POLITECNICO DI MILANO

Facoltà di Ingegneria Industriale

Corso di Laurea in Ingegneria Aeronautica

A GPU parallelized two fields full potential formulation for real gases

Relatore: Prof. Paolo MANTEGAZZA

Co-Relatore: Prof. Marco MORANDINI

Tesi di Laurea di:

Marco FAVALE Matr. 780340

Andrea GADDA Matr. 780849

Anno Accademico 2012-2013

Acknowledgments

We really wish to thank our supervisor Professor Paolo Mantegazza for his encouragement and support. The time and efforts He dedicated to this project is really precious to us.

Our gratitude goes to Professor Marco Morandini for his moral and technical support. Through the last year of our studies He contributed to a great part of our current technical skills.

Special thanks are due to Andrea Parrinello for the experience he brought in our work and for supplying comparative numerical results.

We wish to express our sense of gratitude to all of them for their outstanding helpfulness.

Contents

Abstract

In recent years, the peculiar phenomena of dense gas dynamics have raised great interest since they could be exploited to improve the performances and efficiency of some aerodynamic devices that work in transonic or supersonic regime. Bethe–Zel'dovich–Thompson (BZT) fluids exhibit non classical behaviors such as expansion shocks or compressive fans when their thermodynamic state is near the critical point. These unusual waves have been numerically investigated by means of complicated and computationally expensive models such as Euler or Navier–Stokes equations.

In the present work a simplified model to study these fluids is proposed and implemented. The formulation is based on an independent two field full potential approach that uses as unknowns the density and the velocity potential. The problem is solved using a cell-centered finite volume (FV) discretization in space and explicit time stepping integration schemes. These decisions were taken in order to accelerate the execution using GPUs programmed with the OpenCL language. General purpose GPU computing (GPGPU) allows in fact remarkable speed-ups in simulations, as presented as a result of this work. The developed software is furthermore capable to simulate unsteady flows around moving bodies using an arbitrary Lagrangian Eulerian (ALE) formulation. Some validation results in ideal gas and dense gas regime are presented in order to show the validity of such formulation.

Keywords: CFD, potential flows, dense gas dynamics, BZT fluids, OpenCL, ALE formulation

Sommario

Negli ultimi anni, i fenomeni tipici della gasdinamica dei gas densi hanno suscitato un notevole interesse dato che potrebbero essere sfruttati in certi componenti aerodinamici al fine di ottenere un notevole miglioramento delle loro prestazioni e della loro efficienza. I fluidi di Bethe–Zel'dovich–Thompson (BZT) esibiscono comportamenti non classici come urti di espansione o ventagli di compressione quando il loro stato termodinamico è vicino alle condizioni critiche. Queste onde insolite sono state studiate numericamente attraverso l'uso di complicati modelli matematici come le equazioni di Eulero o di Navier–Stokes che richiedono notevoli oneri computazionali per poter ottenere una soluzione.

Nel presente lavoro viene analizzato ed implementato un modello semplificato per studiare tali fluidi. La formulazione utilizzata si basa su un approccio a potenziale a due campi, che usa come incognite la densità e il potenziale cinetico. Il problema è risolto tramite una discretizzazione in spazio a volumi finiti (FV) a celle centrate e con l'utilizzo di schemi espliciti per l'avanzamento in tempo. Tale decisione è stata presa al fine di poter accelerare la simulazione utilizzando la potenza computazionale fornita dalle moderne schede video (GPU) che sono state programmate con il linguaggio OpenCL. L'utilizzo di schede video permette infatti di ottenere notevoli riduzioni nei tempi di calcolo, come presentato nei risultati di questo lavoro. Il solutore sviluppato è inoltre in grado di simulare correnti instazionarie attorno a corpi che si muovono nel fluido attraverso l'utilizzo di una formulazione arbitrariamente lagrangiana euleriana (ALE). Alcuni risultati di validazione sono presentati per i casi di gas ideale e gas denso.

Parole Chiave: CFD, correnti a potenziale, gas densi, fluidi BZT, OpenCL, formulazione ALE

Chapter 1

Introduction

Now more then ever, the CFD scientific community attention is focused on high accuracy models, such as the Euler and the Full Navier Stokes equations. This trend is supported by the ever increasing computational power of modern computers and supercomputers. However a direct numerical simulation of viscous flows at an engineering relevant Reynolds number is not still possible. Turbulence modeling (RANS, LES, DES etc) is thus a cumbersome necessity that requires a lot of calibration to obtain acceptable results, making experimental studies performed in wind tunnels still a necessary and an essential step in the design of a well-performing aerodynamic component. However, when no fluid separation are expected, less complicated models can be used, such as the Euler equations or the full-potential formulation. In view of a primary engineering evaluation of fluid behavior, these formulations can give very satisfactory results with a much lower computational effort.

The use of the full potential model to solve transonic flows around lifting bodies was firstly pursued at the beginning of the 1970s by Murman and Cole [1]. They solved the Transonic Small Disturbance (TSD) equation with centered finite differences in subsonic regions and backward differences in supersonic regions in order to ensure numerical stability. In 1975, to better satisfy jump conditions, Jameson introduced in [2] a conservative scheme applying some numerical viscosity in supersonic regions and together with Caughley [3] performed the first full potential computations through the use of the finite volume method in 1977. However it was only in 1997 that Neel [4] claimed the first application of the finite volume method to unstructured meshes, adopting a cell centered scheme. The state of the art of full potential flow solutions is represented by the work of Parrinello [5], that introduced in a conservative finite volume method the transport of entropy in thin layers surrounding bodies. This allows to capture shocks positions at the same location as the Euler equations do.

Low cost simulations are recently coming back on the scene due to modern optimization design techniques (such as neural or genetic algorithms) that require many evaluations of the solution in slightly different configurations. Although it is possible to perform some of these simulations with accurate and expensive models, it is absolutely unconceivable to start the optimization with such a method. In the first phases of these processes it is in fact necessary to reduce as son as possible the region in the parameter space where the optimization takes place. Therefore the use of a low cost model is mandatory, leaving high accuracy methods for the last optimization iterations only. Hence, the use of full potential flow solvers has been recently reevaluated since it allows to compute subsonic and transonic solutions with an acceptable quality and with much lower computational requirements than those of an Euler model.

The necessity to perform many computation in a reasonable simulation time, brought the use of parallel computing to be a standard approach in the last decades. Generally, parallel softwares run on CPU clusters where the workload is distributed among different processors. These machines offer a great improvement in performances with respect to a personal computer but are very expensive to buy and to operate. In the last years the parallel computing community focused his attention on a device that is specifically designed to accelerate graphic computations in computer gaming: the Graphic Process Unit (GPU). This hardware component, being made of many cores, is in fact capable to perform the same operation on a great amount of data, as requested in graphic elaborations. This feature is essentially what is needed in many algorithms, and can be efficiently exploited to perform numerical computations that exhibit high data parallelism. GPU programming has rapidly evolved from its primordial state, but it still requires some low level considerations to write a program that runs on a graphic card. Therefore, solution algorithms must be explicitly designed to be executable on a GPU and to take advantage of its features. The benefits to use a GPU to perform data parallel computations relies in the fact that a single graphic card has generally a greater computing power than a CPU of the same price level. Moreover GPUs generally show a higher performance per Watt ratio than CPUs, leading to lower operational costs and heat generation.

Earlier GPGPU approaches were born in the first years after 2000 driven by the boost of performance in numerical simulation provided by GPU, thanks to its highly parallelized hardware architecture. Despite the actual programming models in GPU computing, the initial GPGPU techniques were based on mapping the numerical problem to graphical transformation of pixels and vertexes, as Hagen et al. [6]. These, however, were cumbersome strategies. Later, in 2007, NVIDIA released the CUDA SDK (Software De-

velopment Kit), bringing the GPU programming to a more accessible level. Later, after its formation in 2008, the Khronos Group released the OpenCL 1.0 specification based on which some vendors such as AMD and NVIDIA released their own implementation of this standard. Nowadays software accelerated by GPUs are mostly written with OpenCL and CUDA. The first attempts to use GPGPU in inviscid CFD were made with structured grids since they offer a more efficient access to the GPU memory as shown by Elsen et al. [7] and by Brandvik and Pullan [8]. Successively Corrigan et al. [9] tried an efficient implementation of inviscid and viscid solver on general unstructured grids facing a loss of performance due to a non-consecutive memory access pattern.

In the last years, a new research field caught the attention of a part of the fluid dynamics researchers. This discipline investigates particular phenomena that take place in a gas flow when its thermodynamic conditions are very close to the liquid-vapor critical point and it is therefore called dense gas dynamics. In this regime, molecularly complex fluids called Bethe-Zel'dovich-Thompson (BZT) fluids, exhibit non classical behaviors such as expansion shocks, mixed shock-fan waves and compression fans. Researchers are trying to exploit these peculiarities to improve performances in some turbomachinery applications, where the design of fundamental components such as turbine blades or nozzles cannot prescind from a gas model that takes into account these effects. To determine the optimal geometry of these components, the use of optimization algorithms is increasingly being pursued and a low cost method could then be a great advantage to speed up the first phases. This is a fundamental aspect during the development of Organic Rankine Cycle (ORC) engines where heavy fluids are used to generate power exploiting the typical effects of BZT gases to improve efficiency as shown by Guardone et al. [10] or Colonna et al. [11].

