techniques must be adopted to reach convergence with an acceptable number of iterations and in a reasonable time.

The parallelization of the solutions of the discretized Stokes problem is tackled via a parallel preconditioned Krylov solver, whose details are given in the next chapters.

3.3 Parallel Implementation

Our code is implemented in the project called **LifeV** which is a parallel Finite Elment (FE) library "providing implementations of state of the art mathematical and numerical methods. It serves both as a research and production library. It has been used already in medical and industrial context to simulate fluid structure interaction and mass transport. LifeV is the joint collaboration between four institutions: École Polytechnique Fédérale de Lausanne (CMCS) in Switzerland, Politecnico di Milano (MOX) in Italy, INRIA (REO, ESTIME) in France and Emory University (Sc. Comp) in the U.S.A.".

LifeV is based on a parallel framework called **Trilinos** (for further information see [21]). The Trilinos framework "uses a two level software structure that connects a system of packages. A Trilinos package is an integral unit, usually developed to solve a specific task, by a (relatively) small group of experts. Packages exist beneath the Trilinos top level, which provides a common look-and-feel. Each package has its own structure, documentation and set of examples, and it is possibly available independently of Trilinos".

We will describe the following subset of the Trilinos packages used.

- **Epetra**. The package defines the basic classes for distributed matrices and vectors, linear operators and linear problems. Epetra classes are the common interface used by all the Trilinos packages. Each Trilinos package accepts as input Epetra objects and also LifeV is based on Epetra objects that can be serial or distributed transparently on several processors.

- **AztecOO**. This is a linear solver package based on preconditioned Krylov methods. AztecOO also supports all the Aztec interfaces and functionality, and also provides significant new functionality. It has the main disadvantage to precondition only on the right side and to support only the single precision.

- **Belos**. It provides next-generation iterative linear solvers and a powerful linear solver developer framework. Compared to AztecOO, it brings more preconditioned Krylov methods like MINRES and Recycling methods. Additionally it can apply preconditioners also on the left side, it supports double precision and last but not least it is thread safe.

- **IFPACK**. The package performs various incomplete factorizations, and is used with AztecOO.

- **Teuchos**. This is a collection of classes that can be essential like xml parser to retrieve the Belos parameter.

- **ML**. It is an algebraic multilevel and domain decomposition preconditioner package that provides scalable preconditioning capabilities for a variety of problems. It is used as a preconditioner for AztecOO/Belos solvers and it can also use IFPACK and Amesos as smoothers.

- **Amesos**. The package provides a common interface to certain sparse direct linear solvers (generally available outside the Trilinos framework), both sequential and parallel like SuperLU, SuperLU_dist and UMFPACK.

- **Zoltan**. A tool-kit of parallel services for dynamic, unstructured, and/or adaptive simulations. Zoltan provides parallel dynamic load balancing and related services for a wide variety of applications, including finite element methods, matrix operations, particle methods, and crash simulations.

The Trilinos framework hides every MPI call to make possible to re-use the same code in serial and in parallel. The Epetra library take care of the communication between the various processors, so the user do not need strictly to have some familiarity with distributed memory computing.

When we want to create an Epetra distributed object (both vectors and matrices), we first need to create a communicator that could be the MPI communicator or a "serial" one.
Then it is possible to create an `Epetra_Map` that is a distribution of a set of integer labels (or elements) across the processes. Finally it is possible to create a distributed vector based on this map.

For off-processor communications we can use the `Epetra_Import` and `Epetra_Export` classes, which are used to construct a communication plan that can be called repeatedly by computational classes such as `Epetra_Vector` and Epetra Matrices.

Epetra provides an extensive set of classes to create and fill distributed sparse matrices. These classes allow row-by-row or element-by-element constructions. Support is provided for common matrix operations, including scaling, norm, matrix-vector multiplication. Using Epetra objects, applications do not need to know about the particular storage format and other implementation details, such as the data layout, the number and location of ghost nodes. Epetra furnishes two basic formats, one suited for point matrices, the other for block matrices.

As a general rule, the process of constructing a (distributed) sparse matrix is as follows:

- to allocate an integer array, whose length equals the number of local rows;
- to loop over the local rows, and estimate the number of non-zero elements of each row;
- to create the sparse matrix;
- to fill the sparse matrix.

In our project LifeV we use the `Epetra_FECrsMatrix` class which defines a Compressed Row Storage matrix with the capability to set non-local matrix elements. Like the distributed vectors, the matrices that we use are created with a map:

```
Epetra_FECrsMatrix A(Map);
```

we fill the matrix with the function `SumIntoGlobalValues` that adds the coefficients specified to the matrix, adding them to any coefficient that may exist at the specified location.

In a finite element code, the user often insert more than one coefficient at a time (typically, all the matrix entries corresponding to an elemental matrix). Next, we need to exchange data, so that each matrix element not owned by process 0 could be send to the owner, as specified by `Map`. This is accomplished by calling, on all processes:

```
A.GlobalAssemble();
```

`GlobalAssemble` gathers any overlapping/shared data into the non-overlapping partitioning defined by the `Map` that was passed to this matrix at construction time. Data imported from other processors is stored on the owning processor with a "sumInto" or accumulate operation. This is a **collective** method – every processor must enter it before anyone will complete it. `GlobalAssemble()` calls `FillComplete()` that signals that data entry is complete. It performs transformations to local index space to allow optimal matrix operations, see [40].

3.3.1 Overlapping maps

A typical approach adopted for parallelizing the solution of a PDE involves the decomposition of the domain into sub-domains [34], so that each process operates on a smaller part of the domain. This implies that, when updating a value on the interface that separates these sub-domains, not all the

necessary information may be available on the given processor and communication is needed.

The case of hyperbolic equations such as the one just described presents some further difficulties. In particular the update of $\boldsymbol{\lambda}_i^n$ requires the knowledge of the level set function $\boldsymbol{\psi}^n$ in all the nodes surrounding the node i. This implies that we need to recover values of $\boldsymbol{\lambda}^n$ in a layer of elements around the given one.

Standard parallel mesh decomposition paradigms do not consider this type of communication [26].

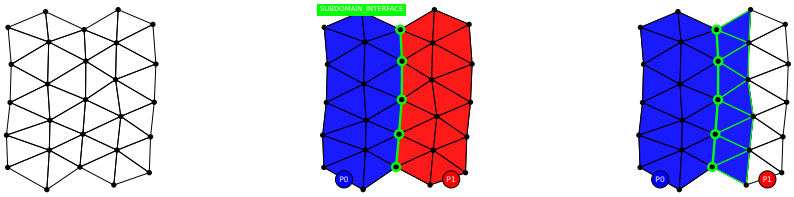

Figure 3.3: *On the left, a sample 2D mesh. On the center, the mesh is distributed on 2 processes, with the green nodes that are shared between the two. On the right, the extended sub-domains associated to the process 0 that contains the complete support to all the nodes that were assigned to that process.*

Thus, we have implemented a specialized parallel data structure that guarantees the access to all the needed data. This has been obtained by creating overlapping sub-domains together with the framework for the necessary parallel communications.

The construction of this data structure is based on the mesh connectivity and it has been made sufficiently general so that it is possible to create an overlapped point-based map or an overlapped element-based map, with an arbitrary number of layers.

The algorithm is summarized in the following:

- to build the connectivity maps that enable to search neighbourhood information;
- partition the mesh;
- identify sub-domain interface entities;
- for each level of overlap:
  - add all neighbours to the current partition;
  - update list of sub-domain interface entities;

Here "entity" may denote either a node or an element of the mesh depending on which type of communication map we wish to build:

An illustration of the this algorithm is given in Figure 3.3, where a two-dimensional domain is split into two sub-domains.

This technique allows to build the framework to access in a parallel setting the n-th neighbour of any mesh entity.

3.3.2 Stokes Solver Algorithm

The Stokes solver is totally transparent to the parallelism, in fact the code does not seem parallel except for few routines like the mesh partitioning and the Epetra routine `GlobalAssemble` (see above).
The algorithm is summarized in the following:

- read the datafile;
- read the mesh and partition it using ParMetis [26];
- read density and viscosity vectors from file or set them from the data in the local mesh;
- create the finite elements;
- create the map for pressure and velocity fields;
- create the Stokes matrix, the preconditioner matrix, the right hand side vector (rhs) and the solution vector;
- fill in the Stokes matrix, the preconditioner matrix and the rhs;
- close the matrices and the rhs (with globalAssemble);
- apply the boundary conditions;
- pass the Stokes matrix, the preconditioner matrix, the right hand side vector (rhs) and the solution vector to Belos/AztecOO and solve the system;
- write the HDF5 binary file of the solution.

During the operation of matrix filling the divergence, gradient pressure and stiffness terms are assembled into the Stokes matrix, while the pressure mass and stiffness terms are assembled into the preconditioner. Notice that this operation is **local** except the all to all communication to close the matrices at the end, performed by `GlobalAssemble` .
So, summarizing the properties of the algorithm:

- scalability: it is highly scalable since it requires few MPI calls, it exploits data locality and it does not requires much memory;
- load balancing: it is highly balanced because every processor has almost the same workload depending on the partitioning;
- amount of communication: it requires communication only to read and partition the mesh, to close the matrices and to write the output file;
- locality: every core works locally exploiting data locality;
- communication masking: the communications are partially masked by Trilinos, but since they are not so many it is not a problem.

For further information on the implementation details see the paragraph 7.2.2.

3.3.3 Tracking Algorithm

The tracking algorithm is loosely similar to the Stokes one, except of the temporal loop and of the overlapped maps to track the physical quantities like viscosity and density (both in red).
The algorithm is summarized in the following:

- read the datafile;
- read the mesh and partition it using ParMetis [26];
- create the overlap map;
- create the finite elements;
- read density and viscosity vectors from file or set them from the data in the local mesh;
- set the the velocity from file or from the Stokes solver output;
- temporal loop; repeat until reached the desired end time;
  - compute the volumes of the Voronoi cells;
  - compute the active structure of the mesh, the entities that are crossed by different composition;
  - compute the volumetric fluxes on the the active entities;
  - compute the mean value of the potential on the the active entities;

  - compute the maximum time step allowed by CFL condition;
  - compute the new interfaces only on the active entities;
  - move compositions on the active entities;
  - update the old composition;
  - update time;
  - write the HDF5 binary file of the solution.
- end simulation when reached the end time.

Let's summarizing the properties of the algorithm:

- scalability: it is highly scalable since it requires few MPI calls, it exploit data locality and it does not requires much memory;
- load balancing: it is highly balanced because every processor has almost the same workload depending on the partitioning;
- amount of communication: it requires more communication than Stokes solver, but still not much because the algorithm works only on the active regions of the domain;
- locality: every core works locally exploiting data locality like in the Stokes case;
- communication masking: the communications are only partially masked by Trilinos, we did not work on hiding them.

For further information on the implementation details see the paragraph 7.3.

3.4 The complete algorithm

In graph 3.4 we summarize each time step of the entire iterative algorithm: we reconstruct the physical quantities from the computed data λ_h^n (only λ_h^0 is given as initial condition), then we solve the Stokes problem. Since the phases are immiscible, they all move following the Stokes flow, so we can finally track the new quantities λ_h^{n+1} given u_h^n.

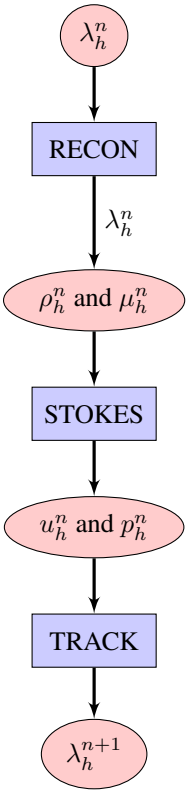

Figure 3.4: *The algorithm that summarize the entire process: first, the physical quantities are reconstructed from the data given, then they are used to solve the Stokes problem and then the velocity and pressure fields of its output are used to track again the the physical quantities.*

CHAPTER 4

Saddle Point Problems

This chapter provides a concise overview of iterative approaches for the solution of saddle point problems. Efficient iterative schemes are of particular importance in the context of large scale computations. In the the last part we introduce a suitable preconditioner for Stokes problems with varying viscosity and we will discuss various options considering also their parallel scalability.

4.1 Saddle Point Problem

Saddle point problems arise frequently in many applications in science and engineering, including mixed formulations of partial differential equations, circuit analysis, constrained optimization and many more. Indeed, constrained problems formulated with Lagrangian multipliers give rise to saddle point systems.

Here we introduce some of the most effective preconditioning techniques for Krylov subspace solvers applied to saddle point problems, including block and constraint preconditioners.

Indeed problems arising from the discretization of saddle point problems result in finite-dimensional system of equations of large size. These prob-

lems may be expressed as a sequence of quadratic minimization problems subject to linear equality constraints like,

$$\begin{cases} min\, \boldsymbol{J}(u) = \frac{1}{2} u^\top \boldsymbol{A} u - f^\top u \\ \text{subject to } \boldsymbol{B} u = g \end{cases} \tag{4.1}$$

Here $\boldsymbol{A} \in \mathrm{R}^{n \times n}$ is symmetric and positive semi-definite, and $\boldsymbol{B} \in \mathrm{R}^{m \times n}$ with $m < n$, $f \in \mathrm{R}^n$ and $g \in \mathrm{R}^m$. The conditions for the first order optimality are expressed by the linear system:

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^\top \\ \boldsymbol{B} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{p} \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{4.2}$$

In (4.2), $\boldsymbol{p} \in \mathrm{R}^m$ is a vector of Lagrange multipliers.

More generally, we consider linear systems of the form:

$$\mathcal{A}x = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^\top \\ \boldsymbol{B} & -\boldsymbol{C} \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} = b \tag{4.3}$$

where $\boldsymbol{A}$ and $\boldsymbol{B}$ are defined as above and $\boldsymbol{C} \in \mathrm{R}^{m \times m}$ is symmetric and positive semi-definite. Systems of the form (4.3) with a non-zero $\boldsymbol{C}$ block arise, for instance, in mixed finite elements approximation of incompressible flow problems, when some form of pressure stabilization is included in the discretization, and in the modelling of slightly compressible materials in linear elasticity theory.

Typically, $\mathcal{A}$ is large and sparse and (4.3) must be solved iteratively, usually by means of Krylov subspace algorithms, but unfortunately, Krylov methods tend to converge very slowly when applied to saddle point systems [2, 10], and good preconditioners are mandatory to achieve rapid convergence.

4.1.1 Properties of saddle point systems

If $\boldsymbol{A}$ is non-singular, the saddle point matrix $\mathcal{A}$ admits the following block triangular factorization:

$$\mathcal{A} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^\top \\ \boldsymbol{B} & -\boldsymbol{C} \end{pmatrix} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{B}\boldsymbol{A}^{-1} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{A}^{-1}\boldsymbol{B}^\top \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \tag{4.4}$$

where $S = -\left(\boldsymbol{C} + \boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{B}^\top\right)$ is the *Schur complement* of $\boldsymbol{A}$ in $\mathcal{A}$. Several important properties of the saddle point matrix $\mathcal{A}$ can be derived from (4.4). To begin with, it is clear that $\mathcal{A}$ is non-singular if and only

if $\boldsymbol{S}$ is. Moreover, since (4.4) defines a congruence transformation, $\mathcal{A}$ is indefinite with n positive and m negative eigenvalues if $\boldsymbol{A}$ is symmetric positive definite (SPD).

There are some important applications in which $\boldsymbol{A}$ is symmetric positive semi-definite and singular, in which case a block factorization of the form (4.4) is not possible. If $\boldsymbol{C}$ is null and $\boldsymbol{B}$ has full rank, then $\mathcal{A}$ is invertible if and only if the null spaces of $\boldsymbol{A}$ and $\boldsymbol{B}$ satisfy $\mathcal{N}(\boldsymbol{A}) \cap \mathcal{N}(\boldsymbol{B}) = \{0\}$. In this case $\mathcal{A}$ is, again, indefinite with n positive and m negative eigenvalues. For instance this is the case of the matrix of the Stokes problem.

In some important applications $\boldsymbol{A}$ is SPD and $\boldsymbol{B}$ is rank deficient and the linear system (4.3) is singular but consistent. Generally speaking, the singularity of $\boldsymbol{A}$ does not affect the convergence of the iterative solver [10]. The simple stratagem of changing the sign of the last m equations in (4.3) leads to a linear system with completely different spectral properties. Indeed, assuming that $\boldsymbol{A}$ is SPD and $\boldsymbol{C}$ is symmetric positive semi-definite, it is easy to see that the non-symmetric coefficient matrix (4.5) is positive definite, in the sense that its spectrum is contained in the right half-plane $\mathcal{R}(z) > 0$.

$$\mathcal{A} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^\top \\ -\boldsymbol{B} & \boldsymbol{C} \end{pmatrix} \tag{4.5}$$

Hence, $-\mathcal{A}$ is a stable matrix. Furthermore, under some reasonable additional conditions on $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$, it can be shown that $\mathcal{A}$ is diagonalizable and has all real and positive eigenvalues. Regardless of the formulation of the saddle point system (symmetric indefinite or non-symmetric positive definite), the convergence of Krylov subspace methods is almost always **extremely slow** unless a good preconditioner is available.

4.2 Preconditioning

Definition 4.2.1. *Preconditioning: it is an application of a transformation, called the preconditioner, that conditions a given problem into a form that is more suitable for numerical solution.*

The efficient solution of large sparse linear systems

$$\mathcal{A}x = b \tag{4.6}$$

is very important in many numerical simulations both in science and in engineering since it is often the most time-consuming part of a computation.

Direct methods, which are based on the factorization of the coefficient matrix $\mathcal{A}$ into more easily invertible matrices, are often used in many industrial codes, in particular where reliability is the primary concern. In fact, they are very robust, and tend to require a predictable amount of resources in terms of memory and time [8, 16].

Unfortunately, direct methods scale badly with problem size in terms of memory requirements and operation counts, especially on problems arising from the discretization of PDEs in three space dimensions. Indeed detailed, three-dimensional multi-physics simulations lead to linear systems of millions or even billions of equations with as many unknowns.

Then for these problems, iterative methods are the only available option. While iterative methods require less memory and often require fewer operations than direct methods (especially when an approximate solution of relatively low accuracy is sought), they are not as reliable as direct methods [2, 3]. In some applications, iterative methods fail and preconditioning is mandatory try to obtain convergence in a reasonable time.

The traditional classification of solution methods as being direct or iterative is an oversimplification and is not a satisfactory description of the present state of art. The difference between the two classes of methods is nowadays blurred, with ideas and techniques from sparse direct solvers that have been transferred (in the form of preconditioners) to the iterative framework, with the result that iterative methods are becoming more and more reliable.

While direct solvers are always based on some version of Gaussian elimination, the field of iterative methods includes a wide variety of techniques, ranging from the classical Gauss-Seidel, Jacobi and SOR iterations to multilevel methods and Krylov subspace methods. Hence it is somewhat misleading to gather all these techniques under a single heading, in particular when one considers also preconditioners.

In the context of linear system the term preconditioning refers to transforming (4.6) into another system with better properties for its iterative solution. A preconditioner is usually in the form of a matrix that applies such transformation. Preconditioning tries to improve the spectral properties of the coefficient matrix. For instance the rate of convergence of the conjugate gradient method depends on the distribution of the eigenvalues of $\mathcal{A}$ in case of symmetric positive definite (SPD) problems. Hence the optimal transformed matrix should have a smaller spectral condition number (i.e. ratio between maximal and minimal eigenvalue) and possibly eigenvalues clustered around 1.

For non-symmetric problems the situation is more complicated and the eigenvalues alone cannot describe the convergence of the non-symmetric matrix iterations like GMRES (Generalized Minimal RESidual method) [18]. In practice $\mathcal{M}$ is a non-singular matrix that approximates $\mathcal{A}$, then the linear system

$$\mathcal{M}^{-1}\mathcal{A}x = \mathcal{M}^{-1}b \tag{4.7}$$

has the same solution as (4.6), but it could be easier to solve. Here $\mathcal{M}$ is called *preconditioner*.

System (4.7) is preconditioned from the left, but it is also possible to precondition from the right

$$\mathcal{A}\mathcal{M}^{-1}y = b, \quad x = \mathcal{M}^{-1}y \tag{4.8}$$

Of course one never computes the preconditioned matrices $\mathcal{M}^{-1}\mathcal{A}$ or $\mathcal{A}\mathcal{M}^{-1}$ explicitly because this can be too expensive, and lead to the loss of sparsity. On the contrary, matrix-vector products and solutions of linear systems $\mathcal{M}z = r$ are performed. Split preconditioning is also possible

$$\mathcal{M}_1^{-1}\mathcal{A}\mathcal{M}_2^{-1}y = \mathcal{M}_1^{-1}b, \quad x = \mathcal{M}_2^{-1}y \tag{4.9}$$

where the preconditioner can be expressed as a product $\mathcal{M} = \mathcal{M}_1\mathcal{M}_2$.

Notice that matrices $\mathcal{M}^{-1}\mathcal{A}$, $\mathcal{A}\mathcal{M}^{-1}$ and $\mathcal{M}_1^{-1}\mathcal{A}\mathcal{M}_2^{-1}$ have the same eigenvalues.

In general, a good preconditioner $\mathcal{M}$ should fulfil the following requirements:

- The preconditioner should be cheap to construct and to apply in terms of storage requirements and computational power.
- The preconditioning system $\mathcal{M}z = r$ should be easy to solve.
- The solution of the preconditioned system shown take less time than the solution of the original system. Moreover, the number of iterations

should be bounded independently from the grid size and, for parallel preconditioners, from the number of processes used.

The first two requirements are in competition each other, then it is necessary to find a balance between the two needs.

An acceptable cost for the construction of the preconditioner, also said *setup time*, will typically depend on whether the preconditioner can be reused or not. In the general situation it could be worthwhile spending some time computing a powerful preconditioner when there is a sequence of linear systems with different right-hand sides and with the same coefficient matrix to be solved, because the setup time can be amortized over repeated solutions. For instance this is the case when solving slightly nonlinear problems by a variant of Newton's method or evolution problems by implicit time discretization.

There are two approaches to constructing preconditioners. One approach consists in designing specialized algorithms that are optimal for a small class of problems. This application specific approach could be very successful, but it requires complete knowledge of the problem, including for example the boundary conditions, the original equations, details of the discretization, the domain of integration. This approach includes preconditioners based on "nearby" PDEs which are easier to solve than the given one. Multi-grid preconditioners are usually of this kind [33].

The problem-specific approach could not always be feasible. The developer may not have complete knowledge of the problem to be solved, or the information might be too difficult to obtain or to use. Moreover, problem-specific approaches are often very sensitive to the problem details, and even small changes in the problem can compromise the efficiency of the solver. For these reasons, there is a need for preconditioning techniques that can be more universally applicable. Consequently, reason there is an enduring interest in general-purpose, purely algebraic methods that use only information contained in the coefficient matrix $\mathcal{A}$.

These techniques, albeit not optimal for any particular problem, could achieve a reasonable efficiency on a wide range of problems. Algebraic methods are easier to develop, use and maintain and are particularly well suited for irregular problems such as those arising from discretizations with unstructured meshes or problems from mesh-free applications.

Also, general-purpose solvers codes can be easily re-targeted and adapted as the underlying application changes. Last but not least, algebraic methods can be fine-tuned to exploit specific features of the given problem.

4.3 Preconditioned Krylov subspace methods

An important class of iterative methods, exploit the property of Krylov subspace [38].

The well known Conjugate Gradient method is widely used for the iterative solution of symmetric definite matrix systems. The main iterative approaches for indefinite symmetric matrix systems are instead the MINRES (Minimum RESidual method) and SYMMLQ (Symmetric LQ method) algorithms [6] which are based on the Lanczos procedure [22, 28]. These algorithms require any preconditioner to be symmetric and positive definite.

An alternative is the Symmetric QMR (SQMR) method [36], which allows to use symmetric and indefinite preconditioning (but it has less clear theoretical convergence properties).

As concerns computational cost, the matrix times vector product may be efficiently computed for sparse or structured matrices. Hence the main issue concerning the overall computational work in the iterative solution of a linear system with these methods is the number of iterations it takes for convergence to the desired accuracy.

Methods which guarantee some monotonic and relevant error reduction at each iteration are favoured in a number of situations: the MINRES method has such a property and thus it is sometimes the preferred choice [10]. However the SYMMLQ method has a related 'Petrov-Galerkin' property and is favoured for reasons of numerical stability when many iterations are required [14].

For a generic linear system

$$\mathcal{A}x = b \tag{4.10}$$

where $\mathcal{A}$ is symmetric (and either indefinite or definite), the MINRES method computes a sequence of iterates $\langle x_k \rangle$ for which the residual $r_k = b - \mathcal{A}x_k$ minimizes $\|r_k\|$ over the subspace.

$$r_0 + span\left(\mathcal{A}r_0, \cdots, \mathcal{A}^k r_0\right) \tag{4.11}$$

The iterates themselves belong to the Krylov subspace where x_0 is the initial iterate (the initial 'guess') and r_0 the corresponding residual,

$$x_0 + \mathcal{K}_k(\mathcal{A}, r_0) = x_0 + span\left(r_0, \mathcal{A}r_0, \cdots, \mathcal{A}^{k-1} r_0\right) \tag{4.12}$$

This minimization property leads to a description of the convergence properties of the MINRES method: since any vector s, in the space (4.11)

can be written as $s = q(\mathcal{A})r_0$ where q is a polynomial of degree k with constant term equal to one we obtain the following inequality

$$r_k \leqslant \|q(\mathcal{A})r_0\| \leqslant \|q(\mathcal{A})\|\|r_0\| \tag{4.13}$$

Now the diagonalization of the symmetric matrix $\mathcal{A}$ as $\mathcal{A} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^\top$, where $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues and the matrix $\boldsymbol{X}$ is the orthogonal matrix of eigenvectors, implies

$$\|q(\mathcal{A})\| = \|\boldsymbol{X}q(\Lambda)\boldsymbol{X}^\top\| = \|q(\Lambda)\| \tag{4.14}$$

because the Euclidean norm is invariant under orthogonal transformations. Moreover $q(\Lambda)$ is a diagonal matrix and then we obtain

$$\|r_k\| \leqslant \min_{q\in\Pi_k, q(0)=1} \max_{z\in\sigma(\mathcal{A})} \|q(z)\|\|r_0\| \tag{4.15}$$

where Π_k is the set of (real) polynomials of degree k and $\sigma(A)$ is the set of eigenvalues of $\mathcal{A}$. Then, only for a real symmetric matrix, convergence depends only on its eigenvalues. At each additional iteration the degree increases by one and so in such cases a reasonable accuracy can be quickly achieved. Various constructions based on the Chebyshev polynomials can give more explicit convergence bounds, but these are more complicated to write for indefinite symmetric matrices.

In most situations $\mathcal{M}$ is constructed in a way such that it is fast to solve the linear systems of the form $\mathcal{M}z = r$ for z when r is given.

It is usually advisable to preserve the symmetry of the system. In fact the iterative solution of non-symmetric linear systems is less reliable and often more expensive in general.

When using MINRES [10], a symmetric and positive definite preconditioner $\mathcal{M}$ needs to be employed so that it is possible to factorize $\mathcal{M}$ as $\mathcal{L}\mathcal{L}^\top$ for some matrix $\mathcal{L}$ (for example the Cholesky factor). This is only a mathematical artefact used to derive the method: this factorization is not required in practice though of course it could be used if it were available. The MINRES iteration is effectively applied to the symmetric system as

$$\mathcal{L}^{-1}\mathcal{A}\mathcal{L}^{-\top}y = \mathcal{L}^{-1}b, \quad \mathcal{L}^\top x = y \tag{4.16}$$

and convergence depends on the eigenvalues of the symmetric and indefinite matrix $\mathcal{L}^{-\top}\mathcal{A}\mathcal{L}^{-1}$.

Via the similarity transformation

$$\mathcal{L}^\top\mathcal{L}^{-\top}\mathcal{A}\mathcal{L}^{-\top}\mathcal{L}^\top = \mathcal{M}^{-1}\mathcal{A} \tag{4.17}$$

it is clear that the eigenvalues of the matrix $\mathcal{M}^{-1}\mathcal{A}$ determine the convergence of the method, hence (4.15) describes the convergence of the preconditioned MINRES iteration with the eigenvalue spectrum $\sigma(\mathcal{A})$ replaced in the preconditioned case by $\sigma(\mathcal{M}^{-1}\mathcal{A})$.

There are similar considerations and good preconditioners should satisfy similar criteria for SYMMLQ. As concerns SQMR (Simplified Quasi-Minimal Residual), it could generally only be used with a symmetric and indefinite preconditioner and there are no estimates of convergence in this case, but practical experience indicates that SQMR convergence can be very good with a suitable indefinite preconditioner [35].

4.4 Block preconditioners

Block preconditioners are based explicitly on the block factorization (4.4). The performance of these preconditioners depends on the availability of approximate fast solvers for linear systems involving $\boldsymbol{A}$ and the Schur complement $\boldsymbol{S}$. If we assume that $\boldsymbol{A}$ and $-\boldsymbol{S} = \boldsymbol{C} + \boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{B}^{\top}$ are both Symmetric Positive Definite, the ideal block diagonal preconditioner is :

$$\mathcal{P}_d = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{S} \end{pmatrix} \tag{4.18}$$

Preconditioning of $\mathcal{A}$ with $\mathcal{P}_d$ gives the matrix $\mathcal{M}$ [10].

$$\mathcal{M} = \mathcal{P}_d^{-1}\mathcal{A} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{A}^{-1}\boldsymbol{B}^{\top} \\ -\boldsymbol{S}^{-1}\boldsymbol{B} & \boldsymbol{0} \end{pmatrix} \tag{4.19}$$

The matrix $\mathcal{M}$ is symmetrizable, non-singular by assumption and it satisfies the following identity

$$(\mathcal{M} - \mathcal{I})\left(\mathcal{M} - \frac{1}{2}\left(1 + \sqrt{5}\right)\mathcal{I}\right)\left(\mathcal{M} - \frac{1}{2}\left(1 - \sqrt{5}\right)\mathcal{I}\right) = 0 \tag{4.20}$$

Hence $\mathcal{M}$ is diagonalizable and has only three distinct eigenvalues: 1, $\frac{1}{2}\left(1 + \sqrt{5}\right)$ and $\frac{1}{2}\left(1 - \sqrt{5}\right)$.
Therefore for each initial residual r_0, we have that $dim\mathcal{K}_{n+m}\left(\mathcal{M}, r_0\right) \leqslant 3$ which means that MINRES will terminate after at most three iterations if it is applied to the preconditioned system with the preconditioner $\mathcal{P}_d$. In the same way, the ideal block triangular preconditioner is:

$$\mathcal{P}_t = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^{\top} \\ \boldsymbol{0} & \pm\boldsymbol{S} \end{pmatrix} \tag{4.21}$$

A diagonalizable preconditioned matrix with only two distinct eigenvalues equal to ± 1 is obtained if we choose the minus sign in (4.21) while choosing the plus sign leads to a preconditioned matrix with all the eigenvalues equal to 1. In this case the matrix is non-diagonalizable, but has a minimum polynomial of degree two. With the other choice of the sign in (4.21), the non-symmetric iterative solver GMRES [38] is guaranteed to converge in exact arithmetic in at most two steps.