In this work an Ex*plicit full* P*otential* Re*al gas* S*olver* (ExPReS) has been written from scratch to run on GPUs with the use of the OpenCL programming language. This choice was adopted to make the solver executable on a wider selection of GPU models than that offered by the CUDA programming model. The use of full potential models to predict the stationary behavior of BZT gases was pursued in 1991 by Cramer and Tarketon [12] using the small–disturbance equation. In this thesis the use of the potential model in dense gas flows is extended to the completely unsteady and non linear problem by means of a finite volume discretization. ExPReS is meant to be a proof of concept to show that the full potential approach can be used not only in ideal gas regimes but even with more complicated gas models, with a significant reduction of simulation times with respect to high fidelity models such as Euler equations solvers, thanks also to the GPU acceleration.

1.1 Thesis overview

In chapter 2 after the presentation of the assumptions underlying the analyzed potential approach, the mass conservation equation and the Bernoulli theorem are introduced and the full potential formulation is obtained. The properties of the chosen gas model are considered in the formulation through the enthalpy function that appears in the Bernoulli equation. The Kutta condition is then discussed in order to fix the indetermination of the circulation around lifting bodies, typical of potential flows. Successively two different ways to solve the problem are presented: the 1-field approach, where the only unknown is the kinetic potential, and the more versatile 2–fields approach where the size of the problem is duplicated due to the addition of the density as second unknown. These two strategies are then compared to highlight the advantages and disadvantages of both of them.

In chapter 3 thermodynamics aspects needed in this work are presented. A general introduction to the axiomatic approach is given where the principal thermodynamic quantities are obtained. They are then specialized to the Polytropic Ideal Gas (PIG) and the polytropic van der Waals gas models during an isentropic transformation. The fundamental derivative of gas dynamics is then introduced. This allows to explain the typical phenomena that distinguish dense gas flows from the more common ideal gas flows.

Chapter 4 is devoted to parallel computing. The OpenMP approach to parallelize the solver is described since it was used in the multithreaded CPU version of ExPReS. General Purpose Graphic Processing Unit (GPGPU) computing is then introduced with its features and limitations. Finally the main concepts of the OpenCL programming language are described, focusing on those that are strictly related to the implementation of ExPReS.

Numerical discretization techniques used to approximate the 2–fields full potential formulation are outlined in chapter 5. After a general introduction to the Finite Volume Method (FVM) a comparison between the node-centered and cell-centered formulation is performed with a particular attention to the implementation aspects related to the GPGPU. The final choice of the cell-centered approach is justified with the use of two example algorithm. The space discretization is then analyzed and the cell-centered gradient reconstruction problem is investigated with different schemes. Numerical time integration schemes complete the final discretized form of the full potential problem. A von Neumann stability analysis is then performed in order to find the stability boundaries of such schemes. Boundary conditions are finally discussed.

In chapter 6, after an introduction on the structure of the source code of ExPReS, some subsonic and transonic ideal gas flows are presented to validate

the solver. Then the capabilities of the full potential formulation to simulate dense gas flows is tested. Comparisons are made with another potential solver and with results available so far in literature. Some unsteady solutions are then computed in order to test the accuracy of time transient solutions around oscillating bodies. The speedups achieved by using the multithreaded ad the GPU versions of ExPReS are finally presented and some comparisons with speedups obtained by other commercial softwares are made.

In appendix A after a disquisition on the Lagrangian and Eulerian points of view, the Arbitrary Lagrangian Eulerian (ALE) formulation is deduced for the 2–fields full potential problem.

In appendix B the complete expression of the fundamental derivate of gas dynamics is obtained for the two gas models considered in this work.

1.2 Free software

An important aspect of this work is that it is entirely done only with software that is freely available on the Internet. This document is written with LaTeX. The operating system used is Ubuntu 13.04 64-bit which comes with all the software packages needed for this work. The compiler used for the C source codes is GCC[13], that provides an OpenMP[14] implementation and has autovectorization capabilities. GPU programming in this work is based on the OpenCL API and OpenCL C language[15] provided by the NVIDIA SDK which is distributed with Ubuntu as an optional package. Debugging and profiling of the C/C++ code written for this work are accomplished using software such as GDB [16], GNU gprof [17], Valgrind [17] and for the GPU code gDEBugger [18]. The adopted program for the visualization of the results is ParaView [19]. All the graphics that appear on this document are created with Veusz [20]. Meshes are generated with gmsh [21].

These are only few examples of the free software used in this work. It is truly amazing how all these tools can be easily accessed with a Linux operating system.

Chapter 2

Potential CFD

Many models can be used to study the behavior of an aerodynamic flow. The earliest steps in Computational Fluid Dynamics (CFD) were made in the 1970s considering inviscid flows only. This was due to the extremely limited computational power available in those years. The first methods to compute the aerodynamic flows around lifting bodies were based on integral formulations discretized through the use of panel methods. The introduction of more complex models went hand in hand with the growth of computer performances, leading to the diffusion of finite volume and finite element methods to solve inviscid streams and low Reynolds viscous flows. Nowadays many sophisticated and computationally expensive models are in use to simulate the aerodynamics around arbitrary complicated bodies, but there's still a long way to go before obtaining solutions that faithfully reproduce the physics of highly detached flows. The computing of those solutions could in principle be faced through a Direct Numerical Simulation (DNS), but up to now the capabilities of the modern clusters still don't allow such computations for engineering interesting cases.

When considering attached flows around aerodynamic bodies as planes, wings or airfoils, viscosity effects can be neglected. This is especially true whenever the aim is to estimate structural loads in the preliminary project. During these first phases it is necessary to have some low cost methods to span all the parameters space where optimizations take place. To perform these preliminary computations it is therefore advantageous the use of a series of simplified models to simulate streams with a much lower computational effort than those for viscous flows. The simplest inviscid simulation can be performed with a panel method. This is a quite computationally cheap approach, but it is suitable only for subsonic or highly supersonic flows. To the present day, almost every commercial aircraft flies in the transonic regime, for which these methods are not accurate enough. A relatively efficient method

that allows transonic calculations is the full potential CFD, that still deals with inviscid flows, but can take into account compressibility effects up to weak shocks. This simpler approach generally allows a quite accurate calculation of lifting loads, while only some of the contributes to the drag force can be obtained. These contributes can be for example the induced or the wake drag. Obviously viscous effects are completely lost and it is therefore not possible to estimate viscous drag.

In this chapter general full potential flows are firstly introduced with all their funding hypothesis. Then two two possible formulations are presented. The first one is based only on the velocity potential; the second one uses also the fluid density as unknown. Finally the two approaches are compared and their pros and cons are discussed.

2.1 Full potential flows

A possible way to reduce the complexity of the model is to use the full potential formulation. Some simplifying hypothesis must be introduce. The first one is to consider the entropy uniform and constant all over the flow field.

$$s\left(\boldsymbol{r},\, t\right) = s_{\infty}. \tag{2.1}$$

This assumption is completely valid if no shock waves are present, but it is no more satisfied in the transonic and supersonic regime, where there is a finite jump of all thermodynamic variables (included entropy) across shocks. It should be noticed that in the transonic regime shocks are weak and therefore hypothesis (2.1) is still adoptable being the entropy jump negligible. As demonstrated by Landau and Lifshitz in [22] the variation of entropy across a weak shock is proportional to the cube of the jump in pressure

$$s_2 - s_1 \approx \frac{1}{12\, T_1} \left(\frac{\partial^2 v}{\partial P^2}\right)_s \Bigg|_1 \left(P_2 - P_1\right)^3, \tag{2.2}$$

where the subscript 1 and 2 indicates respectively the upstream and the downstream thermodynamic states.

The second funding assumption of the full potential model is that the flow is irrotational, and thus the velocity $\boldsymbol{u}$ can be expressed as the gradient of the velocity potential ϕ

$$\boldsymbol{u} = \boldsymbol{\nabla}\phi. \tag{2.3}$$

It is immediately highlighted that this relation gives an indetermination of any addictive constant on the velocity potential. In fact, the governing

equations of the full potential problem include only the derivatives of ϕ and therefore, if $\tilde{\phi}$ is a solution, then $\tilde{\phi} + C$ with C constant is a solution too. This offset is not a problem because only the derivatives of the velocity potential are necessary to compute the aerodynamic field. However this is not the only indetermination. In fact, in multiply connected domains, there is an indetermination on the circulation around every body. This problem is solved imposing the well known Kutta condition explained in section §2.1.1. On the contrary of (2.1), assumption (2.3) is not a problem in transonic flows where shocks are mainly normal to the streamlines. Consider in fact the Crocco's theorem, valid only in the stationary case, (2.4)

$$\boldsymbol{\nabla} h^t + \boldsymbol{\omega} \times \boldsymbol{u} = T \, \boldsymbol{\nabla} s, \tag{2.4}$$

where T is the fluid temperature, $\boldsymbol{\omega} = \boldsymbol{\nabla} \times \boldsymbol{u}$ is the vorticity and h^t is the total enthalpy. Since the entropy jump is uniform across a normal shock, it follows that $\boldsymbol{\nabla} s = 0$. Moreover being the asymptotic free stream omototalenthalpic, even $\boldsymbol{\nabla} h^t = 0$. Therefore, in case of normal shock, $\boldsymbol{\omega} \times \boldsymbol{u} = 0$. In the 2D case this means that no vorticity is generated i.e. $\boldsymbol{\omega} = 0$, while in the 3D case that product could annihilate also if $\boldsymbol{\omega} /\!\!/ \boldsymbol{u}$. It is therefore possible to have isentropic vortexes that travel along their axes.

To obtain the full potential problem it is necessary to introduce the mass and the momentum conservation equations. Adopting an Eulerian formulation, the unknown of the problem are the mass density $\rho = \rho(\boldsymbol{r}, t)$ and the velocity potential $\phi = \phi(\boldsymbol{r}, t)$.

The mass conservation principle states that the mass in a fixed control volume can vary only because there is a mass flux through its boundary

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_V \rho = - \oint_{\partial V} \rho \, \boldsymbol{u} \cdot \hat{\boldsymbol{n}}, \tag{2.5}$$

where the infinitesimal elements dV and dS are not written because their presence is deducible from the domain of integration [1].

Being the control volume fixed, the time derivative can be moved under the integral sign, and applying the divergence theorem to the right hand side of equation (2.5), the differential form of the mass conservation equation can be obtained

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \, \boldsymbol{u}) = 0. \tag{2.6}$$

[1] This is only a notational simplification and the reader should always keep in mind that those elements are present due to dimensional reasons, even if they are not explicitly written

The governing equation for the second unknown ϕ, can be deduced from the momentum conservation principle. In Eulerian formulation, the momentum in a fixed control volume can vary only due to momentum flux through the boundary of the volume and to the stress acting on it

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_V \rho\,\boldsymbol{u} = -\oint_{\partial V} \rho\,\boldsymbol{u}\,(\boldsymbol{u}\cdot\hat{\boldsymbol{n}}) + \oint_{\partial V}\boldsymbol{t}, \tag{2.7}$$

where $\boldsymbol{t}$ is the total stress on the boundary of the volume. In the hypothesis of inviscid flow, stresses are generated only by pressure actions, i.e. $\boldsymbol{t} = -P\,\hat{\boldsymbol{n}}$. Applying again the divergence theorem and substituting equation (2.6), the differential form of the momentum conservation equation is obtained

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\boldsymbol{\nabla})\,\boldsymbol{u} + \frac{\boldsymbol{\nabla} P}{\rho} = 0. \tag{2.8}$$

By applying the identity

$$(\boldsymbol{u}\cdot\boldsymbol{\nabla})\,\boldsymbol{u} = \frac{1}{2}\boldsymbol{\nabla}\,(\boldsymbol{u}\cdot\boldsymbol{u}) + \boldsymbol{\omega}\times\boldsymbol{u} \tag{2.9}$$

to equation (2.8) and recalling that the velocity field is irrotational, we have

$$\frac{\partial \boldsymbol{u}}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\,(\boldsymbol{u}\cdot\boldsymbol{u}) = -\frac{\boldsymbol{\nabla} P}{\rho}. \tag{2.10}$$

Equation (2.10) can be rewritten in term of velocity potential as

$$\frac{\partial\,\boldsymbol{\nabla}\phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\,(\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi) = -\frac{\boldsymbol{\nabla} P}{\rho} \tag{2.11}$$

or

$$\boldsymbol{\nabla}\left(\frac{\partial\phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi\right) = -\frac{\boldsymbol{\nabla} P}{\rho}. \tag{2.12}$$

The right hand side of equation (2.12), can be expressed as the gradient of enthalpy h. In fact

$$h = e + P\,v, \tag{2.13}$$

where e is the internal specific energy, and v is the specific volume.

Applying the gradient operator to both sides of equation (2.13), gives

$$\boldsymbol{\nabla} h = \boldsymbol{\nabla} e + P\,\boldsymbol{\nabla} v + v\,\boldsymbol{\nabla} P. \tag{2.14}$$

The gradient of energy is computed from $e(s, v)$, where the two independent variables s and v are expressed in an Eulerian point of view as $s = s(\boldsymbol{r}, t)$ and $v = v(\boldsymbol{r}, t)$

$$\boldsymbol{\nabla} e = \left(\frac{\partial e}{\partial s}\right)_v \boldsymbol{\nabla} s + \left(\frac{\partial e}{\partial v}\right)_s \boldsymbol{\nabla} v = T\,\boldsymbol{\nabla} s - P\,\boldsymbol{\nabla} v. \tag{2.15}$$

Substituting equation (2.15) in equation (2.14), the gradient of enthalpy results

$$\boldsymbol{\nabla} h = T\,\boldsymbol{\nabla} s + v\,\boldsymbol{\nabla} P. \tag{2.16}$$

In the isentropic, $\boldsymbol{\nabla} s = 0$ and therefore

$$\boldsymbol{\nabla} h_{is} \;=\; v\,\boldsymbol{\nabla} P_{is} \tag{2.17}$$

or

$$\boldsymbol{\nabla} h_{is} = \frac{\boldsymbol{\nabla} P_{is}}{\rho}. \tag{2.18}$$

where the subscript is has been introduced in order to highlight that P_{is} is obtained through an isentropic transformation.

Substituting equation (2.18) in equation (2.12), it is possible to obtain

$$\boldsymbol{\nabla}\left(\frac{\partial \phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi \cdot \boldsymbol{\nabla}\phi + h_{is}\right) = 0, \tag{2.19}$$

that integrated along a streamline departing from the asymptotic flow, gives the well known Bernoulli theorem

$$\frac{\partial \phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi \cdot \boldsymbol{\nabla}\phi + h_{is} = \frac{1}{2}V_\infty^2 + h_\infty. \tag{2.20}$$

Since the asymptotic state is uniform, this relation can be extended between two generic points A and B of the domain as follows

$$\begin{cases} \dfrac{\partial \phi_A}{\partial t} + \dfrac{1}{2}\boldsymbol{\nabla}\phi_A \cdot \boldsymbol{\nabla}\phi_A + h_{is\,A} = \dfrac{1}{2}V_\infty^2 + h_\infty \\ \dfrac{\partial \phi_B}{\partial t} + \dfrac{1}{2}\boldsymbol{\nabla}\phi_B \cdot \boldsymbol{\nabla}\phi_B + h_{is\,B} = \dfrac{1}{2}V_\infty^2 + h_\infty \end{cases} \tag{2.21}$$

$$\Downarrow$$

$$\frac{\partial \phi_A}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi_A \cdot \boldsymbol{\nabla}\phi_A + h_{is\,A} = \frac{\partial \phi_B}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi_B \cdot \boldsymbol{\nabla}\phi_B + h_{is\,B}. \tag{2.22}$$

It will be further shown in chapter 3, that h_{is} can be expressed as a function of the density only, obtaining thus the closed problem (2.23)

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \, \boldsymbol{\nabla} \phi) = 0 \\ \\ \dfrac{\partial \phi}{\partial t} + \dfrac{1}{2} \boldsymbol{\nabla} \phi \cdot \boldsymbol{\nabla} \phi + h_{is} (\rho) = \dfrac{1}{2} V_{\infty}^2 + h_{\infty} \end{cases} \tag{2.23}$$

This formulation is valid for every gas model and not only for the polytropic ideal gas, as long as it is possible to express the enthalpy as a function of the density alone.

2.1.1 Kutta condition

To solve the indetermination of the circulation around lifting bodies it is necessary to cut the domain with a wake and to impose the Kutta condition across it. This requirement is needed only in case of lifting flows and it forces the flow to detach from the trailing edge and not from a point on the airfoil upper surface.

In the stationary case the wake is just a generic mathematical way to obtain a simply connected domain, while in the unsteady case the wake becomes a thin whirling layer along which vorticity is advected. In case of viscous flows the wake is not necessary because this thin whirling layer naturally detaches from the sharp trailing edge due to the effect of viscosity. On the other side, the solution of the Euler equations around lifting bodies deserves some special considerations. In fact in this model viscosity is neglected and it should not be possible to find the correct value of circulation. However due to numerical stabilization, some numerical viscosity is introduced, thus forcing the rear stagnation point to be located at the geometrical trailing edge.

One possible way to impose the Kutta condition in the non stationary case is to enforce the pressure continuity across the wake

$$\Delta P = P_{up} - P_{low} = 0, \tag{2.24}$$

with the obvious meaning of symbols.

In the isentropic case, all thermodynamic variables can be written as functions of the single variable ρ. Therefore it is possible to express the pressure as

$$P = P(\bar{s}, \, \rho) = P_{is}(\rho) \, . \tag{2.25}$$

As it will be better specified in chapter 3, this function is monotone, because

$$\left(\frac{\partial P}{\partial \rho}\right)_s = \frac{\mathrm{d}P_{is}}{\mathrm{d}\rho} = c^2 > 0, \tag{2.26}$$

where c is the speed of sound. Thus condition (2.24) ensures that even density is continuous across the wake.

It follows that enthalpy h_{is} is continuous too and therefore it is possible to obtain the wake equation from the Bernoulli theorem

$$\frac{\partial \phi_{up}}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi_{up} \cdot \boldsymbol{\nabla}\phi_{up} = \frac{\partial \phi_{low}}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi_{low} \cdot \boldsymbol{\nabla}\phi_{low}. \tag{2.27}$$

Introducing the velocity potential jump across the wake $\Delta\phi = \phi_{up} - \phi_{low}$, and expressing

$$\frac{1}{2}\boldsymbol{\nabla}\phi_{up} \cdot \boldsymbol{\nabla}\phi_{up} = \frac{1}{2}\boldsymbol{\nabla}\phi_{up} \cdot \boldsymbol{\nabla}\left(\Delta\phi + \phi_{low}\right) \tag{2.28a}$$

$$\frac{1}{2}\boldsymbol{\nabla}\phi_{low} \cdot \boldsymbol{\nabla}\phi_{low} = \frac{1}{2}\boldsymbol{\nabla}\phi_{low} \cdot \boldsymbol{\nabla}\left(\phi_{up} - \Delta\phi\right) \tag{2.28b}$$

it is possible to obtain

$$\frac{\partial\, \Delta\phi}{\partial t} + \frac{1}{2}\left(\boldsymbol{\nabla}\phi_{up} + \boldsymbol{\nabla}\phi_{low}\right) \cdot \boldsymbol{\nabla}\Delta\phi = 0, \tag{2.29}$$

that is a standard advection equation for the quantity $\Delta\phi$, that is conveyed along the wake with the mean speed between the upper and the lower sides of the wake.