The ideal preconditioners $\mathcal{P}_d$ and $\mathcal{P}_t$ are not practical choices, because the exact Schur complement $\boldsymbol{S}$ is not available and is generally a dense matrix. Then in practice, $\boldsymbol{A}$ and $\boldsymbol{S}$ are substituted by some approximations,

$\hat{\boldsymbol{A}} \approx \boldsymbol{A}$ and $\hat{\boldsymbol{S}} \approx \boldsymbol{S}$. If the approximations are chosen appropriately, the preconditioned matrices have most of their eigenvalues clustered around the eigenvalues of the ideally preconditioned matrices $\mathcal{P}_d^{-1}\mathcal{A}$ and $\mathcal{P}_t^{-1}\mathcal{A}$. Of course the choice of the approximations $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{S}}$ is a highly problem-dependent. Often the matrices $\boldsymbol{A}$ and $\boldsymbol{S}$ are not explicitly available, rather it is given a prescription for computing the **action** of $\boldsymbol{A}^{-1}$ and $\boldsymbol{S}^{-1}$ on given vectors.

The application of these techniques to other saddle point problems is more problematic. For instance in the absence of elliptic operators it is not very clear how to build suitable approximations $\hat{\boldsymbol{A}} \approx \boldsymbol{A}$ and $\hat{\boldsymbol{S}} \approx \boldsymbol{S}$. One possibility is to use incomplete factorizations of $\boldsymbol{A}$ to build $\hat{\boldsymbol{A}}$, but it is unclear how to construct good approximations to the (typically dense) Schur complement $\boldsymbol{S}$. In this case an alternative approach is the constraint preconditioning [2, 10].

4.5 Silvester preconditioner

Although Stokes preconditioners with fixed viscosity are nowadays well established, there is less literature on Stokes preconditioners with varying viscosity. A first choice is the classical Silvester preconditioner [10]. Considering the problem:

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{B}^\top \\ \boldsymbol{B} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{V} \\ \boldsymbol{P} \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{4.22}$$

we note that it can be preconditioned ideally with

$$\mathcal{P} = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{S} \end{pmatrix} \tag{4.23}$$

where $\boldsymbol{S}$ is the Schur complement.
Hence (4.23) can be approximated leading to the Silvester preconditioner

$$\mathcal{P}_d = \begin{pmatrix} \hat{\boldsymbol{A}} & \boldsymbol{0} \\ \boldsymbol{0} & -diag\,(\boldsymbol{M}) \end{pmatrix} \tag{4.24}$$

where $\boldsymbol{A}$ is approximated in $\hat{\boldsymbol{A}}$, which is an easy invertible matrix and $\boldsymbol{S}$ is approximated in $diag\,(\boldsymbol{M})$, where $\boldsymbol{M}$ is the mass matrix of the pressure. Often $\boldsymbol{A}$ is approximated with a Multi-Grid algorithm that keeps the spectral properties unchanged. Let use the notation $\langle \cdot, \cdot \rangle$ for the Euclidean scalar product. Consider:

$$\mu_{min} = \inf_{\Omega} \mu(x), \quad \mu_{max} = \sup_{\Omega} \mu(x) \tag{4.25}$$

where the natural assumption is $\mu_{min} > 0$ and $\mu_{max} < \infty$.

Let consider finite element:
velocity $X_h \subset [H^1(\Omega)]^d$ and pressure $S_h \subset L_0^2(\Omega)$ spaces. Assume that the spaces pair X_h, S_h is stable in the Ladyzenskaja-Babuska-Brezzi sense so there exists a mesh independent constant $c_0 > 0$ such that:

$$c_0 < \inf_{q_h \in \mathbb{Q}_h} \sup_{v_h \in \mathbb{V}_h} \frac{(q_h, \nabla \cdot \mathbf{v}_h)}{\|q_h\| \|\nabla \cdot \mathbf{v}_h\|} \tag{4.26}$$

then it holds:

$$c_0^2 \mu_{max}^{-1} \boldsymbol{M} \leqslant \boldsymbol{S} \leqslant \mu_{min}^{-1} \boldsymbol{M} \tag{4.27}$$

From this result it follows:

$$cond\left(\boldsymbol{M}^{-1}\boldsymbol{S}\right) \leqslant c_0^{-2} \frac{\mu_{max}}{\mu_{min}} \tag{4.28}$$

Indeed we need a particular attention to the ratio Δ because it can make explode the condition number if not opportunely treated. In [19] they proved better estimates than (4.28) but only in the special case of Bingham fluids.

Then, the preconditioner with $\boldsymbol{M}$ rescaled is not effective for varying viscosity as detailed also in [45], where it is proposed to rescale $\boldsymbol{M}$ using the ratio

$$\Delta = \frac{\mu_{max}}{\mu_{min}} \tag{4.29}$$

There are various ways to operate:

- rescaling the entire matrix by a fixed global Δ,
- rescaling the local finite element matrices by a local Δ calculated only in the finite element [37].

Furthermore, sometimes only rescaling could not be effective as reported in [5], where the rescaled lumping of the $\boldsymbol{M}$ is considered.

4.6 Other common techniques

Other preconditioning techniques have been proposed in the literature on saddle point problems. For instance, the classical Uzawa method can be shown to be a special type of block triangular preconditioner. Another example is the Augmented Lagrangian formulation: the assumption that $\boldsymbol{A}$ is non-singular may be too restrictive and in fact $\boldsymbol{A}$ is singular in many applications. However, it is often possible to use augmented Lagrangian techniques [13] to replace the original saddle point system with an equivalent one having the same solution but in which $\boldsymbol{A}$ is now non-singular. Then, block diagonal and block triangular preconditioners based on approximate Schur complement techniques could still be applicable. The augmented Lagrangian idea can also be useful in cases where $\boldsymbol{A}$ is highly ill-conditioned and in order to transform the original saddle point system into one that is easier to precondition.

CHAPTER 5

HPC concepts and architectures

In this chapter we introduce some technical details of the High Performance Computing machines that have been used and some important definitions necessary to understand the analysis of the results of the next chapter.

5.1 Definitions

A very important concept in parallel computing is the **scalability**, which is a measure of the capability of an algorithm to perform when increasing the number of parallel tasks and/or workload.

Albeit the term scalability is widely used throughout literature in several application fields, many different operational definitions and interpretations coexist. A general definition is given in [27], which makes an abstraction from the specific application considered. The definition of scalability requires a performance metric typically function of several parameters. The choice of the performance metric strongly influences the type of scalability considered, which is usually reached for a bounded region in the parameters space.

We consider the **parallel efficiency** as performance metric. While floating point efficiency is usually measured in FLOPS (Floating Point Op-

erations per Second) and hardware efficiency is measured in FLOPS/W (FLOPS per Watt), the parallel efficiency of an operation is here defined as the inverse of the elapsed time T spent to perform such an operation. We distinguish between strong and weak scalability:

- **Strong** scalability: to assess the strong scalability the workload is fixed and we measure the parallel efficiency while varying the numbers of processors. Optimal scalability implies that efficiency is proportional to the number of processors, yet it is very difficult to obtain it in practice. The overall time T is the sum of the time spent for the computations, and that spent for the communications between processors. While for the former operations the performance scales, for the latter it does not and for most algorithms, as the workload per process decreases the relative importance of communication increases.
- **Weak** scalability: in this case the ratio between the workload and the number of processor is kept constant. Weak scalability measures the capability of the algorithm to handle problems of increasing size. Optimal scalability implies, in this case, a constant parallel efficiency. The lack of weak scalability can be caused by either a loss of performance of the computing part, or by an increase of the latency time introduced by the hardware communication.

We already remarked that different definitions of performance lead to different types of scalability. For instance, in a domain decomposition context, parallel scalability is often interpreted as the independence of the convergence rate from the sub-domain size H [34]. This is in analogy with general requirement of independence from the mesh size h.

Another important indicator for parallel efficiency is the **speedup** σ, whose definition is

$$\sigma = \frac{T_{serial}}{T} \tag{5.1}$$

where T_{serial} is the time spent in the serial execution. If an algorithm is strongly scalable, σ is the *number of processors*.

5.2 Machine used

We have used 4 high performance machines that belong to barely three generations:

- the first generation is represented by Sp6,
- the second generation is represented by SGI Altix ICE 8200 and Blue Gene/P,
- the third generation is represented by Blue Gene/Q.

Figure 5.1: *The Sp6 machine.*

5.2.1 Sp6 machine

We used the Sp6 machine installed at CINECA (Consorzio interuniversitario dell'Italia NordEst) in Casalecchio di Reno (Bologna, Italy). The Sp6 machine 5.1 is a IBM pSeries 575 cluster based on IBM Power6 processor, 4.7 GHz with Simultaneous Multi-Threading (SMT) support. The Internal Network is a fat-tree Infiniband x4 DDR and the Operating System is AIX 6. It has 168 Computing Nodes and 5376 Computing Cores (32 core/node). The RAM is 21 TB (128 Gb/node), while the total disk space is around 1.2 PB.

The peek power is 101 TFLOPS versus 3 TFLOPS of Sp5, the previous machine. Every node has 128 GB of shared memory. The particularity of these machine are the AIX Operating System.

5.2.2 Blue Gene/P machine

We used the Blue Gene/P machine installed at CINECA. The design of Blue Gene/P is the evolution from Blue Gene/L. Each Blue Gene/P Compute chip contains four PowerPC450 processor cores, running at 850 MHz. The cores are cache coherent and the chip can operate as a 4-way symmetric multiprocessor (SMP). The memory subsystem on the chip consists of small private L2 caches, a central shared 8 MB L3 cache, and dual DDR2 memory controllers. In Figure 5.2 is depicted a Blue Gene/P machine.

The chip also integrates the logic for node-to-node communication, using the same network topology as Blue Gene/L, but at with the double of the bandwidth. A compute card contains a Blue Gene/P chip with 2 or 4 GB DRAM, comprising a "compute node".

A single compute node has a peak performance of 13.6 GFLOPS. 32 Compute cards are plugged into an air-cooled node board. A rack contains 32 node boards (thus 1024 nodes, 4096 processor cores). By using many small, low-power, and densely packaged chips, Blue Gene/P exceeded the power efficiency of other supercomputers of its generation.

The Blue Gene/P supercomputer and its predecessor are unique in the following aspects:

- Trading the speed of processors for lower power consumption. Blue Gene uses low frequency and low power embedded PowerPC cores with floating point accelerators. While the performance of each chip is relatively low, the system could achieve better performance to energy ratio, for applications that could use larger numbers of nodes. Moreover also the memory per core is small: 512 MB or in some rare cases 1 GB.

- Dual processors per node with two working modes: co-processor mode where one processor handles the computation and the other handles communications; and virtual-node mode, where both processors are available to run the code, but the processors share both the computation and the communication load.

- System-on-a-chip design. All node components were embedded on one chip, with the exception of 512 MB external DRAM.

- A large number of nodes (scalable in increments of 1024 up to at least $65,536$)
- Three-dimensional torus interconnect with auxiliary networks for global communications (broadcast and reductions), I/O and management;
- Lightweight OS per node for minimum system overhead (system noise).

Albeit these features are needed to improve the parallel scalability, they also make more difficult to use the machine.

Figure 5.2: *The Blue Gene/P machine.*

5.2.3 SGI Altix ICE 8200 machine

We have used the SGI Altix ICE 8200 called *Jade* installed at CINES (*Centre Informatique National de l'Enseignement Supérieur*) in Montpellier (France) thanks to HPC-Europa2 project.

SGI Altix ICE 8200 is a cluster made of 46 racks as can be seen in Figure 5.3. It has a peak power of 267 TFLOPS. It has $23,040$ cores spread over 2880 blades (nodes) each one with two Intel Quad-Core E5472 (name-code *Harpertown*) and X5560 (name-code *Nehalem*) processors.

The network is an Infiniband (IB 4x DDR) double plane network for the Harpertown section of the machine. On the Nehalem section of the machine, 4 InfiniBand 4X QDR drivers provide 72 IB 4X QDR ports on output of each IRU (576 GB/s). Each IRU encloses 16 compute blades (nodes), while there are 2 processors per node (blade). Each processor has 4 cores (Harpertown 3 GHz) and 4 GB of memory per core. By default the

IB plan is used for MPI communications with a non-blocking hypercube topology and the other one is used for access to the parallel file system LUSTRE.

It is possible to compare Blue Gene/P and SGI Altix ICE 8200 because they belong to the same generation even if they are based on different philosophies: SGI Altix ICE 8200 is based on powerful processors and lot of memory per node while Blue Gene/P has much less memory per node and slower processors, but it has three proprietary dedicated networks.

Figure 5.3: *The SGI Altix ICE 8200 machine installed at CINES, Montpellier, France.*

5.2.4 Blue Gene/Q machine

We have used the Blue Gene/Q machine called *Fermi* installed at CINECA. The Blue Gene/Q Compute chip is called *A2* and it is a 64-bit PowerPC processor with 18 cores which are 4-way simultaneously multi-threaded, and they run at 1.6 GHz. 16 processor cores are used for computing, while a 17th core is used for operating system assist functions such as interrupts, asynchronous I/O, MPI pacing and RAS. Lastly the 18th core is used as a redundant spare, used to increase manufacturing yield. Each processor core has a SIMD Quad-vector double precision floating point unit (IBM QPX). In this Blue Gene generation IBM has decided to improve the computational power of the CPUs since it was the Blue Gene Achille's heel.

It delivers a peak performance of 204.8 GFLOPS at 1.6 GHz. The chip is mounted on a compute card along with 16 GB DDR3 DRAM (i.e., 1 GB for each user processor core).

A compute drawer have 32 compute cards, each water cooled. A "midplane" (crate) of 16 compute drawers have a total of 512 compute nodes,

electrically interconnected in a 5D torus configuration (4x4x4x4x2). Beyond the midplane level, all connections are optical. Racks have two midplanes, thus 32 compute drawers, for a total of 1024 compute nodes, 16,384 user cores and 16 TB RAM. Separate I/O drawers, placed at the top of a rack or in a separate rack, are air cooled and contain 8 compute cards and 8 PCIe expansion slots for Infiniband networking [1]. Blue Gene/Q systems also topped the Green500 list of most energy efficient supercomputers with up to 2.1 GFLOPS/W. Blue Gene/Q has been ranked as 7-th powerful high performance machine in the world in June 2012.

Figure 5.4: *The Blue Gene/Q machine installed at CINECA, Casalecchio di Reno, Bologna, Italy.*

To conclude we report a Table 5.1 that highlights the main features of these machines.

Machine	Processor	Network	Proc. Freq. (GHz)	Vendor
Sp6	Power6	fat-tree Infiniband x4 DDR	4.7	IBM
Altix ICE 8200	Intel Quad-Core E5472/X5560	Infiniband (IB 4x DDR)	3.0	SGI
Blue Gene/P	PowerPC450	3D proprietary torus	0.85	IBM
Blue Gene/Q	PowerPc A2	5D proprietary torus	1.6	IBM

Table 5.1: *Main architecture features of Sp6, Blue Gene and SGI Altix ICE 8200.*

CHAPTER 6

Numerical tests

In this chapter we investigate the parallel performance of the Stokes solver implemented in LifeV on various HPC machines of the type already introduces in section 5.2. We finally report weak and strong scalability results for some of the preconditioners described in the next section.

6.1 Numerical results on the Sp6 machine

On the Sp6 machine (for further details on this machine see 5.2) we tested the performance of the code considering only the Stokes solver. The tracking solver at that time was not available. We have considered constant density and viscosity. The main purpose was to test the behaviour of the code. We focused on how toughen up the code looking to bottlenecks, memory leaks and any other common problems. We have considered a preliminary version of the Stokes solver that has been preconditioned with a black-box preconditioner from the Ifpack package [39], in particular it was ILU-DD.

In the Figure 6.1 we report the speed up of the Stokes solver. Being a preliminary test, it was performed with 1 up to 512 processors (strong scalability) using only 200.000 tetrahedra. $\mathbb{P}^2$ and $\mathbb{P}^1$ finite elements were used respectively for velocity and pressure. The binary was optimized tuning the

compiler flags, in order to obtain the maximum performance targeting the underling architecture.