By a momentum conservation across the wake it is possible to demonstrate that the normal velocity must be the same on the upper and the lower sides of the wake. Therefore the following condition on the velocity potential must hold

$$\boldsymbol{\nabla}\phi_{up} \cdot \hat{\boldsymbol{n}}_{\text{wake}} = \boldsymbol{\nabla}\phi_{low} \cdot \hat{\boldsymbol{n}}_{\text{wake}}\,, \tag{2.30}$$

where $\hat{\boldsymbol{n}}_{\text{wake}}$ is the normal to the wake. This is a condition only on the derivatives of ϕ and it is important to keep in mind that the value velocity potential is generally discontinuous across the wake, with the jump value established by the Kutta condition. In the unsteady case it could happen that the tangential component of the velocity to the wake is discontinuous, making the wake a thin whirling layer. By the Kelvin theorem, when the circulation (i.e. lift) on the airfoil increases, it is necessary that some negative circulation is released in the wake, since that is the only place where vorticity can be confined being the remainder of the domain irrotational.

2.2 1–field full potential formulation

The one field formulation of the full potential problem can be derived expressing the density as a function of the velocity potential by means of the Bernoulli theorem. To do that a model for the gas has to be introduced in order to specify the function $h_{is}(\rho)$. In this paragraph the gas that will be considered is the Polytropic Ideal Gas (PIG). The thermodynamic properties of this model of gas will be better described in chapter 3, but up to now, the only necessary function is the enthalpy that, in the case of isentropic transformation can be expressed as

$$h_{is}^{PIG}(\rho) = \frac{\gamma}{\gamma-1}\frac{P_\infty}{\rho_\infty^\gamma}\rho^{\gamma-1}. \tag{2.31}$$

Introducing equation (2.31) in equation (2.20), it is possible to specialize the Bernoulli theorem to the particular case of polytropic ideal gas as

$$\frac{\partial\phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi + \frac{\gamma}{\gamma-1}\frac{P}{\rho} = \frac{1}{2}V_\infty^2 + \frac{\gamma}{\gamma-1}\frac{P_\infty}{\rho_\infty}. \tag{2.32}$$

Noting that for the PIG model the speed of sound c can be expressed as

$$c^2(P,\rho) = \frac{\gamma P}{\rho} \tag{2.33}$$

and that for an isentropic transformation the pressure can be obtained from

$$\frac{P}{\rho^\gamma} = \frac{P_\infty}{\rho_\infty^\gamma}, \tag{2.34}$$

it is possible to solve equation (2.32) for ρ

$$\rho = \rho_\infty\left[1 - \frac{\gamma-1}{c_\infty^2}\left(\frac{\partial\phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi - \frac{1}{2}V_\infty^2\right)\right]^{\frac{1}{\gamma-1}}. \tag{2.35}$$

Equation (2.35) is then substituted in the integral form of the mass conservation equation to obtain a formulation suitable to be discretized using the finite volume method that is

$$\begin{aligned}\int_V \frac{\partial}{\partial t}\rho_\infty\left[1 - \frac{\gamma-1}{c_\infty^2}\left(\frac{\partial\phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi - \frac{1}{2}V_\infty^2\right)\right]^{\frac{1}{\gamma-1}} \\ + \oint_{\partial V}\rho_\infty\left[1 - \frac{\gamma-1}{c_\infty^2}\left(\frac{\partial\phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi - \frac{1}{2}V_\infty^2\right)\right]^{\frac{1}{\gamma-1}}\boldsymbol{\nabla}\phi\cdot\hat{\boldsymbol{n}} = 0.\end{aligned} \tag{2.36}$$

Computing the time derivative in the first term it is possible to obtain the final equation of the 1–field formulation.

$$-\int_V \frac{\rho_\infty}{c_\infty^2}\left[1-\frac{\gamma-1}{c_\infty^2}\left(\frac{\partial\phi}{\partial t}+\frac{1}{2}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi-\frac{1}{2}V_\infty^2\right)\right]^{\frac{2-\gamma}{\gamma-1}}\left[\frac{\partial^2\phi}{\partial t^2}+\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\left(\frac{\partial\phi}{\partial t}\right)\right]$$
$$+\oint_{\partial V}\rho_\infty\left[1-\frac{\gamma-1}{c_\infty^2}\left(\frac{\partial\phi}{\partial t}+\frac{1}{2}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi-\frac{1}{2}V_\infty^2\right)\right]^{\frac{1}{\gamma-1}}\boldsymbol{\nabla}\phi\cdot\hat{\boldsymbol{n}}=0. \tag{2.37}$$

This expression can be written as

$$M\left(\phi,\dot{\phi}\right)\ddot{\phi}+C\left(\phi,\dot{\phi}\right)\dot{\phi}+K\left(\phi\right)\phi=0, \tag{2.38}$$

where, using for example a node centered finite volume formulation and approximating the solution as

$$\phi\left(\boldsymbol{r},t\right)=\sum_{k=1}^{N_v}N_k\left(\boldsymbol{r}\right)\Phi_k\left(t\right) \tag{2.39}$$

the mass, dumping and stiffness matrices are given by

$$M_{ik}\left(\phi,\dot{\phi}\right)=-\int_{V_i}\frac{\rho_\infty}{c_\infty^2}\left[1-\frac{\gamma-1}{c_\infty^2}f(\phi,\dot{\phi},V_\infty)\right]^{\frac{2-\gamma}{\gamma-1}}N_k\left(\boldsymbol{r}\right), \tag{2.40}$$

$$C_{ik}\left(\phi,\dot{\phi}\right)=-\int_{V_i}\frac{\rho_\infty}{c_\infty^2}\left[1-\frac{\gamma-1}{c_\infty^2}f(\phi,\dot{\phi},V_\infty)\right]^{\frac{2-\gamma}{\gamma-1}}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}N_k\left(\boldsymbol{r}\right), \tag{2.41}$$

$$K_{ik}\left(\phi,\dot{\phi}\right)=\oint_{\partial V_i}\rho_\infty\left[1-\frac{\gamma-1}{c_\infty^2}f(\phi,\dot{\phi},V_\infty)\right]^{\frac{1}{\gamma-1}}\boldsymbol{\nabla}N_k\left(\boldsymbol{r}\right)\cdot\hat{\boldsymbol{n}}, \tag{2.42}$$

where $f(\phi,\dot{\phi},V_\infty)=\dfrac{\partial\phi}{\partial t}+\dfrac{1}{2}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi-\dfrac{1}{2}V_\infty^2$, i is the index of the finite volume and N_v is the number of finite volumes.

Wake In the 1–field formulation, since the single variable is the velocity potential ϕ, the only way to impose the Kutta conditions is to use equation (2.27).

Many numerical methods relies on the enforcing of conservation equations over control volumes. If an implicit time discretization has been adopted, it is generally needed to solve a linear system. To impose the Kutta condition it is therefore necessary to free some lines in the system where the discretized

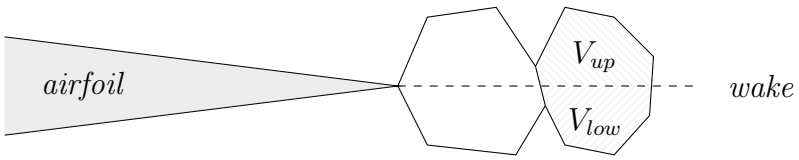


Figure 2.1: Wake

counterpart of equation (2.27) will be overwritten. To free these lines it is possible to perform a linear combination of the two equations that discretize the mass conservation above and below the wake. This procedure is equivalent to enforce the mass conservation across the wake.

To clarify ideas, referring to figure 2.1, the system of equations

$$\begin{cases} \dfrac{d}{dt}\displaystyle\int_{V_{up}} \rho_{up} + \oint_{\partial V_{up}} \rho_{up}\, \boldsymbol{\nabla}\phi_{up} \cdot \hat{\boldsymbol{n}} & = 0 \\ \dfrac{d}{dt}\displaystyle\int_{V_{low}} \rho_{low} + \oint_{\partial V_{low}} \rho_{low}\, \boldsymbol{\nabla}\phi_{low} \cdot \hat{\boldsymbol{n}} & = 0 \end{cases} \tag{2.43}$$

is transformed into the system

$$\begin{cases} \dfrac{d}{dt}\displaystyle\int_{V_{up}\cup V_{low}} \rho + \oint_{\partial V_{up}\cup\partial V_{low}} \rho\, \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}} = 0 \\ \dfrac{\partial \phi_{up}}{\partial t} + \dfrac{1}{2}\boldsymbol{\nabla}\phi_{up} \cdot \boldsymbol{\nabla}\phi_{up} = \dfrac{\partial \phi_{low}}{\partial t} + \dfrac{1}{2}\boldsymbol{\nabla}\phi_{low} \cdot \boldsymbol{\nabla}\phi_{low} \end{cases} \tag{2.44}$$

2.3 2–fields full potential formulation

In the 2–fields formulation, system (2.23) is directly used, without the necessity to reduce the system to a single equation of 1 unknown only. It will be shown in section §2.4 that this formulation, despite dealing with two variables instead of one, presents many advantages over the 1–field formulation.

Wake The Kutta condition can be here imposed in many different ways depending on the adopted numerical discretization technique. The form (2.24) and the form (2.27) are completely equivalent. The first one, as stated in section §2.1.1, implies that

$$\rho_{up} = \rho_{low}, \tag{2.45}$$

and it is therefore a linear constraint in one of the two unknowns. It is thus preferable to the version (2.27).

In an implicit time discretization technique, it is necessary to perform the same linear combinations on the equations of the solving system as in the 1–field implicit case.

2.3.1 ALE Formulation

A possible way to obtain the solution around a moving or deforming body is to formulate the problem in the Arbitrary Lagrangian Eulerian (ALE) form. Another possible approach is to use the transpiration boundary conditions (see section §5.5.2), although this strategy is suitable for small displacements only.

The ALE formulation is deepened in appendix A. The ALE form of the 2–fields full potential problem is given by

$$\begin{cases} \displaystyle\int_{V(t)} \frac{D_{\boldsymbol{v}_m}\rho}{Dt} + \oint_{\partial V(t)} \rho \left(\boldsymbol{\nabla}\phi - \boldsymbol{v}_m\left(\boldsymbol{r},\, t\right)\right) \cdot \hat{\boldsymbol{n}} = 0 \\ \displaystyle\frac{D_{\boldsymbol{v}_m}\phi}{Dt} - \boldsymbol{v}_m \cdot \boldsymbol{\nabla}\phi + \frac{1}{2}\boldsymbol{\nabla}\phi \cdot \boldsymbol{\nabla}\phi + h_{is}\left(\rho\right) = \frac{1}{2}V_\infty^2 + h_\infty \end{cases} \tag{2.46}$$

where the volume $V\left(t\right)$ is a time dependent control volume connected to the mesh that moves through the fluid with the assigned velocity $\boldsymbol{v}_m\left(\boldsymbol{r},\, t\right)$, with the subscript m denoting that it is the mesh speed. This formulation is valid only if the geometric conservation property

$$\boldsymbol{\nabla} \cdot \boldsymbol{v}_m = 0 \tag{2.47}$$

is satisfied. This is always true for rototranslations.

2.4 1–field vs. 2–fields

The two different formulations (2.23) and (2.37) are now analyzed and their pros and cons are discussed.

2.4.1 Number of unknowns

The first and probably only obvious advantage of the 1–field formulation is that it involves half of the unknowns with respect to the 2–fields formulation. This could be a remarkable aspect in an implicit time stepping method.

In fact it requires the solution of many linear systems whose factorization cost increases as a power of the size of the matrix[2]. Moreover, the less the unknowns, the less the memory required to hold the factorized matrices.

This aspect leads to the possibility to solve problems with more control volumes than those that could be solved with the 2–fields formulation using the same memory amount. Despite this advantage, it should be noticed that the 1–field formulation (2.37) is a strongly non linear equation. Hence the computations required to assemble the 1–field equations are much heavier than those of the 2–fields, blurring partially away the time that can be earned earned solving a smaller system.

2.4.2 Time discretization

Moreover, looking at the time discretization problem, the 2–fields formulation is a system of first order explicit partial differential equations, i.e., once the system (2.23) has been discretized in space, it can be generally expressed as

$$\dot{x} = f\left(x\right), \tag{2.48}$$

with $x = \left(\rho,\, \phi\right)^T$. On the contrary, the 1–field equation (2.37) is implicit in the second order time derivative, leading to a system of equations of the form

$$g\left(\phi,\, \dot{\phi},\, \ddot{\phi}\right) = 0. \tag{2.49}$$

Explicit time scheme This aspect leads to a great difference if an explicit time stepping technique is used, because in the case of equation (2.48) it is possible to obtain the solution without solving a nonlinear problem. Using for example a forward Euler scheme the solution at the time step t_{n+1} is obtained as

$$x^{n+1} = x^n + \Delta t\ f\left(x^n\right) \tag{2.50}$$

and the nonlinearity of the system affects only the computing of the resudual $f\left(x^n\right)$.

On the contrary, in the case of equation (2.49), introducing the explicit

[2]The cost of a basic LU factorization is proportional to n^3, but depending on the sparsity of the matrix that exponent can slightly decrease.

Euler method (2.51)

$$\dot{\phi}^n = \frac{\phi^{n+1} - \phi^n}{\Delta t} \tag{2.51a}$$

$$\ddot{\phi}^n = \frac{\dot{\phi}^{n+1} - \dot{\phi}^n}{\Delta t}, \tag{2.51b}$$

the system (2.49), collocated at the time instant t_n, becomes

$$g\left(\phi^n, \dot{\phi}^n, \ddot{\phi}^n\right) = 0 \tag{2.52}$$

$$g\left(\phi^n, \dot{\phi}^n, \frac{\dot{\phi}^{n+1} - \dot{\phi}^n}{\Delta t}\right) = 0, \tag{2.53}$$

that is a nonlinear system in the unknown $\dot{\phi}^{n+1}$ that can be for example solved with the Newton–Raphson agorithm

while $|\delta\dot{\phi}_k| < \text{toll}$

$$\dot{\phi}_0^{n+1} = \dot{\phi}^n \tag{2.54a}$$

$$\left[\frac{\partial g}{\partial \dot{\phi}^{n+1}}\right]_k \left\{\delta\dot{\phi}_k\right\} = -\left\{g\left(\phi^n, \dot{\phi}^n, \frac{\dot{\phi}_k^{n+1} - \dot{\phi}^n}{\Delta t}\right)\right\} \tag{2.54b}$$

$$\dot{\phi}_{k+1}^{n+1} = \dot{\phi}_k^{n+1} + \delta\dot{\phi}_k \tag{2.54c}$$

end while

where n is the temporal index (as usual) and k is the Newton–Raphson iteration index. It should be noted that despite the time stepping scheme is explicit, equation (2.54b) requires the solution of many linear systems at each time step. Here the non linearity affects the computing of the residual (right hand side of equation (2.54b)) and also the Jacobian matrix $\left[\frac{\partial g}{\partial \dot{\phi}^{n+1}}\right]$ of the system . Using a space discretization technique based on local support basis functions, this matrix is highly sparse, but, if a direct linear system solver is used, it needs to be factorized[3] in order to solve system (2.54b),

[3]If a non–exact Newton–Raphson algorithm is used it could be possible to avoid the matrix re–factorization for many steps

thus requiring a much greater usage of memory resources than the explicit 2–fields formulation.

Implicit time scheme If an implicit time discretization is used it is necessary to solve a set of non–linear equations in both the 1–field and 2–fields cases. This is generally done with a Newton–Raphson algorithm that requires the computing of the jacobian matrix. For 2–fields system, this Jacobian is relatively much simpler than the 1–field case, requiring to compute the functional variation of system (2.23) instead of equation (2.37). The resulting formulas are much more complex in the 1–field case, reducing the benefits given by the halved number of unknowns. These formulas are not presented here because they are beyond the scope of the present work and strongly depend on the time discretization scheme adopted.

2.4.3 Real gases

One of the main objectives of this work is to investigate the possibility to include a real gas model in the full potential formulation. The van der Walls model is the simplest way to represent real gas effects and it will be formally introduced in chapter 3. When this particular gas is subjected to an isentropic transformation its enthalpy h_{is} changes according to

$$h_{is}^{vdW}(\rho) = \left(P_\infty + a\,\rho_\infty^2\right)\left(\frac{\rho}{\rho_\infty}\,\frac{1-\rho_\infty\,b}{1-\rho\,b}\right)^{\gamma}\,\frac{\gamma-\rho\,b}{(\gamma-1)\,\rho} - 2\,a\,\rho, \tag{2.55}$$

where ρ is the gas density and a, b and γ are gas specific constants that will be better explained in chapter 3. Relation (2.55) corresponds exactly to equation (2.31) for the polytropic ideal gas, but it cannot be analytically solved for the variable ρ. This fact prevents to reverse the Bernoulli theorem to obtain an expression that relates the fluid density to the velocity potential, as it was done in the case of polytropic ideal gas equation (2.35). This obstacle makes the 1–field approach unfeasible when real gas effects are sought for.