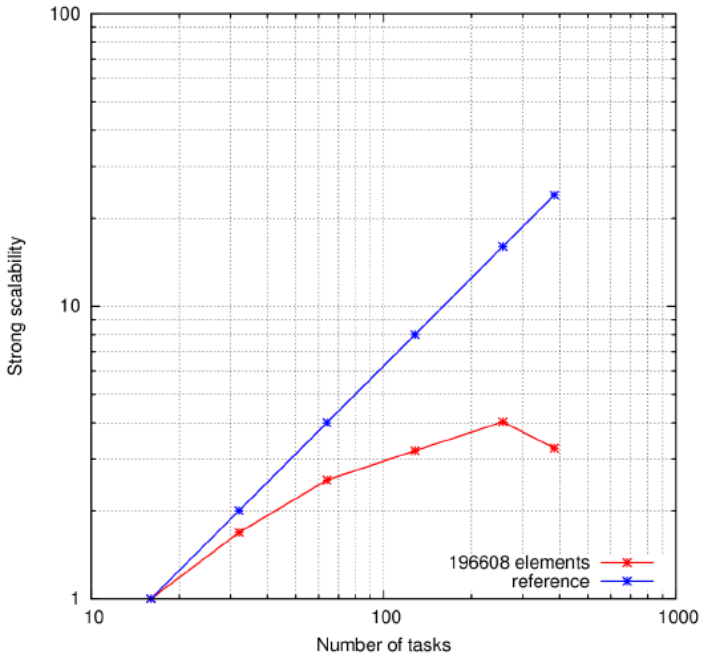

Figure 6.1: *The strong scalability on AIX relative to eight tasks.*

Looking at Figure 6.1 it is clear that the behaviour is far from optimal. Since it is a strong scalability plot, it is normal to see a degradation of the performance because the data processed per processors lowers, while it increases the numbers of processors, but it is not the only reason.

Analysing the data at every step of the simulation, we have noticed that the time to read and partition the mesh rise steeply when increasing the number of CPUs as can be seen in Figure 6.2. This phenomenon worsen with bigger meshes since every processor read and partition the mesh taking only its local part.

Thus, a big issue is the i/o time, indeed the number of iteration increases remarkably when the number of processors increase as can been seen in Figure 6.3. In fact the preconditioner used does not scale properly, in particular we numerically see that the solution does not even converge with more 395 processors. This type of preconditioners may be a good solution only for a small number of processors (up to 64 processors).

Lastly another bottleneck is the writing of output file, as can been seen in Figure 6.4. Pie charts that depicts the ratio between the various parts of

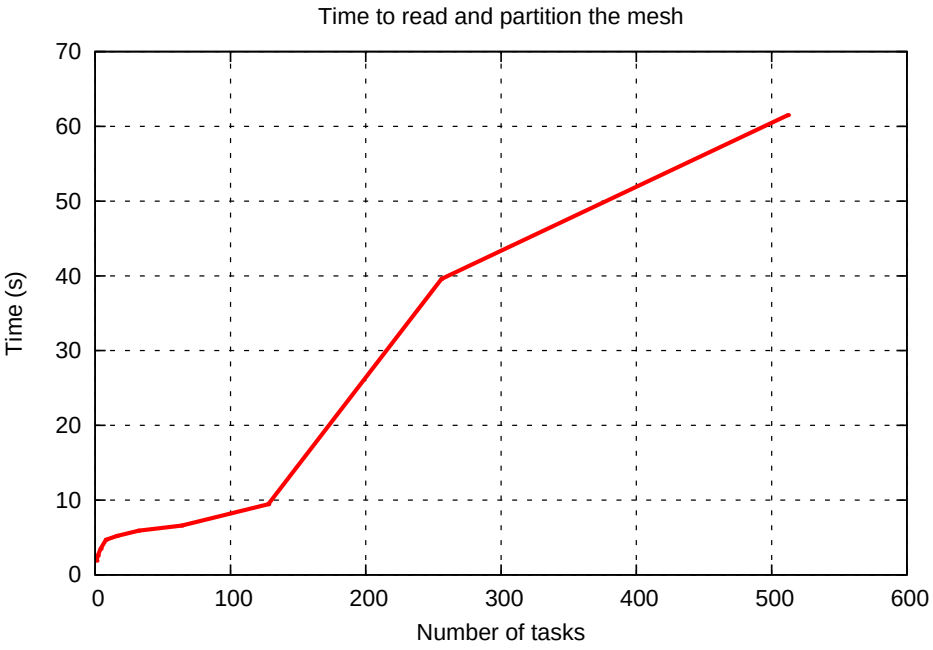

Figure 6.2: *The time to read and partition the mesh. In this case the mesh had 25,000 elements*

the simulation: in blue the time spent building the space finite elements, in brown the time to assemble the matrix of the linear system, in orange the time to write the solution in a HDF5 file, in light blue the time to read and partition the mesh, in green the time to set the system, in yellow the time to apply the boundary conditions.

The results of the test, carried on the preliminary version of the code, has put in evidence the importance of an efficient I/O.

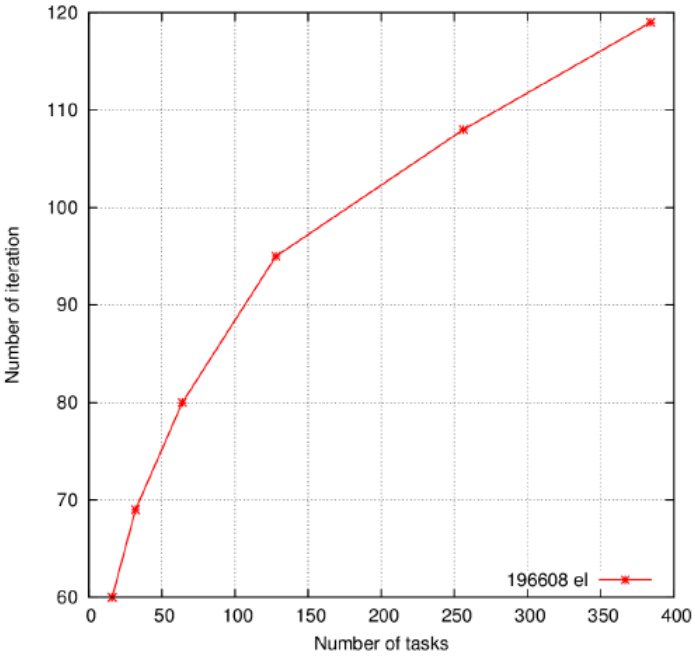

Figure 6.3: *Iteration number of the preconditioner only. The conditioning of the preconditioned matrix worsens when increasing the number of processors at a point (395 processors) that the simulation does not converge.*

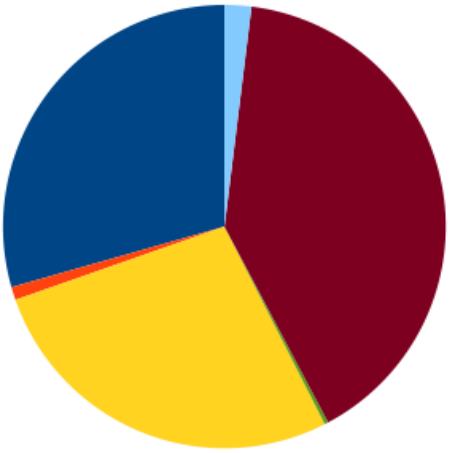

Figure 6.4: *Serial case.*

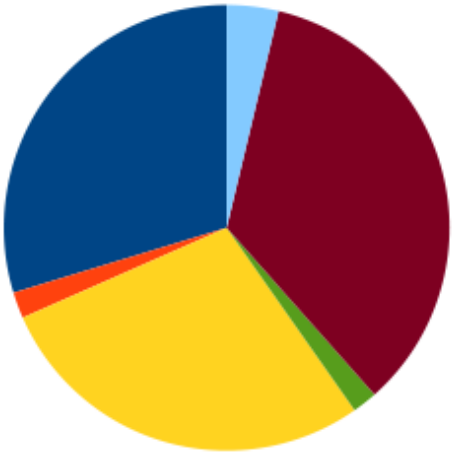

Figure 6.5: *Two processors.*

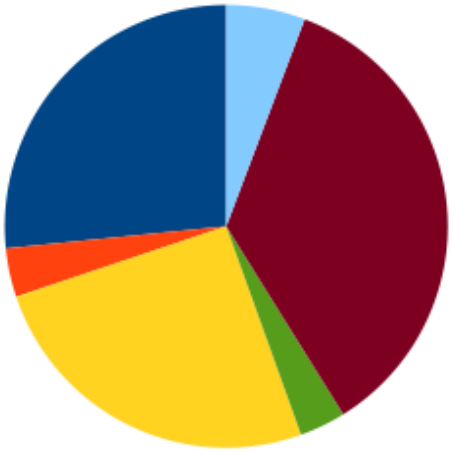

Figure 6.6: *Four processors.*

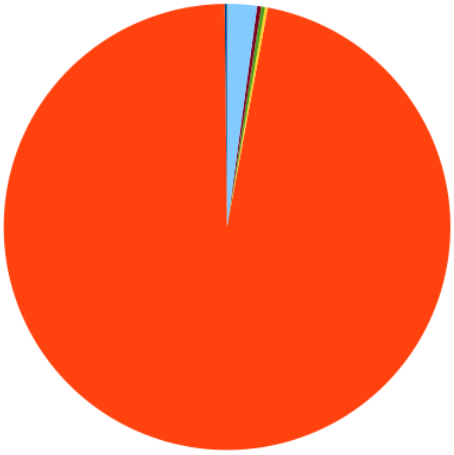

Figure 6.7: *256 CPUs.*

Figure 6.8: *Pie charts of the various parts of the simulation: in blue the time spent building the space finite elements, in brown the time to assemble the matrix of the linear system, in orange the time to write the solution in a HDF5 file, in light blue the time to read and partition the mesh, in green the time to set the system, in yellow the time to apply the boundary conditions. Note in orange the time to write the output.*

6.2 Numerical results on Blue Gene/P machine

On Blue Gene/P the output bottleneck has been resolved changing the interface from HDF5 1.6 to HDF5 1.8 as can be seen in Figure 6.9: in blue the old interface, in pink the new one, in green the new interface using the compression while in red the serial text-based output. We can notice that the compression is not relevant thanks to large network bandwidth of the Blue Gene/P.

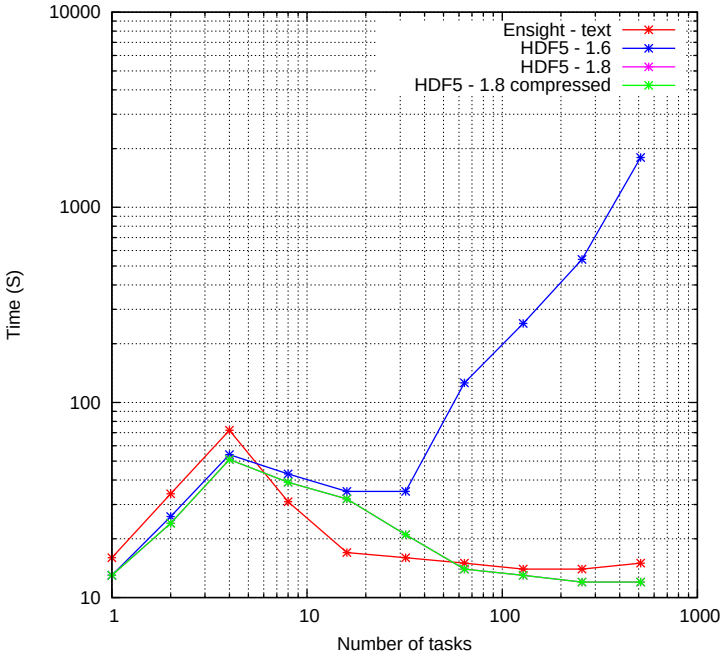

Figure 6.9: *The time to write the output file varying the number of processors. Purple line is hidden by the green line, in this test we did not seen any difference between the compressed and not compressed HDF5 output.*

We decided to off-load the partitioning of the mesh from the simulation to avoid the bottleneck when reading the mesh, the results can be seen in Figure 6.10 and in Table 6.1. In red we have the **online** partitioning (during the computation) while in blue and green the **offline** one. During online partitioning each processor reads the entire mesh and partitions it taking its own local part, while during offline partitioning only the local part of the mesh is read.

After solving the I/O bottleneck, the scalability on Blue Gene/P improved distinctively as can be seen in Figure 6.11, although it was not opti-

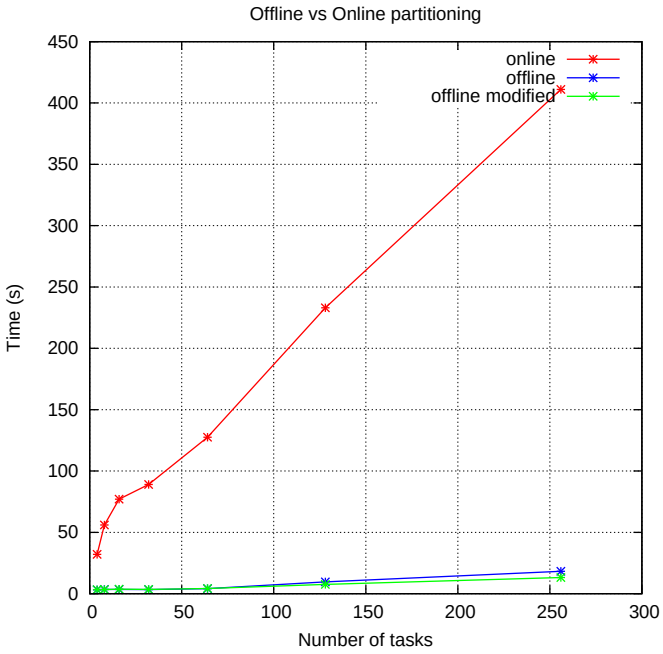

Figure 6.10: *Time to read and partition the mesh varying the number of processors.*

elements	offline (s)	online (s)
196,608	3.3	127.5
1,572,864	34.2	842.3

Table 6.1: *Comparison between offline and online partitioning with two different number of elements.*

mal. This inefficiency was due to high memory footprint since on the Blue Gene/P there is only 1 GB per processor and this lead to have only a few number of DOFs (Degree of Freedom) per processor (i.e. $100,000$ DOFs) before saturating the processor memory.

From this consideration, we started to investigate tools to reduce memory requirements and we adopted Scalasca [15], Massif and Tau [43] to profile the code. In this way we identified the slowest routines and some unneeded functions which we have decided to remove. Additionally we have substituted `Boost::uBLAS` with C-like vectors/matrices whenever possible.

The overall performance improvement was of just 5%, and unfortunately we did not notice any remarkable decrease of memory footprint. Therefore

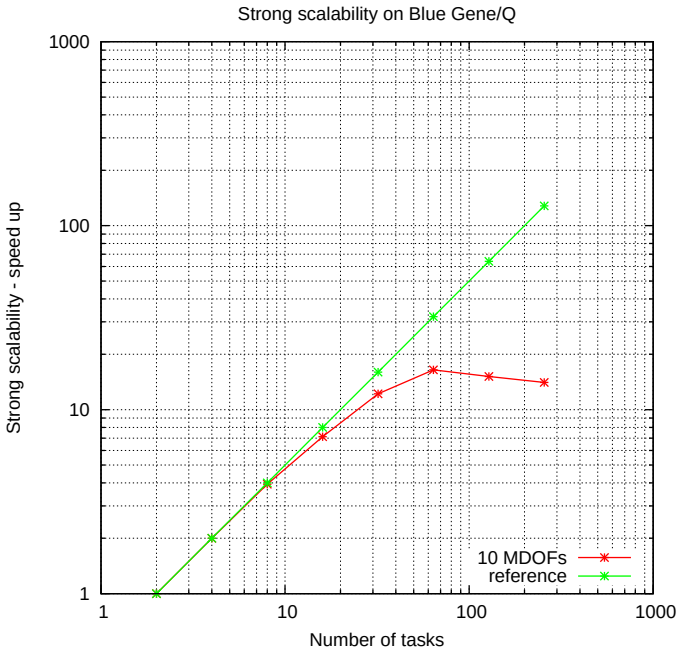

Figure 6.11: *Strong scalability on Blue Gene/P.*

we decided to rethink our code in a way that should occupy less memory.