2.4.4 Transonic flows

As it will be better explained in section §5.3.2, for numerical stability reasons it is generally needed to use an upwinding technique on the value of density where the flow is supersonic. This allows to restore the spacial causality when computing the mass flux through a surface of a control volume. In fact, where the fluid speed is greater than the speed of sound, the flow particles cannot

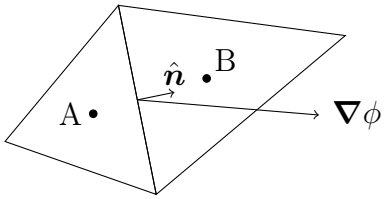


Figure 2.2: Mass flux

"feel" what's happening downstream of their position. It is thus necessary to use an upstream value of density, that can be computed using a linear approximation of the density field around the point where the flux has to be computed. This means that the flux is computed with a value of density $\tilde{\rho}$ defined as

$$\tilde{\rho} = \rho + \boldsymbol{\nabla}\rho \cdot \boldsymbol{\Delta l}, \tag{2.56}$$

where $\boldsymbol{\Delta l}$ is a vector in the streamline direction pointing countercurrent. The length of this vector can be changed, allowing a calibration of the numerical dissipation needed to stabilize the numerical integration. It should be noted that equation (2.56), requires the gradient of the density field. In the 2–fields formulation this is accomplished computing the first order spatial derivatives of one of the the unknowns, while in the 1–field formulation it must be computed as

$$\boldsymbol{\nabla}\rho = \rho_\infty \boldsymbol{\nabla}\left[1 - \frac{\gamma-1}{c_\infty^2}\left(\frac{\partial\phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi \cdot \boldsymbol{\nabla}\phi - \frac{1}{2}V_\infty^2\right)\right]^{\frac{1}{\gamma-1}}. \tag{2.57}$$

Equation (2.57) requires the computing of second order spatial derivatives of the velocity potential. If the solution is approximated by means of a Ritz series expansion, this requirement would mean to use a set of basis functions that are at least of class $\mathcal{C}^1$, i.e. whose first order derivatives are continuous and therefore their second order derivatives are integrable. The use of such a basis introduces a lot of technical difficulties in the implementation of a solver, discouraging the use of this upwinding approach for transonic solutions in the 1–field formulation.

A simpler technique to stabilize the numerical integration is to compute the mass flux using the value of density in the upstream control volume, instead of reconstructing $\tilde{\rho}$ by a linear interpolation. With reference to figure 2.2, the mass flux is therefore computed with a value of density $\tilde{\rho}$ given by

$$\tilde{\rho} = \begin{cases} \rho_A & \text{if } \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}} > 0 \\ \\ \rho_B & \text{if } \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}} < 0 \end{cases} \tag{2.58}$$

This strategy can be applicable in the 1–field formulation too but it does not allow any control of the numerical dissipation.

2.4.5 ALE formulation

The Arbitrary Lagrangian Eulerian (ALE) formulation is a suitable approach when the boundaries of the domain are not fixed in space but they need to move with an assigned trajectory. This technique can be used for example to solve aeroelastic problems where a flexible structure must be left free to deform in the flow field.

A detailed presentation of the theory on which this formulation relies is given in appendix A, but some elements must be introduced here in order to highlight the differences between the 1–field and the 2–fields approaches.

In order to obtain the ALE formulation it is necessary to relate the time derivatives of the variables connected to the grid with the time derivative of the Eulerian field. To do that it is necessary to introduce the symbol $\frac{D_{\boldsymbol{v}_m}}{D\,t}$, that represents the time derivative with respect to an observer connected to the mesh nodes. In this notation $\boldsymbol{v}_m = \boldsymbol{v}_m(\boldsymbol{r}, t)$ and $\boldsymbol{a}_m = \boldsymbol{a}_m(\boldsymbol{r}, t)$ indicates respectively the mesh velocity and acceleration expressed in Eulerian form. The subscript v_m is used here to distinguish that symbol from the material time derivative. These relations are

$$\frac{D_{\boldsymbol{v}_m}\rho}{D\,t} = \boldsymbol{\nabla}\rho \cdot \boldsymbol{v}_m + \frac{\partial\rho}{\partial t} \tag{2.59}$$

and

$$\frac{D^2_{\boldsymbol{v}_m}\phi}{D\,t^2} = [\boldsymbol{\nabla}(\boldsymbol{\nabla}\phi) \cdot \boldsymbol{v}_m] \cdot \boldsymbol{v}_m + 2\,\frac{\partial\,\boldsymbol{\nabla}\phi}{\partial\,t} \cdot \boldsymbol{v}_m + \boldsymbol{\nabla}\phi \cdot \boldsymbol{a}_m + \frac{\partial^2\phi}{\partial\,t^2} \tag{2.60a}$$

$$= \boldsymbol{\nabla}(\boldsymbol{\nabla}\phi) : (\boldsymbol{v}_m \otimes \boldsymbol{v}_m) + 2\,\frac{\partial\,\boldsymbol{\nabla}\phi}{\partial\,t} \cdot \boldsymbol{v}_m + \boldsymbol{\nabla}\phi \cdot \boldsymbol{a}_m + \frac{\partial^2\phi}{\partial\,t^2} \tag{2.60b}$$

$$= \phi_{/ij}\; v_i^m\; v_j^m + 2\,\dot{\phi}_{/i}\; v_i^m + \phi_{/i}\; a_i^m + \ddot{\phi} \tag{2.60c}$$

that solved for the partial time derivatives give

$$\frac{\partial\rho}{\partial t} = \frac{D_{\boldsymbol{v}_m}\rho}{D\,t} - \boldsymbol{\nabla}\rho \cdot \boldsymbol{v}_m \tag{2.61}$$

and

$$\frac{\partial^2 \phi}{\partial\, t^2} = \frac{D^2_{\boldsymbol{v}_m} \phi}{D\, t^2} + \boldsymbol{\nabla}\left(\boldsymbol{\nabla}\phi\right) : \left(\boldsymbol{v}_m \otimes \boldsymbol{v}_m\right) - 2\, \frac{D_{\boldsymbol{v}_m} \boldsymbol{\nabla}\phi}{D\, t} \cdot \boldsymbol{v}_m - \boldsymbol{\nabla}\phi \cdot \frac{\partial \boldsymbol{v}_m}{\partial t}. \tag{2.62}$$

At the right hand side there are now quantities that are all computable on a moving mesh, being all the time derivatives of the kind $\frac{D_{\boldsymbol{v}_m}}{Dt}$. Substituting these relations in the Eulerian full potential problem, it is possible to obtain its ALE counterpart.

It must be underlined that the 2–fields formulation is first order in time and therefore only equation (2.61) is needed. In the 1–field approach, that is second order in time, also equation (2.62) is necessary. This relation however introduces second order derivatives in space. This in turn requires $\mathcal{C}^1$ shape functions.

Chapter 3

Thermodynamics

As shown in chapter 2 the full potential formulation requires some knowledge of the gas thermodynamic behavior. This is required since it is necessary to know the enthalpy function that connects the mass conservation equation to the Bernoulli theorem. Thermodynamics allows to take into account compressibility effects that are not negligible in many modern aerodynamic applications such as transonic flight or turbine expanders. Neglecting or excessively approximating those effects can lead to a poor design of lifting surfaces and profiles with a great loss in working efficiency and a consequently rising in operational costs of the final product.

In this chapter after a general introduction to thermodynamics, the case of polytropic ideal gas and van der Waals gas are examined. The latter allows the study of some peculiar phenomena typical of the dense gas regime such as expansion shocks and mixed waves. These topics, detailed in section §3.4, are currently of great interest among the scientific community because it has been recently shown in literature that some of these phenomena could be exploited to improve the efficiency in some turbomachinery applications [23],[24] as Organic Rankine Cycle (ORC) engines. In section §3.5 it is shown a useful strategy to extend the possibility to use ExPReS with general gas models that better describe the dense gas region.

3.1 The axiomatic approach to thermodynamics

The brief introduction to thermodynamics of gases here proposed is based on the paper [25] by L. Galgani and A. Scotti and taken up by Callen in [26]. In the present work the axiomatic approach to thermodynamics is presented following appendix D of [27] and the emphasis is given to the fundamental

thermodynamic quantity that is the entropy.

The adjective axiomatic comes from the three axioms that the fundamental quantity must comply with in order to represent the thermodynamic properties of a physical system. One of these hypothesis, the superadditivity, brings in the mathematical function entropy the second principle of thermodynamics that represents the irreversibility of every physical process.

In this formulation not only the entropy, but also the internal energy can be taken as the primary quantity from which all other thermodynamic variables can be derived, leading to the so called energetic representation. This approach is completely equivalent to the entropic one since both can be deduced from the other.

One of the main advantages of the axiomatic formulation is that all other derived quantities, such as pressure and temperature, can be obtained differentiating these primary functions.

For the purpose of this work only simple thermodynamic systems are considered. Such a system is composed by a single chemical component in isotropic fluid state (liquid or gaseous) and in absence of any chemical reaction. Moreover no liquid–vapor phase transition are here considered.

In a simple closed thermodynamic system all the equilibrium states are completely identified by its internal energy E, volume V, and mass M through the entropy function

$$S = S\left(E,\, V,\, M\right). \tag{3.1}$$

On the contrary a simple equation of state is not enough to fully represent the thermodynamic state of a fluid. For example, as it will be shown in section §3.2, from the well known equation of state of an ideal gas $PV = RT$ it is not possible to reconstruct the fundamental relation while the reverse path is always possible. This is due to the fact that the equation of state is obtained by a differentiation process carried out on relation (3.1) with a consequent loss of information.

The aforementioned three properties that the entropy function (3.1) must satisfy are:

- First order homogeneity

$$S\left(\lambda E,\, \lambda V,\, \lambda M\right) = \lambda\, S\left(E,\, V,\, M\right) \qquad \lambda > 0; \tag{3.2}$$

- Monotony with respect to the variable E

$$S\left(E',\, V,\, M\right) \geq S\left(E'',\, V,\, M\right) \quad \Longleftrightarrow \quad E' \geq E''; \tag{3.3}$$

- Superadditivity

$$S\left(E_1 + E_2,\, V_1 + V_2,\, M_1 + M_2\right) \geq S\left(E_1,\, V_1,\, M_1\right) + S\left(E_2,\, V_2,\, M_2\right); \tag{3.4}$$

The property of the entropy superadditivity is a way to express the well known second principle of thermodynamics and plays the role of irreversibility in the mathematical description of thermodynamic transformations.