Another important point is to reduce the communication, for example by creating the graph for matrix initialization before the creation of the matrices. Further information on this issues is contained in the next chapter 7.2.2.

Beside this, we have implemented lighter quadrature rules to heavily reduce the memory footprint as can be seen in Table 6.2. Also finite elements can impact on the memory footprint. Indeed in our case have we have considered only the stable couples: $\mathbb{P}_b^1$ / $\mathbb{P}^1$ and $\mathbb{P}^1 nc$ / $\mathbb{P}^0$ respectively for the velocity and for the pressure fields and we abandoned the couple $\mathbb{P}^2$ / $\mathbb{P}^1$ since it is heavier than the previous ones.

elements	qR 4 pt	qR 15 pt	qR 64 pt
196,608	786,432	2,949,120	12,582,912
1,572,864	6,291,456	94,371,840	6,039,797,760

Table 6.2: *Comparison between different types of quadrature rules: number of doubles stored for every quadrature rule in case of two different meshes.*

Other important points are the choice of the linear solver and the pre-

conditioner. As a solver we considered both algorithms in Belos and in AztecOO [20], while Belos has many more solver like MINRES, AztecOO has been used long assuring high reliability. On Blue Gene/P we have used GMRES.

6.2.1 Preconditioning

When we consider optimal preconditioners as those illustrated in 4.5, it is very important to select a suitable preconditioner for the elliptic part of the problem.

We decide to use an Algebraic Multi-Grid (AMG) preconditioner because it is very scalable and it has a low memory footprint, since it does not built directly the preconditioner.

There are only a few open-source multi-grid libraries that can scale well on high performance machines, the most used are ML and Hypre. In our case we have chosen the ML package from the Trilinos library since the version of Hypre at that time suffered scalability problems [5] and furthermore our code was already based on Trilinos.

Compared to geometric multi-grid, AMG can have advantages because of its ability to take into account for variations in viscosity and adaptively refined meshes in the grid hierarchy. AMG requires a setup phase, in which a coarse grid hierarchy and corresponding restriction and interpolation operators are constructed.

In ML, we use a processor-local (uncoupled) coarse grid aggregation scheme. When the number of unknowns per processor becomes small in the aggregation process, we repartition to a smaller number of processors. The new parallel partitioning often allows aggregation of unknowns originally on different processors. We use an aggregation threshold of 0.01, and 3 sweeps of smoother for both the pre- and post-smoothing. As a smoother we considered several choices: the AztecOO implementation of GMRES, ILU, symmetric Gauß-Seidel, Jacobi. For this choice of ML parameters, the small coarse grid problem is set up on a single processor and solved by a direct method (SuperLU [29]).

The most delicate parameter in a AMG algorithm is the smoother, in fact it highly affects the cost to construct the preconditioner and the preconditioner spectral properties. GMRES and ILU are proved to be the most robust smoothers, while Cholesky, Gauß-Seidel, Jacobi are cheaper but can fail occasionally when increasing the number of processor or using more "distorted" meshes. It is important to notice that ILU smoother is usable only with small meshes, since with larger meshes the time to build the

preconditioner becomes infeasible. The coarse grid aggregation scheme is very important, but often a processor-local (uncoupled) scheme is scalable and robust. In some case it is important to use the energy minimization scheme built in ML to find good damping parameters for the prolongator smoothing, like in the case of Gauß-Seidel smoother that in this way can outperform GMRES smoother.

After this improvements we moved to test the code on SGI Altix ICE 8200 and on Blue Gene/Q machines.

6.3 Numerical results on SGI Altix ICE 8200

SGI Altix ICE 8200 has a totally different architecture compared to Blue Gene (see 5.2.3), for this reason we tested our code on this machine to see its behaviour compared to the Blue Gene architecture.

As shown in Figure 6.12 it is preferable to use a Domain Decomposition ILU in a parallel environment up to 32 processors as it is easier to setup and it still has good performance.

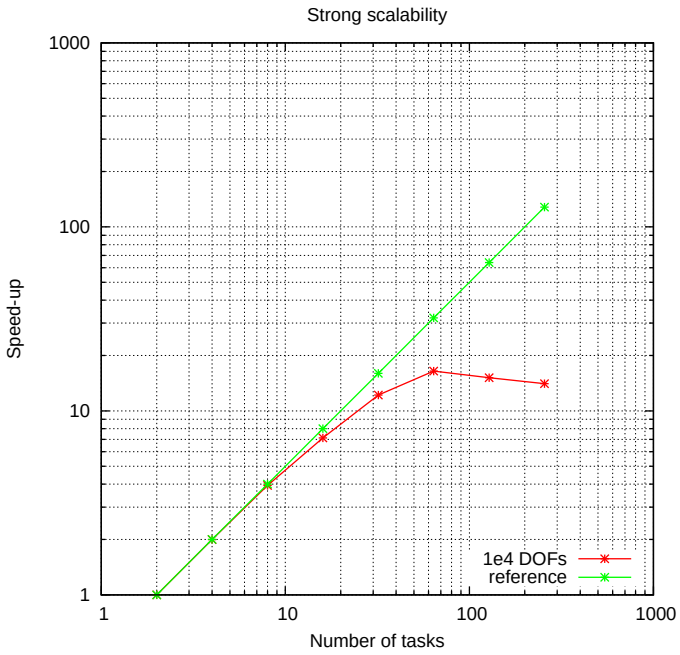

Figure 6.12: *Strong scalability of GMRES solver preconditioned with ILU-DD from Ifpack library on SGI Altix ICE 8200.*

When increasing the number of processors, the only viable choice is a Multi-Grid preconditioner. In Figure 6.13 we show the weak scalability between various smoothers. ILU smoother is the smoother that occupies the most amount of the memory since it actually builds physically the preconditioner. Notice that Gauß-Seidel with repartition an AztecOO smoothers is the most scalable.

On SGI Altix ICE 8200 we were able to run a simulation up to 40 million DOFs and in Figure 6.14 we report the speed up of the linear solver. The best smoothers that we found are based on GMRES (used as a smoother) and on Gauß-Seidel with repartition, because they are cheap to apply and

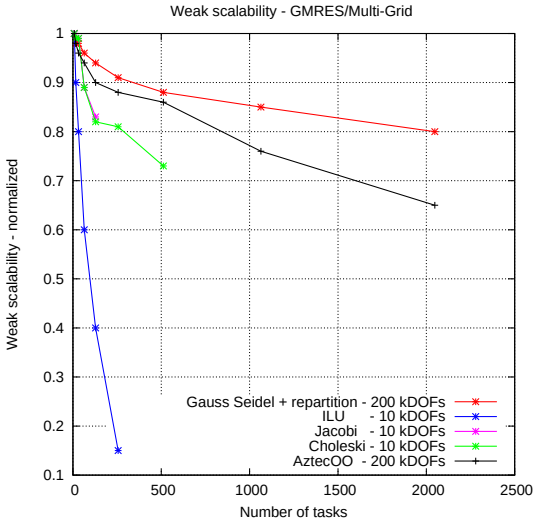

Figure 6.13: *Weak scalability of GMRES solver preconditioned with ML with different smoothers on SGI Altix ICE 8200.*

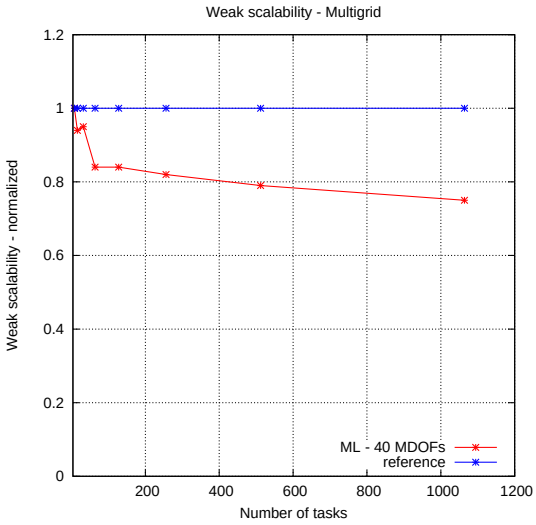

Figure 6.14: *Weak scalability of GMRES solver preconditioned with ML,* $40,000$ *DOFs per processor up to* 40 *MDOFs.*

do not use a lot of memory. In Figure 6.15 we report the number of GMRES iterations of the solver depending on the type of smoother and on the number of processors. In blue the ILU smoother leads to the less iteration than GMRES smoother (in red). Notice the importance of repartition (the

rebalancing) in case of Gauß-Seidel smoother: in pink the case without rebalancing that cannot converge with more than 8 processors, while the case with rebalancing (the green line) outperforms even the other smoothers.

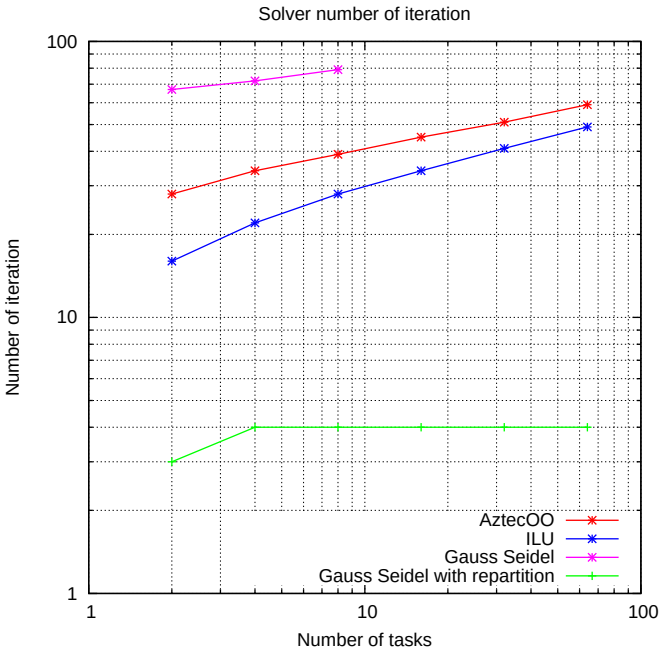

Figure 6.15: *Number of iterations of the GMRES solver preconditioned with ML with different smoothers.*

The time to setup the preconditioner plays an important role. In Figure 6.16 the timings to setup the Multi-Grid preconditioner are shown: in red a case of a ILU smoother with 10^5 DOFs, in blue and green a Gauß-Seidel smoother with respectively 10^5 DOFs and 10^7 DOFs and then a AztecOO smoother with 10^7 DOFs. Notice the remarkably time difference between Gauß-Seidel/AztecOO and ILU.

Lastly we compare the Blue Gene/P and SGI Altix ICE 8200, as illustrated in Figure 6.17. The red line is the reference, the brown line is the Blue Gene/P in case of 5.8 MDOFs while the blue and green lines are respectively the SGI Altix ICE 8200 for 42 MDOFs and 5.8 MDOFs. To achieve 42 MDOFs we have run one processor per node to obtain the maximum memory in the node.

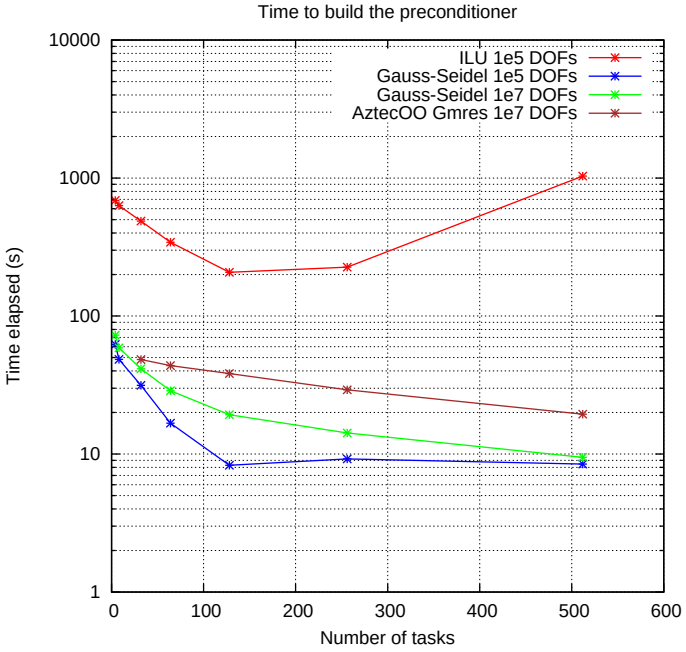

Figure 6.16: *Time to construct the multi-grid preconditioner.*

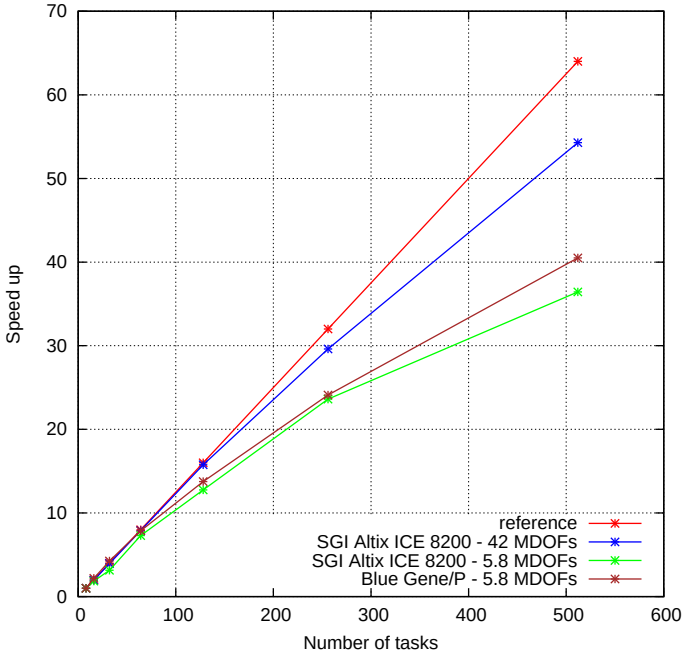

Figure 6.17: *Speed up comparing Blue Gene/P and SGI Altix ICE 8200 machines.*

6.3.1 Schur complement preconditioning

In this section we present various results considering highly changing viscosities and densities. We choose the Multi-Grid preconditioner of the previous sections for the elliptic part. We test different choices to precondition the Schur matrix:

- the classical $diag\,(\boldsymbol{M})$ rescaled by Δ,
- the matrix $\boldsymbol{M}$ lumped and rescaled by Δ,
- the entire matrix $\boldsymbol{M}$ rescaled by Δ.

We have decided to rescale by a global fixed value Δ. In Figure 6.18 we can see the results. In red the lumping of $\boldsymbol{M}$, in green the $diag\,(\boldsymbol{M})$ and in blue the approximate inverse of $\boldsymbol{M}$ based on a Multi-Grid W-cycle. We can notice that the $diag\,(\boldsymbol{M})$ works up to $\Delta = 10^3$ and the lumped $\boldsymbol{M}$ works until $\Delta = 10^8$, while the approximate inverse of $\boldsymbol{M}$ works very well up to $\Delta = 10^{14}$.

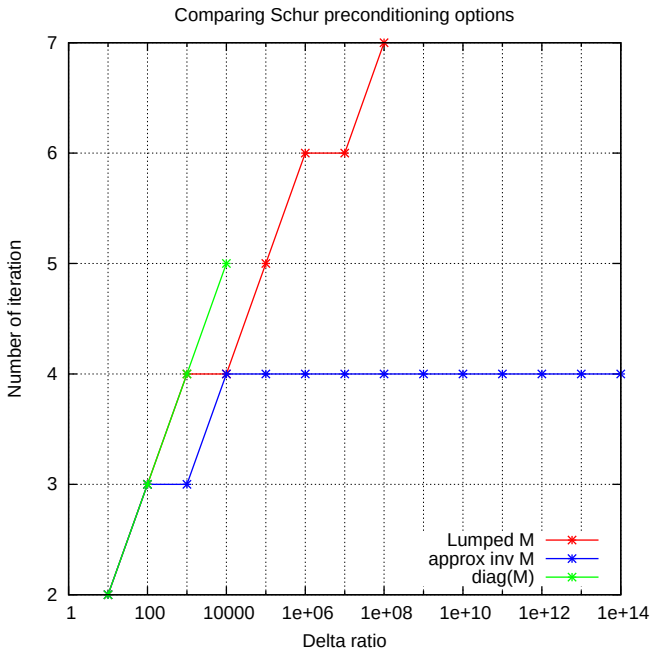

Figure 6.18: *The number of solver iteration as increase* Δ.

Therefore we choose the approximate Multi-Grid based of $\boldsymbol{M}$ as preconditioner. In Figure 6.19 we investigated its behaviour with bigger meshes, as in Figure 6.18 we report the Lumped $\boldsymbol{M}$ $(400,000$ DOFs) and the

Multi-Grid approximate based of $\boldsymbol{M}$ $(400,000$ DOFs) respectively in red and blue, while we report in green the Multi-Grid approximate based of $\boldsymbol{M}$ $(1$ MDOFs) with 64 cores on Blue Gene/Q machine (for further details on Blue Gene/Q see 5.2.4). In the case of 1 MDOFs, the increase of 1 iteration is mainly due to the increasing number of mesh elements.

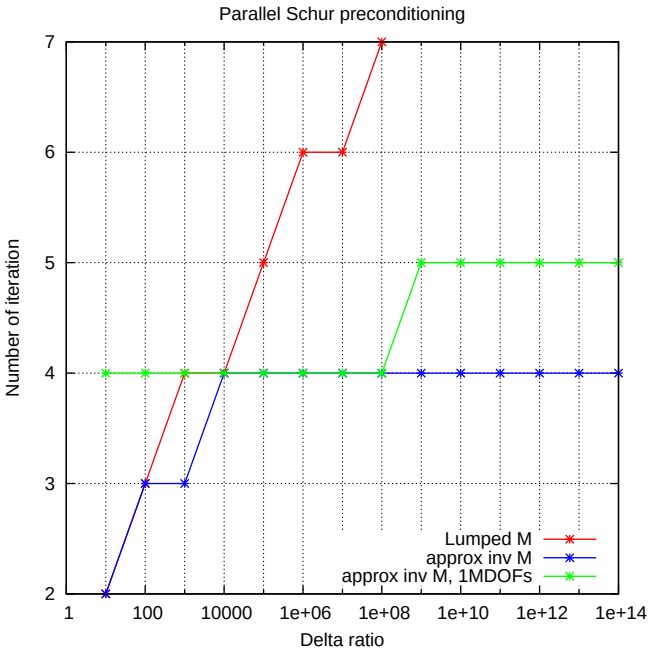

Figure 6.19: *The number of solver iterations as increase* Δ *with* 64 *processors.*

In Figure 6.20 we can notice that, even increasing the number of DOFs, the number of iterations remain barely constant. All tests reported in Figure 6.20 have been performed with 64 processors; we notice that with more than 1 MDOFs we need to substitute ParMetis with Zoltan in the repartitioning because ParMetis freezes killing the simulation. In red a mesh of 100 kDOFs, in blue a mesh of 1 MDOFs and in green a mesh of 10 MDOFs.

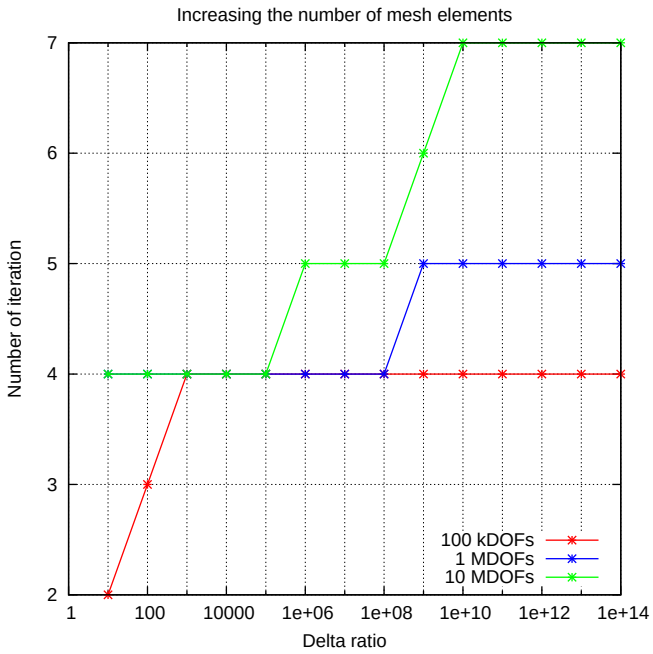

Figure 6.20: *Number of iterations increasing the mesh dimensions.*

Heterogeneous density

In Figure 6.21 we show that the preconditioner performs well with small variations of density, and in Figure 6.22 we can see that even in presence of high variation of density the preconditioner performs well.

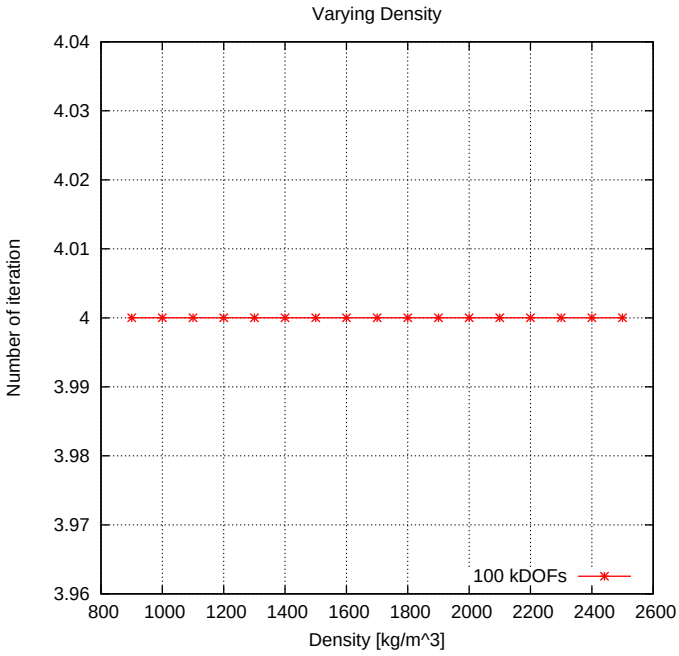

Figure 6.21: *Number of iterations when increasing density.*

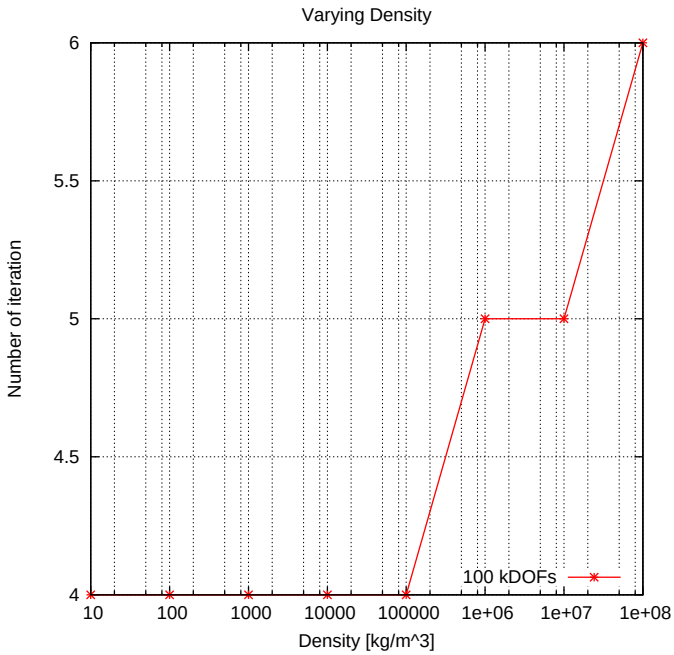

Figure 6.22: *Number of iterations when increasing density of different order of magnitude.*

6.4 Numerical results on Blue Gene/Q machine

On Blue Gene/Q we have focused on investigating various linear system solvers and testing various multi-grid parameters. In Figure 6.23 we present the comparison between various solvers in Belos and in AztecOO packages. We have considered only the solvers that can be used with the Stokes problem: the GMRES solver from AztecOO and the BlockGMRES, MINRES, GCRODR (GMRES with recycling) from Belos.

The size of the problem was of 400 kDOFs: every solver was able to converge in 4 iterations, but the GMRES family of solver was always faster both in Belos and AztecOO packages. Surprisingly we can notice that the Flexible GMRES in Belos resulted slightly faster (1.3 s) than BlockGmres Belos solver.

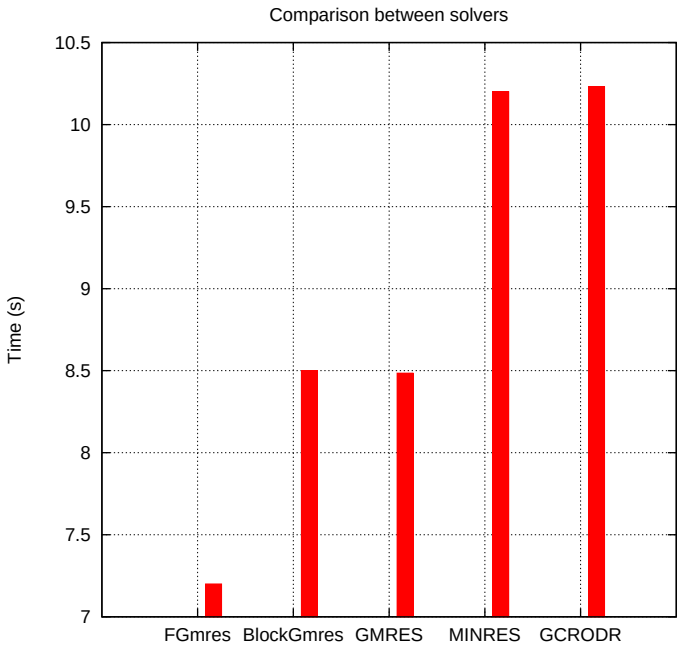

Figure 6.23: *Comparison of solver time with* 400 *kDOFs on* 4 *processors between various solvers in Belos and AztecOO.*

In Figure 6.24 we show that number of iterations of the solver can be influenced by the rebalance algorithm used. In red the number of iteration per numbers of processors using graph partitioning ParMetis algorithm, while in blue we show the number of iteration per numbers of processors using the Zoltan Hyper-graph partitioning. For more than 64 processors only Zoltan

works.

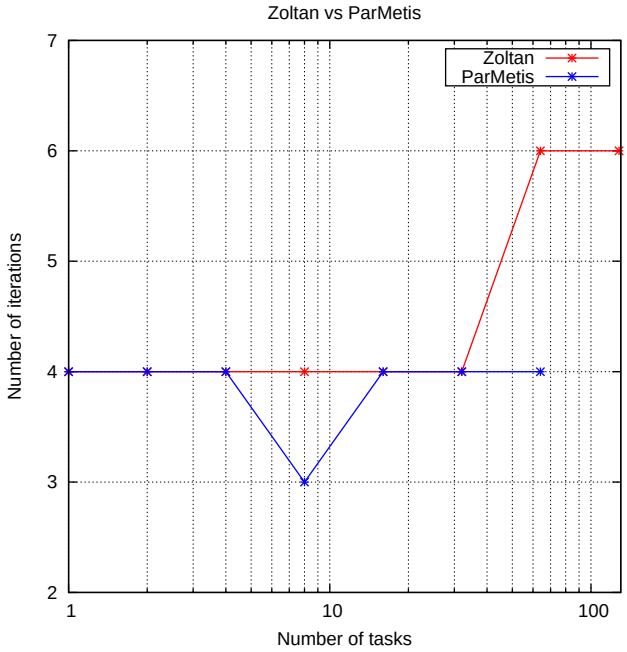

Figure 6.24: *Number of iterations using ParMetis or Zoltan as a repartition algorithm.*

Lastly we show, in Figure 6.25, the weak scalability on Blue Gene/Q considering about $450,000$ DOFs per processor: in green the time to apply the boundary conditions, in black the time to construct the finite elements spaces, in pink the time to assemble all the operators, in blue the setup of the preconditioner, and in red the time spent to solve the linear system.

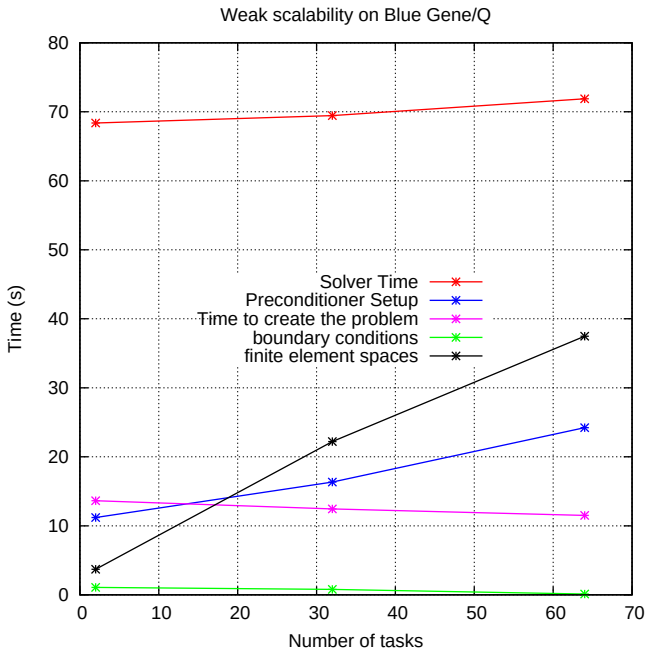

Figure 6.25: *Weak scalability on Blue Gene/Q.*

CHAPTER 7

Implementation aspects

This chapter is sub-divided in three parts: in the first part we describe some general aspects of the porting process on HPC machines in particular the Blue Gene architecture. In the second part we illustrate the porting of the Stokes solver and finally we describe the tracking framework.

7.1 Porting

We ported the LifeV project [9] to RISC processor in particular to Power6 processor and to AIX Operating System . The porting required to remove the dependencies from GCC/GNU tools of the original program and we had to rewrite various parts in order to fully comply with the standard C++ since XL compiler is more rigorous about the ISO than the other compilers.

In the case of Blue Gene/P and Blue Gene/Q, the porting was far less laborious since it was based on previous work on Sp6. In this case, it focused on improving the performance as detailed in section 7.2.2 and in the previous section 6.2.

7.2 Stokes framework

We have implemented two Stokes solvers: the first one was based on the original framework already present in LifeV. In 7.2.1 we provide a brief introduction to motivate why we shifted to a different paradigm and then we present this new framework in 7.2.2.

7.2.1 Stokes Solver

Firstly we have implemented a Stokes solver based on the pre-existent `OseenSolver` class in Lifev, but we have decided to abandon the first implementation after the preliminary tests on Sp6 machine (see 5.2.1) in favour of a complete rewritten solver based on a different paradigm. The first implementation was based on the idea of a generic `Solver` object: a user should be able to run a solver without knowing almost anything about the operators assembled and used.