To obtain the energetic representation, the second property plays an important role since it allows to solve the entropy function for the variable E

$$E = E\left(S,\, V,\, M\right) \tag{3.5}$$

and since the inverse of a monotonic function is still monotonic, even the internal energy E is invertible too. This alternation highlights the equivalence of the energetic and entropic representations.

To complete the thermodynamic theory it is necessary to add to the three axioms the Nernst's postulate

$$\left(\frac{\partial E}{\partial S}\right)_{V,M} \to 0 \quad \Longrightarrow \quad S \to 0, \tag{3.6}$$

where the derivative $(\partial E/\partial S)_{V,M}$ will be shown in section §3.1.1 to be the definition of the temperature.

Exploiting the homogeneity property of the entropy it is possible to introduce the specific version of the fundamental relation (3.1)

$$s\left(e,\, v\right) \triangleq \frac{1}{M} S\left(E,\, V,\, M\right) = S\left(\frac{E}{M},\, \frac{V}{M},\, 1\right) = S\left(e,\, v,\, 1\right), \tag{3.7}$$

where the specific energy e and volume v have been defined as

$$e \triangleq \frac{E}{M} \tag{3.8}$$

and

$$v \triangleq \frac{V}{M}. \tag{3.9}$$

Similarly, it is possible to introduce the specific version of the fundamental relation (3.5)

$$e\left(s,\, v\right) \triangleq \frac{1}{M} E\left(S,\, V,\, M\right) = E\left(\frac{S}{M},\, \frac{V}{M},\, 1\right) = E\left(s,\, v,\, 1\right). \tag{3.10}$$

3.1.1 First law of thermodynamics

The first law of thermodynamics states that the total energy of an isolated system is constant. It can be inferred from differentiating the function $e = e(s, v)$

$$\mathrm{d}e = \left(\frac{\partial e}{\partial s}\right)_v \mathrm{d}s + \left(\frac{\partial e}{\partial v}\right)_s \mathrm{d}v. \tag{3.11}$$

By definition the two derivative that appear in relation 3.11 are respectively the temperature and the pressure

$$T = T(s, v) \triangleq \left(\frac{\partial e}{\partial s}\right)_v \tag{3.12a}$$

$$P = P(s, v) \triangleq -\left(\frac{\partial e}{\partial v}\right)_s \tag{3.12b}$$

and hence the first law can be rewritten as

$$\mathrm{d}e = T\,\mathrm{d}s - P\,\mathrm{d}v, \tag{3.13}$$

which is also known as

$$\mathrm{d}e = Q_\delta + W_\delta, \tag{3.14}$$

where the infinitesimal amount of heat Q_δ and of work have been defined as

$$Q_\delta \triangleq T\,\mathrm{d}s \tag{3.15}$$

$$W_\delta \triangleq -P\,\mathrm{d}v. \tag{3.16}$$

The particular notation $(.)_\delta$ is taken from [27] to highlight the fact that those quantities are not two exact differentials because they depend from the transformation path. Since they are not state variables, any variation of them, even infinitesimal, is meaningless.

The two relations (3.12) are the so called equations of state that are both necessary to fully represent the thermodynamic system. However they are not independent because they are obtained differentiating the same primary function. In fact, using the Schwarz theorem, it is possible to show the connection between the two equations of state

$$\left(\frac{\partial T}{\partial v}\right)_s = \frac{\partial^2 e}{\partial v\,\partial s} = \frac{\partial^2 e}{\partial s\,\partial v} = -\left(\frac{\partial P}{\partial s}\right)_v, \tag{3.17}$$

that is one of the compatibility conditions between the two functions $T(s, v)$ and $P(s, v)$.

3.1.2 Enthalpy

As shown in chapter 2 the 2–fields formulation (2.23) requires the knowledge of the thermodynamic potential enthalpy. It can be obtained applying the Legendre transformation to the internal energy substituting the specific volume v with the pressure P. The Legendre transformation is necessary since the pressure is a derivative of the internal energy and thus, as demonstrated in [27], a direct change of the independent variable from the specific volume to the pressure, would imply a loss of information. However this transformation leads to the introduction of a new function that guarantees that all the informative content will be preserved.

To define the Legendre transformation, consider the general function $f(x,\, y)$ and its partial derivative

$$z = \frac{\partial f(x,\, y)}{\partial y} = \mathcal{Z}(x,\, y)\,, \tag{3.18}$$

which can be inverted with respect to the variable $y = \mathcal{Z}^{-1}(x,\, z) = \mathcal{Y}(x,\, z)$.

If it is necessary to change the independent variable of the function from x to z without any loss of information, it is necessary to define a new function $g(z,\, y)$ as

$$g = f - z\,y, \tag{3.19}$$
$$g(x,\, z) = f(x,\, \mathcal{Y}(x,\, z)) - z\,\mathcal{Y}(x,\, z)\,, \tag{3.20}$$

where the new function g does no more depend on the variable y.

Applying the Legendre transformation to the internal energy substituting the variable v with P, the enthalpy function is obtained

$$P = -\frac{\partial e(s,\, v)}{\partial v} = \mathcal{P}(s,\, v)\,, \tag{3.21}$$

which can be inverted with respect to the variable $v = \mathcal{P}^{-1}(s,\, P) = \mathcal{V}(s,\, P)$. The enthalpy is therefore defined as

$$h = e + Pv \tag{3.22}$$
$$h(s,\, P) = e(s,\, \mathcal{V}(s,\, P)) + P\,\mathcal{V}(s,\, P)\,. \tag{3.23}$$

3.1.3 Specific heats

In order to better understand some peculiarities of dense gas behavior, it is necessary to introduce the constant volume and constant pressure specific heats

$$c_v \triangleq T\left(\frac{\partial s}{\partial T}\right)_v, \tag{3.24}$$

$$c_P \triangleq T\left(\frac{\partial s}{\partial T}\right)_P. \tag{3.25}$$

These two quantities represent the infinitesimal amount of heat that a unitary mass must be given to increase its temperature of one unit in a process carried out respectively at constant volume or at constant pressure.

From the first law of thermodynamics $T\,\mathrm{d}s = \mathrm{d}e + P\,\mathrm{d}v$ it is immediate to find out that

$$c_v = \left(\frac{\partial e}{\partial T}\right)_v, \tag{3.26}$$

$$c_P = \left(\frac{\partial e}{\partial T}\right)_P + P\left(\frac{\partial v}{\partial T}\right)_P = \left(\frac{\partial h}{\partial T}\right)_P. \tag{3.27}$$

3.1.4 Polytropic assumption

To clarify what the adjective polytropic means, consider the function $e = e(s, v)$ and perform the variable substitution (with a certain notational abuse, since $e(T, v)$ and $e(s, v)$ are two different mathematical functions)

$$e(T, v) = e(\mathcal{S}(T, v), v). \tag{3.28}$$

This variable substitution would lead to a loss of information as stated in section §3.1.2, and therefore the Legendre transformation is carried out, defining the Helmholtz thermodynamic potential f

$$f(T, v) = e(\mathcal{S}(T, v), v) - T\,\mathcal{S}(T, v) \tag{3.29a}$$

$$f(T, v) = e(T, v) - T\,\mathcal{S}(T, v), \tag{3.29b}$$

where $s = \mathcal{S}(T, v)$ is obtained solving the function $T = \mathcal{T}(s, v) = (\partial e/\partial s)_v$ for the variable s.

In the following steps, it will be necessary to differentiate equations (3.29) with respect to their two variables. Consider first the variable v and let's differentiate equation (3.29a) with respect to it

$$\left(\frac{\partial f}{\partial v}\right)_T = \left(\frac{\partial e}{\partial s}\right)_v\Bigg|_{s=\mathcal{S}(T,v)}\left(\frac{\partial \mathcal{S}}{\partial v}\right)_T + \left(\frac{\partial e}{\partial v}\right)_s\Bigg|_{s=\mathcal{S}(T,v)} - T\left(\frac{\partial \mathcal{S}}{\partial v}\right)_T. \tag{3.30}$$

Recalling definitions (3.12), leads to

$$\left(\frac{\partial f}{\partial v}\right)_T = -P\left(T,\, v\right). \tag{3.31}$$

Let's differentiate now equation (3.29b)

$$\left(\frac{\partial f}{\partial v}\right)_T = \left(\frac{\partial e}{\partial v}\right)_T - T\left(\frac{\partial \mathcal{S}}{\partial v}\right)_T. \tag{3.32}$$

Comparing equation (3.31) and equation (3.32), the following relation is obtained

$$\left(\frac{\partial e}{\partial v}\right)_T = T\left(\frac{\partial \mathcal{S}}{\partial v}\right)_T - P\left(T,\, v\right). \tag{3.33}$$

Consider now the derivative of relation (3.29b) with respect to the variable T

$$\left(\frac{\partial f}{\partial T}\right)_v = \left(\frac{\partial e}{\partial T}\right)_v - \mathcal{S}\left(T,\, v\right) - T\left(\frac{\partial \mathcal{S}}{\partial T}\right)_v \tag{3.34}$$

and recalling the definition of the constant volume specific heat (3.24) and (3.26), it is possible to obtain

$$\left(\frac{\partial f}{\partial T}\right)_v = -\mathcal{S}\left(T,\, v\right). \tag{3.35}$$