It is a sort of black box that performs the assembling of the entire matrix. For this reason, this approach is easier for elementary simulations and it is also user friendly, but it is less effective for more complicated problems since it is less flexible. This approach does not allow to optimize memory occupation e CPU usage easily. This is due to the generality of a big solver class that it has to contain several data structures not always used by all applications.

The choice to abandon the first implementation was motivated also by another reason: a source code is generally used and modified by several people and our aim is to make it available to a wide audience. Hence it is very important that the source code should be easy to be used, understood and modified.

```
1 OseenSolver< mesh_Type > fluid ( fluidDataPtr,
                                   *uFESpacePtr,
3                                  *pFESpacePtr,
                                    commPtr,
5                                   nLagranMult );
  fluid.setUp ( dataFile );
7 fluid.buildSystem ( ); // build the matrices
  fluid.updateSystem ( rhs ); //set the rhs
9 fluid.iterate ( bcH ); //solve the linear system
```

Code 7.1: *example of the old Stokes class. The user have to specify only few information.*

7.2.2 Stokes Assembler

After we have performed the tests of Sp6 machine described in section 5.2.1, we decided to write a new framework to reduce memory footprint:

the idea was to create a lighter class based on the `Assembler` paradigm, containing only the operators needed by the Stokes problem and avoiding unnecessary data structures. In box 7.2 is shown the declaration of the class with the attributes. This class permits to the user to choose from outside the quadrature rules, the finite elements and the methods to build the various operators needed by the Stokes problem.

```
template< typename meshT, typename matrixT, typename vectorT>
class StokesAssembler
{public:
// Empty Constructor
StokesAssembler();
// Destructor
virtual ~StokesAssembler() {};

// the method to set the FE Spaces
void setup ( uFESpace, pFESpace );
// method to set the stiffness operator
void addViscousStress ( matrix,  viscosity);
// method to set the stiffness operator
void addGradPressure ( matrix );
// method to set the divergence operator
void addDivergence ( matrix, coefficient = 1.);
// method to set the pressure mass
addPressureMass ( matrix, coefficient);
//method to add RHS
addMassRhs(vectoRhs, function_T fun, vector coef, time);
private:
    // Velocity FE space
    fespacePtr M_uFESpace;
    // Pressure FE space
    fespacePtr M_pFESpace;

    // Current FE
    currentFEPtr M_viscousCFE;
    currentFEPtr M_gradPressureUCFE;
    currentFEPtr M_gradPressurePCFE;
    currentFEPtr M_divergenceUCFE;
    currentFEPtr M_divergencePCFE;
    currentFEPtr M_massPressureCFE;

    // CurrentFE for the mass rhs
    currentFEPtr M_massRhsCFE;

    // Local matrix
    localMatrixPtr M_localViscous;
    localMatrixPtr M_localGradPressure;
    localMatrixPtr M_localDivergence;
    localMatrixPtr M_localMassPressure;
    localVectorPtr M_localMassRhs;};
```

Code 7.2: *the structure of the stokes class.*

In this way, the user has more freedom to assemble the operators that are effectively needed. The methods that builds the various operators require smart pointers as input for matrices and vectors so they avoid useless copy

of data structures while being memory safe. Every private attribute is actually a smart pointer, so if it is not instantiated it occupies almost nothing. The class `StokesAssembler` relies on other data structures and functions to assemble the operators, for instance we report the computation of the pressure mass matrix in code 7.3. We loop the mesh elements updating the values of the basis functions in the quadrature nodes on the given element (line 9), hence we assemble the local mass matrix (line 15) and finally we assemble the global matrix (line 21).

```
1  addPressureMass ( matrix, coefficient)
   {     // Some constants
3      const UInt nbElements (M_pFESpace->mesh()->numElements() );
       const UInt fieldDim (M_pFESpace->fieldDim() );
5
       // Loop over the elements
7      for (UInt iterElement (0); iterElement < nbElements; ++iterElement)
       {
9
           M_massPressureCFE->update ( M_pFESpace->mesh()->element (
       iterElement), UPDATE_PHI | UPDATE_WDET );
11
           // Clean the local matrix
13         M_localMassPressure->zero();

15         // local stiffness
           AssemblyElemental::mass (*M_localMassPressure, *M_massPressureCFE,
        coefficient, fieldDim);
17
           // Assembly
19         for (UInt iFieldDim (0); iFieldDim < fieldDim; ++iFieldDim)
           {
21             assembleMatrix ( matrix,
                                *M_localMassPressure,
23                              *M_massPressureCFE,
                                *M_massPressureCFE,
25                              M_pFESpace->dof(),
                                M_pFESpace->dof(),
27                              iFieldDim, iFieldDim,
                                offsetUp, offsetLeft);
29         }
       }
31 }
```

Code 7.3: *the method to assemble the mass pressure matrix.*

Moreover the object-oriented paradigm helps to keep the memory footprint low because every-time the object is deleted its memory is freed. Nevertheless, it can happen in the high performance machines that the garbage collector does not free the memory during the computation (like on the Sp6 machine). Thus a trick, that works well, to free the memory is to enclose parts of the code in a separate scope to force the memory to be cleared.

Data structures, pointed by the smart pointer, survives outside the scope. Indeed we can selectively chose what leave in memory and what not, there is a small example in the box 7.4.

```
//these are smart pointers
matrixPtr stokesMatrix;
solverPtr stokesSolver;
vectorPtr stokesSolution;
{
   // build the various operators needed
   stokesAssembler.addDivergence ( *stokesMatrix, 1. );
   stokesSolver->setOperator ( stokesMatrix );
}
//at this point survive only the stokesSolver
//object and all data structures linked to it

//solve the linear system
stokesSolver->solve ( stokesSolution );
```

Code 7.4: *example of scope reduction.*

7.2.3 Mesh movement

Since a free surface boundary condition is applied on the top of the computational domain, we need to move the mesh along the free boundary.

As reported in the example in the box 7.5, we firstly instanced in a new vector (line 2) the velocity field that has been previously interpolated to $\mathbb{P}^1$ elements (line 4).

```
velocity->subset(*solution);
vectorPtr_Type disp( new vector_Type( u_FESpace->map(), Repeated ) );
*disp = mFESpace->feToFEInterpolate( *uFESpace, *velocity );
transformer.moveMeshSelected( *disp, ( p_FESpace->dof().numTotalDof() +
    c_FESpaceP0->dof().numTotalDof() ) * 3 );

moveMeshSelected( const VECTOR& disp, UInt dim )
{
    points_Type points_Type;
    points_Type& pointList( M_mesh.pointList );
    for ( unsigned int i = 0; i < M_mesh.pointList.size(); ++i )
    {
        for ( UInt j = 0; j < nDimensions; ++j )
        {
            int id = pointList[i].id();
            Int marker = pointList[i].marker();

            switch (marker)
            {
                case 3000:
        pointList[ i ].coordinate( j ) = M_pointList[ i ].coordinate( j
    ) + disp[ j * dim + id ];
                    break;
                case 0:
        pointList[ i ].coordinate( j ) = M_pointList[ i ].coordinate( j
    ) + disp[ j * dim + id ];
                    break;
            }
        }}}
```

Code 7.5: *the algorithm to move the mesh.*

Then we call a function that moves selectively the node of the mesh according to the velocity field and the boundary flag that allow to identify the nodes to be moved. Since the movement is often small we do not need to regularize the mesh after moved.

7.3 Tracking framework

In this section we briefly describe the implementation choices of the tracker algorithm in paragraph 3.3.3.

Let us consider the code in 7.6, the algorithm is the following:

- partition the mesh and build the overlapped parts (line 2),

- create the connectivity map for the nodes (line 4),
- create the scalar finite element spaces to track the compositions variables (line 6),
- create the data structures to handle the ghost maps (line 8),
- create the data structures containing the mesh and the time informations (line 10),
- create the tracking solver (line 12),
- set up the initial value of the composition (line 16),
- set the velocity field for the tracking solver (line 20),
- solve Stokes problem and export the solution (line 24),
- set up the step count for the temporal cycle (line 28),
- set again the velocity field for the tracking solver (line 30),
- set initial time, end time and time step (line 34),
- check if the time step is consistent (line 38),
- advance the solution to the next time step and advance the current time of Δt (line 42),
- get new values of the viscosity and density (line 46),
- solve Stokes problem and iterate.

```
1  //create the overlapped maps
2  meshPart.setBuildOverlappingPartitions( true );
3
4  // create node neighbourhood informations for Track solver
5  createTrackNodeNeighbors ( *localMeshPtr, *Members->comm );
6
7  // Scalar finite element space of the composition variable.
8  feSpacePtr_Type c_FESpaceP1( new feSpace_Type( ) );
9
10 GhostHandler<mesh_Type> ghostP1( );
11
12 // Create the data file for the interface tracking solver
13 TrackData<mesh_Type> trackData;
14
15 // Instantiation of the track solver.
16 trackSolver_Type trackSolver ( );
17
18 // set up the initial value of the composition
19 trackSolver.computeInitialComposition();
20
```

```
// set the velocity field for the Track solver
vectorPtr_Type compositionVelocity ( new vector_Type ( u_FESpaceP1->map(),
     Repeated ) );
*compositionVelocity = u_FESpaceP1->feToFEInterpolate( *u_FESpace, *
    velocity, Repeated );
trackSolver.setVelocity( compositionVelocity );
//solve Stokes - not shown
// export the composition
    UInt stepCount( 0 );
// time loop of the hyperbolic eq. that track the physical quantities
    for ( Real currentTime( initialTime ); currentTime < endTime -
    tolerance; stepCount++ )
    {
      velocity->subset( *stokesSolution );
      *compositionVelocity = u_FESpaceP1->feToFEInterpolate( *u_FESpace, *
    velocity, Repeated );
      trackSolver.setVelocity( compositionVelocity );

      // Track solver inner loop
      trackData.dataTime()->setInitialTime( currentTime );
      trackData.dataTime()->setEndTime( currentTime + timeStep );
      trackData.dataTime()->setTimeStep( timeStep );

      // Check if the time step is consistent, i.e. if innerTimeStep +
    currentTime < endTime.
      if ( trackData.dataTime()->isLastTimeStep() )
      { trackData.dataTime()->setTimeStep( trackData.dataTime()->leftTime
    ());}
      // advance the solution to the next time step
      Real timeComputed = trackSolver.step();
     // Advance the current time of \Delta t.
     trackData.dataTime()->updateTime( timeComputed );
      // get new values of the viscosity and density
      physProp->update( composition );
      viscosity = physProp->getProperty( viscosityName );
      density = physProp->getProperty( densityName );
       // divide with timestep
      *density /= timeStep;

      //solve Stokes - not shown

      }
//end of computation
```

Code 7.6: *the structure of track algorithm.*

CHAPTER 8

Applicative example

In this chapter we describe an application of the developed framework based on the benchmark presented in [45].

8.1 Sedimentary basin simulations

The simulation that is described in this section is performed on the domain visualized in Figure 8.1. We consider three different layers of rock divided by non-horizontal planes. This fact induces the formation of the Rayleigh-Taylor instability that is purely generated from the gravitational forces, since the initial condition for the velocity is set equal to zero in each point of the domain. The physical properties of the strata are listed in Tab. 8.1. The domain simulates a section of the upper part of the Earth crust and its dimensions are $3 \times 1 \times 5\ km$. The domain is discretized with a three-dimensional tetrahedral mesh with about 25 million DOFs. The boundary conditions that are imposed in the model are the following: on Γ_B, the bottom of the domain, the velocity is set to zero, on Γ_S and Γ_L the condition for the velocity is $\boldsymbol{u} \cdot \boldsymbol{n} = 0$, so there is no velocity in the normal direction. In fact, the upper part Γ_S behaves like a free surface, but since the configuration considered in this chapter does not take into account the

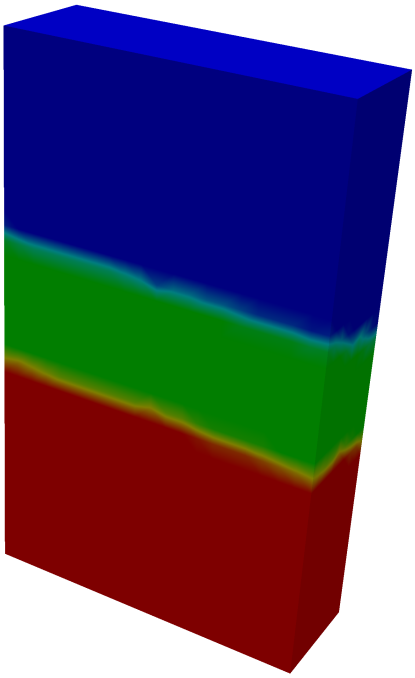

Figure 8.1: *A test section of the Earth crust. The dimensions of the domain are* $3 \times 1 \times 5$ *km.*

	Physical properties	
	density (kg/m^3)	viscosity (Pa · s)
upper layer	3.0	1.1e14
middle layer	2.0	6.3e13
bottom layer	0.1	3.2e13

Table 8.1: *Physical properties of the three layers.*

compaction of the layers, there is no movement of the grid, so Γ_S does not evolve in time.

The time evolution of the test is depicted in Figure 8.2. The simulated time frame is 5 millions of years (Ma). The time step is set to 0.1 Ma for the implicit Stokes solver, while the tracking solver performs a variable number of sub-steps in order to keep the Courant number acceptable. In any case, the number of sub-steps is never greater than 40. Only half of the domain is displayed in the figures in order to show what happens inside the basin. The lower stratum, that is the lighter one, tends to move up, pushing the other two layers down. The classic mushroom-shaped interface develops during the rising of the lower layer, until the velocity decreases when the

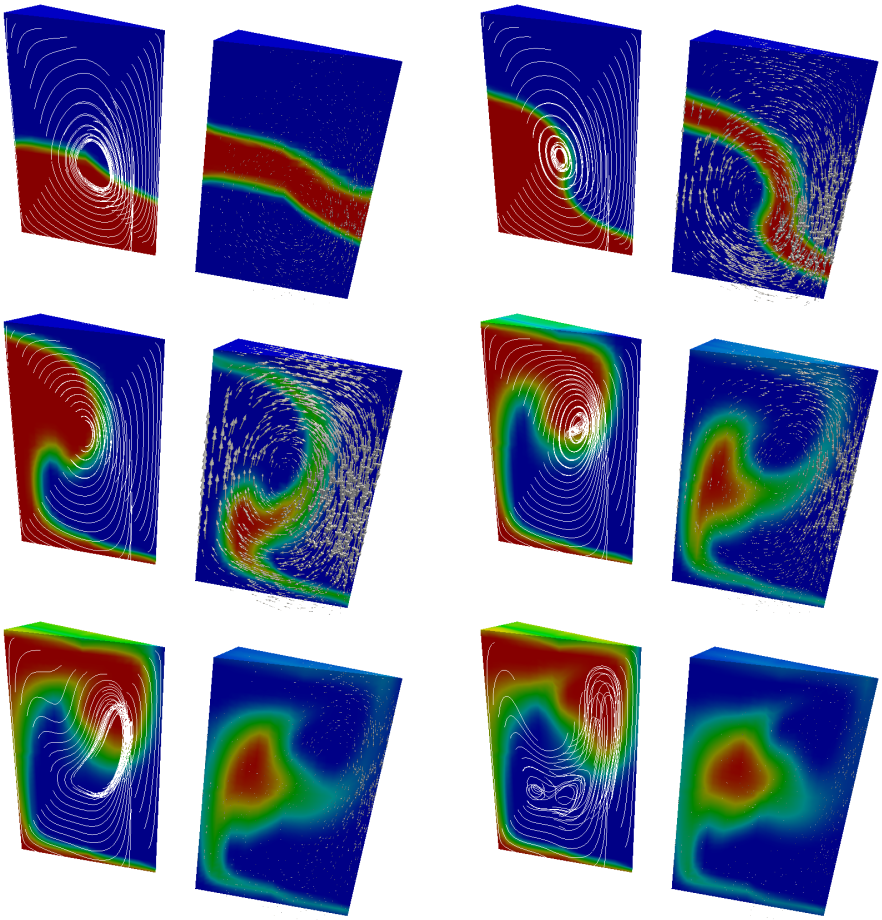

Figure 8.2: *Time evolution of the test. In each picture, the bottom layer with streamlines (left) and the middle layer with the velocity field (right) are displayed. From the top left to the bottom right the times are:* 0.8 *Ma,* 1.6 *Ma,* 2.4 *Ma,* 3.2 *Ma,* 4.0 *Ma,* 4.8 *Ma.*

lighter stratum sets above the others.

The discretized version of the test has about 25 millions degrees of freedom (DOFs). The Stokes system matrix is solved with a Flexible GMRES method (see paragraph 6.4). Different preconditioners have been considered: while algebraic preconditioners of the incomplete factorization family (from the IFPACK package in Trilinos) show good performance on serial runs, we have found that they scale poorly in parallel. Multi-grid preconditioners, such as the ones implemented in the ML library from Trilinos, show optimal scaling with the number of parallel processes. In particular, the multi-grid preconditioner has been set up to use a three-level scheme,

with a Smoothed Aggregation approach to create the prolongation and restriction operators. On the coarsest level a direct solver from the SuperLUdist [29] package was adopted, while on the finer levels the smoother is a Gauß-Seidel method, for more information see paragraph 6.2.1.

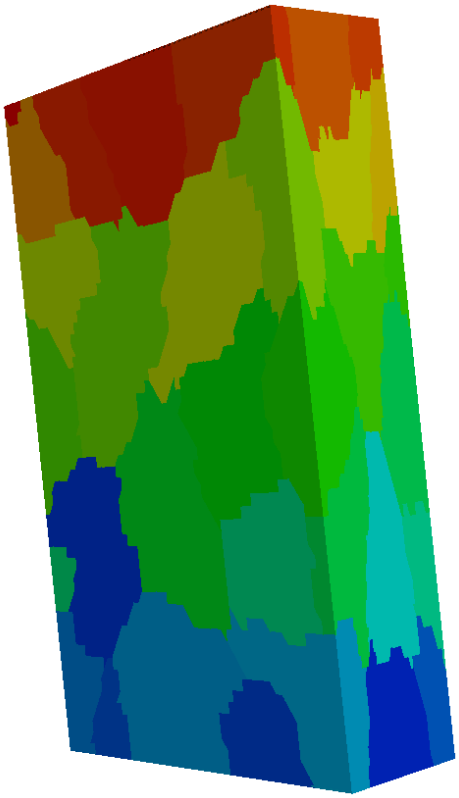

Figure 8.3: *Mesh partitioned among the processors.*

Even if the preconditioning techniques is optimal from the point of view of the dimension of the matrix and the number of parallel processes [10], the condition number of the system could remain strongly influenced by the ratio of the viscosity values, but not in this test-case. In particular, for the test-case shown above, the condition number is practically proportional to the viscosity ratio, see chapter 6.

The test have been performed with a growing number of parallel processes on the SGI Altix from CINES. Figure 8.4 shows the strong scalability performance of the solver. The test was performed running one processor per node to obtain the maximum memory available in the node.

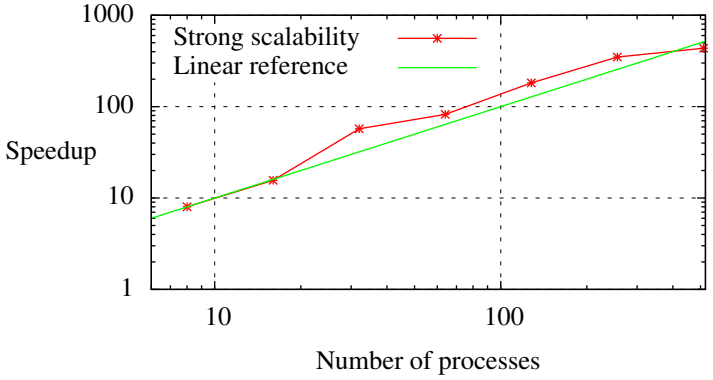

Figure 8.4: *The strong scalability with a linear reference line of the test-case on SGI Altix ICE 8200.*

8.1.1 Conclusion

Our code enables the simulation of the test-case [45] on HPC machines with more than 25 MDOFs. Now the main difficulty to scale more on machine like Blue Gene/Q is to find bigger meshes and to reduce the memory requirements.

8.1.2 Lithostatic test-case

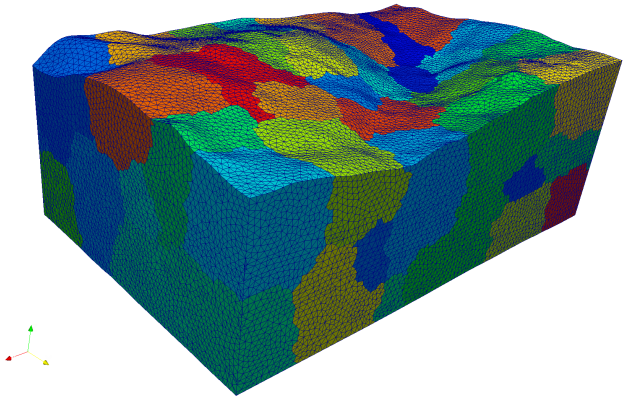

Figure 8.5: *Mesh partitioned among the processors in the lithostatic test-case.*

This is another test-case still based on [45], where we simulate only the lithostatic pressure (the Stokes problem). In Figure 8.5 it is presented the physical domain and the mesh partitions.

The continuous and the discrete problems are still the same, as well the parameters used to solve the Stokes problem in the previous test-case and discussed in the previous chapters. In this test we investigate two aspects: the behaviour of our code with a high number of DOFs (the maximum that we can achieve with biggest mesh that we can have) and the behaviour of the preconditioner with highly varying viscosity (layers with a ratio of 10^6).

The test was 48 MDOFs big over 128 processors and it was performed on the Blue Gene/Q at CINECA, Italy. The test was performed with 32, 64 and 128 cores.

In Figure 8.6 it is presented the pressure field of the solution.

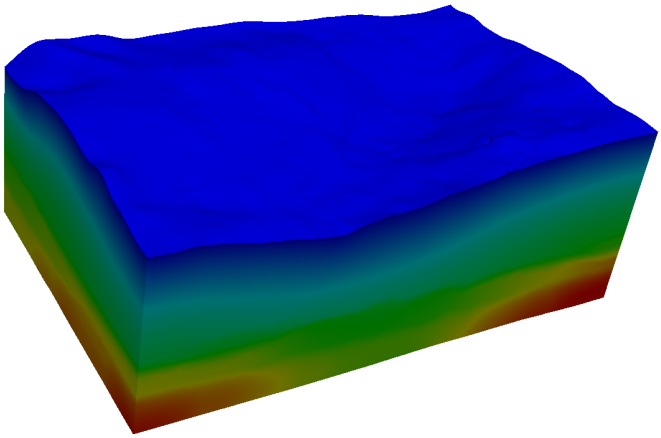

Figure 8.6: *Pressure field of the lithostatic test-case.*

CHAPTER 9

Conclusion

9.1 Conclusion

This thesis deals with the porting on HPC machines of a Finite Element code, the parallel preconditioning of a Stokes problem with varying viscosity and the parallel implementation of the algorithm introduced in [30, 46]. We introduce a suitable parallel preconditioner for Stokes problems with varying viscosity. We investigate the parallel performance of the Stokes solver implemented on various HPC machines.

9.1.1 Difficulties

The biggest difficulties arose during the code porting to AIX Operating System and to XL compiler, in fact we took around one year to port the code. Also debugging in a HPC programming environment with hundreds of MPI tasks was quite difficult and it slowed down the code development.

9.1.2 Original Work and goals reached

We have developed a novel tracking solver and a Stokes solver that can run on High Performance Machines of the last generation. The approach

that we have adopted is based on coarse grain parallelism. More precisely, the numerical techniques adopted is domain decomposition. Further improvements on modern hybrid architectures could be reached by adopting a shared memory protocol (SMP) type of parallelism at the sub-domain level.

Given the hyperbolic nature of the equation for the tracking of the layers, we have the need of collecting data from neighbour sub-domain. To collect this information we have introduced a dedicated parallel data structure on overlapping sub-domains that allows to reconstruct the numerical fluxes across the sub-domain interfaces in an efficient way.

The accuracy and performance of the method has been assessed in a surrogate test. In particular, given the dimensions of realistic sized sedimentary basins, we focused on the correct and optimal configuration of the preconditioner of the Stokes system, that happens to be the most expensive part of the computation.

We have tested several options, including preconditioning tools made available by state of the art parallel linear algebra packages. By the selection of the appropriate preconditioning we were able to reach good scalability on more than one-thousand processors. The code has also been applied to realistic industrial test-cases.

9.1.3 Future work

Future work will include the introduction of the compaction of the rocks, which requires to introduce a suitable compaction model and modify the grid to account for new sedimented layers. The coupling with subsurface fluid models is also an interesting extension. In the future we will also investigate other stable polynomial couples and stabilization for low order polynomial.

List of Figures

List of Tables

Listings

Bibliography

[1] VV. AA. IBM Blue Gene/Q supercomputer delivers petascale computing for high-performance computing applications. Technical Report 112-028, IBM United States Hardware, 2012.

[2] Michele Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182:418–477, 2002.

[3] Michele Benzi, Golub, and Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005.

[4] B. Biju-Duval. *Sedimentary Geology*. Edition Technip, 2002.

[5] Carsten Burstedde et al. Large-scale adaptive mantle convection simulations. *Geophysical Journal International*, 2013.

[6] M. A. Saunders C. C. Paige. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[7] G. Schubert D.L. Turcotte. *Geodynamics*. Cambridge Univ Press, 2002.

[8] I. S. Duff, A. M. Erisman, and Reid J. K. *Direct Methods for Sparse Matrices*. Clarendon, 1986.

[9] École Polytechnique Fédérale de Lausanne (CMCS), CH; Politecnico di Milano (MOX), ITA; INRIA (REO, ESTIME), FR, and Emory University (Math&CS), GA US. *LifeV User Manual*, 2010.

[10] H.C. Elman, D.J. Silvester, and A.J. Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics*. Oxford University Press, USA, 2005.

[11] A. Ern and J.L. Guermond. *Theory and practice of finite elements*, volume 159. Springer Verlag, 2004.

[12] F. Salvini F. Storti and K. McClay. Fault-related folding in sandbox analogue models of thrust wedges. *Journal of Structural Geology*, 19:583–602, 1997.

[13] M. Fortin and R.G. Lowinski. Augmented Lagrangian Methods: Application to the solution of Boundary-Value Problems. *Stud. Math. Appl.*, 15:743, 1983.

[14] J. M. Odersitzki G. L. G. S Leijpen, H. A. Van Der Vorst. Effects of rounding errors in determining approximate solutions in Krylov solvers for symmetric indefinite linear systems. *SIAM J. Matrix Anal. Appl.*, 22:726–751, 2000.

[15] Wolf Geimer and other. The scalasca performance toolset architecture. *Journal Concurrency and Computation*, 22(6):702–719, 2010.

[16] A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, 1981.

[17] Vivette Girault and Pierre-Arnaud Raviart. *Finite Element Methods for Navier-Stokes Equations*. Springer, 1980.

[18] A. Greenbaum, V. Pták, and Z. Strakoš. Any nonincreasing convergence curve is possible for GMRES. *J. Matrix Anal. Appl.*, 17:465, 1996.

[19] P. P. Grinevich and M. A. Olshanskii. An iterative method for the Stokes type problem with variable viscosity. *J. Sci. Comput.*, 31(5):3959–3978, 2009.

[20] M.A Heroux. *AztecOO user's guide*, 2004.

[21] M.A. Heroux, R.A. Bartlett, V.E. Howle, R.J. Hoekstra, J.J. Hu, T.G. Kolda, R.B. Lehoucq, K.R. Long, R.P. Pawlowski, E.T. Phipps, et al. An overview of the Trilinos project. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):397–423, 2005.

[22] B. F. Ischer. *Polynomial Based Iteration Methods for Symmetric Linear Systems*. Wiley-Teubner, 1996.

[23] A. Ismail-Zadeh, I. Tsepelev, C. Talbot, and A. Korotkii. Three-dimensional forward and backward modelling of diapirism: numerical approach and its applicability to the evolution of salt structures in the Pricaspian basin. *Tectonophysics*, 387(1):81–103, 2004.

[24] A. Ismail-Zadeh, IA Tsepelev, C. Talbot, and P. Oster. A numerical method and parallel algorithm for three-dimensional modeling of salt diapirism. *Problems in dynamics and seismicity of the earth*, 31:62–76, 2000.

[25] A. Jameson, W. Schmidt, E. Turkel, et al. Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes. *AIAA paper*, 81(125):9, 1981.

[26] George Karypis and Vipin Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[27] D. E. Keyes. How scalable is domain decomposition in practice? In *Proceedings of the 11th Intl. Conf. on Domain Decomposition Methods*, pages 286–297, 1998.

[28] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand.*, 45:255–282, 1950.

[29] X.S. Li and J.W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software (TOMS)*, 29(2):110–140, 2003.

[30] M. Longoni, A.C.I. Malossi, and A. Villa. A robust and efficient conservative technique for simulating three-dimensional sedimentary basins dynamics. *Computers & Fluids*, 39(10):1964–1976, 2010.

[31] P. Massimi. *Perspective in computational salt tectonics*. PhD thesis, Politecnico di Milano, 2006.

[32] P. Massimi, A. Quarteroni, and G. Scrofani. An adaptive finite element method for modeling salt diapirism. *Mathematical Models and Methods in Applied Sciences*, 16(4):587–614, 2006.

[33] D. A. Mousseau, V. A. Knoll and W. J. Rider. Physics-based preconditioning and the Newton–Krylov method for non-equilibrium radiation diffusion. *J. Comput. Phys.*, 160:743, 2000.

[34] A.M. Quarteroni and A. Valli. *Numerical approximation of partial differential equations*, volume 23. Springer Verlag, 2008.

[35] N. M. Nachtigal R. W. F. Reund. A new Krylov–subspace method for symmetric indefinite linear systems. In *Proceedings of 14th IMACS World Congress on Computational and Applied Mathematics*, pages 1253–1256. IMACS, 1994.

[36] N. M. Nachtigal R. W. Freund. Software for simplified Lanczos and QMR algorithms. *Appl. Numer. Math.*, 19:319–341, 1995.

[37] Geenen Rehman et al. On iterative methods for the incompressible Stokes problem. *Int. J. Numer. Meth. Fluids*, 65:1180–1200, 2011.

[38] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[39] M. Sala and M. Heroux. Robust algebraic preconditioners with IFPACK 3.0. Technical Report SAND-0662, Sandia National Laboratories, 2005.

[40] M. Sala and M. Heroux. Trilinos tutorial. Technical Report SAND2004-2189, Sandia National Laboratories, 2010.

[41] B.C. Schreiber and K.J. Hsü. Evaporites. *Developments in Petroleum Geology*, 2:87–138, 1980.

[42] G. Scrofani. *Numerical Basins Modeling and Tectonics*. PhD thesis, Politecnico di Milano, 2007.

[43] S. Shende and A. D. Malony. TAU: The TAU parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[44] R.W. Smith. AUSM(Ale): a geometrical Conservative Arbitrary Lagrangian-Eulerian Flux Splitting Scheme. *J. Comp. Physics*, 150:268–286, 1999.

[45] A. Villa. *Three dimensional geophysical modeling: from physics to numerical simulation*. PhD thesis, Politecnico di Milano, 2010.

[46] A. Villa and L. Formaggia. Implicit tracking for multi-fluid simulations. *Journal of Computational Physics*, 229(16):5788–5802, 2010.