Considering the cross second order derivatives of the Helmholtz potential, and applying the Schwartz theorem

$$\left(\frac{\partial \mathcal{S}}{\partial v}\right)_T = -\frac{\partial^2 f}{\partial v\, \partial T} = -\frac{\partial^2 f}{\partial T\, \partial v} = \left(\frac{\partial P}{\partial T}\right)_v, \tag{3.36}$$

it is possible to rewrite equation (3.33) as

$$\left(\frac{\partial e}{\partial v}\right)_T = T\left(\frac{\partial P}{\partial T}\right)_v - P\left(T,\, v\right). \tag{3.37}$$

Integrating equation (3.37) in the variable v, the energy function $e\left(T,\, v\right)$ can be obtained

$$e = \phi\left(T\right) + \int_{v_0}^{v} \left[T\left(\frac{\partial P}{\partial T}\right)_v - P\left(T,\, v\right)\right] \mathrm{d}v, \tag{3.38}$$

where the function $\phi\left(T\right)$ is necessary since the integration is carried out on the variable v only. Its derivative is indicated as

$$c_v^0(T) = \frac{\mathrm{d}\phi}{\mathrm{d}T} \tag{3.39}$$

and represent the ideal gas contribution to the total value of $c_v(T, v)$. The full expression of the constant volume specific heat is then

$$\begin{aligned} c_v(T, v) &= \frac{\mathrm{d}\phi}{\mathrm{d}T} + \int_{v_0}^{v} T\left(\frac{\partial^2 P}{\partial T^2}\right)_v \mathrm{d}v \\ &= c_v^0(T) + \int_{v_0}^{v} T\left(\frac{\partial^2 P}{\partial T^2}\right)_v \mathrm{d}v. \end{aligned} \tag{3.40}$$

The hypothesis of polytropic gas implies that c_v^0 does not depend from temperature and is therefore constant.

As it will be shown in section §3.2 and §3.3, for the two particular models of polytropic ideal gas and polytropic van der Waals gas, the value of c_v (and not only c_v^0) is constant since the integrand function annihilates.

3.1.5 Speed of sound

In gas dynamics a relevant quantity is the speed of sound that is the speed that appears in the acoustic equation[1] and with which weak perturbations propagate in the fluid. It is defined as

$$c^2(s, \rho) = \left(\frac{\partial P}{\partial \rho}\right)_s, \tag{3.41}$$

that is always greater than zero for thermodynamic stability reasons ([26]).

3.2 Polytropic ideal gas model

The fundamental entropic relation that defines the polytropic ideal gas (PIG) is

$$s(e, v) = s_0 + R \ln\left[\left(\frac{e}{e_0}\right)^{\frac{1}{\gamma-1}} \frac{v}{v_0}\right], \tag{3.42}$$

where γ is a gas dependent constant and $R = \mathfrak{R}/MM$, with $\mathfrak{R} = 8.314\,\mathrm{J\,K^{-1}\,mol^{-1}}$ that is the universal gas constant and MM the molar mass of the considered gas.

From the kinetic theory it is possible to show that γ depends from the complexity of the gas molecule only[2] and its value is

[1] The acoustic equation is a linearization of the full potential problem

[2] This approximation is valid under the hypothesis that no vibrational degrees of freedom are active in the molecule

$$\gamma = \begin{cases} \frac{5}{3} & \text{monoatomic gas} \\ \frac{7}{5} & \text{linear molecule} \\ \frac{4}{3} & \text{nonlinear molecule} \end{cases}$$

that is valid if all an only the rigid degrees of freedom of the molecule are active.

Solving equation (3.42) for e it is possible to obtain the fundamental relation in its energetic representation

$$e\left(s,\, v\right) = e_0 \left(\frac{v_0}{v}\right)^{\gamma-1} \exp\left[\left(\gamma - 1\right) \frac{s - s_0}{R}\right]. \tag{3.43}$$

Differentiating this relation, recalling definitions (3.12), the two equations of state for the PIG gas can be obtained

$$P\left(s,\, v\right) = \left(\gamma - 1\right) \frac{e_0}{v_0} \left(\frac{v_0}{v}\right)^{\gamma} \exp\left[\left(\gamma - 1\right) \frac{s - s_0}{R}\right] \tag{3.44a}$$

$$T\left(s,\, v\right) = e_0 \left(\frac{v_0}{v}\right)^{\gamma-1} \frac{\gamma - 1}{R} \exp\left[\left(\gamma - 1\right) \frac{s - s_0}{R}\right], \tag{3.44b}$$

where the function (3.43) can be highlighted giving

$$P\left(e,\, \rho\right) = \left(\gamma - 1\right) \rho\, e \tag{3.45a}$$

$$T\left(e\right) = \frac{\gamma - 1}{R}\, e. \tag{3.45b}$$

Combining equations (3.45) it is possible to obtain the well known PIG equation of state

$$P = \rho\, R\, T, \tag{3.46}$$

that by itself does not represent all the thermodynamic characteristics of the gas.

3.2.1 Isentropic transformations

In an isentropic transformation, since entropy remains constant, the variation of pressure can be obtained evaluating equation (3.44a) for $s = s_0$

$$P_{is}\left(v\right) = P\left(s_0,\, v\right) = \left(\gamma - 1\right) \frac{e_0}{v_0} \left(\frac{v_0}{v}\right)^{\gamma}. \tag{3.47}$$

Evaluating now equation (3.44a) in the reference gas state, the reference value for pressure can be obtained

$$P_0 = (\gamma - 1)\,\frac{e_0}{v_0}. \tag{3.48}$$

Dividing equation (3.47) by equation (3.48), the isentropic relation between P and v is obtained

$$\frac{P_{is}}{P_0} = \left(\frac{v_0}{v}\right)^{\gamma}, \tag{3.49}$$

that, changing the variable v to ρ, can be rewritten as

$$P_{is}(\rho) = \frac{P_0}{\rho_0^{\gamma}}\,\rho^{\gamma}, \tag{3.50}$$

that can be re–expressed as the well known

$$\frac{P}{\rho^{\gamma}} = \text{const.} \tag{3.51}$$

3.2.2 Enthalpy

To solve the 2–fields full potential problem (2.23) it is necessary to compute the enthalpy function $h_{is}(\rho)$ for an isentropic transformation. Given that

$$h = e + Pv, \tag{3.52}$$

substituting in equation (3.52), equation (3.45a) solved for e becomes

$$e(P,\,\rho) = \frac{P}{(\gamma - 1)\,\rho}, \tag{3.53}$$

it is thus possible to obtain

$$h(P,\,\rho) = \frac{\gamma}{\gamma - 1}\,\frac{P}{\rho}. \tag{3.54}$$

If the pressure is now expressed in term of isentropic transformation, the final function $h_{is}^{vdW}(\rho)$ can be deduced

$$h_{is}^{PIG}(\rho) = h(P_{is}(\rho),\,\rho) = \frac{\gamma}{\gamma - 1}\,\frac{P_0}{\rho_0^{\gamma}}\,\rho^{\gamma-1}. \tag{3.55}$$

3.2.3 Speed of sound

Recalling definition (3.41), and substituting $v = 1/\rho$

$$c^2(s,\,\rho) = \left(\frac{\partial P}{\partial \rho}\right)_s = -v^2\left(\frac{\partial P}{\partial v}\right)_s = v^2\left(\frac{\partial^2 e}{\partial v^2}\right)_s. \tag{3.56}$$

Performing all the needed differentiations and with the use of equation (3.53), it can be demonstrated that the speed o sound is expressible as

$$c^2(P,\,\rho) = \gamma\,\frac{P}{\rho}, \tag{3.57}$$

that, using the equation of state (3.46), can be reformulated as the well known

$$c^2(T) = \gamma\,R\,T. \tag{3.58}$$

3.3 Polytropic van der Waals gas model

The fundamental entropic relation that defines the polytropic van der Waals (PvdW) model is

$$s(e,\,v) = s_0 + R\ln\left[\left(\frac{e+\frac{a}{v}}{e_0+\frac{a}{v_0}}\right)^{\frac{1}{\delta}}\,\frac{v-b}{v_0-b}\right], \tag{3.59}$$

where δ, a and b are gas dependent constant. It is noted that this equation turns in the PIG fundamental relation with $a = 0$, $b = 0$ and $\delta = \gamma - 1$. By assumption, the quantity δ is constant and therefore γ is constant too. However in this case γ does not represent the specific heats ratio as it is for the PIG model. In fact in the PvdW model, the value of c_p is not constant even under the polytropic assumption. The meaning of constants a and b are respectively the attraction between gas particles (called van der Waals force) and the average volume excluded from v by the presence of gas molecules.

Solving equation (3.59) for e it is possible to obtain the fundamental relation in its energetic representation

$$e(s,\,v) = -\frac{a}{v} + \left(e_0+\frac{a}{v_0}\right)\left(\frac{v_0-b}{v-b}\right)^{\delta}\,\exp\left[\delta\,\frac{s-s_0}{R}\right]. \tag{3.60}$$

With a similar procedure as the one performed for the PIG model, the equations of state for the PvdW gas can be obtained

$$P(s,\, v) = -\frac{a}{v^2} + \delta \left(e_0 + \frac{a}{v_0}\right) \left(\frac{v_0 - b}{v - b}\right)^{\delta} \frac{1}{v - b} \exp\left(\delta \frac{s - s_0}{R}\right) \tag{3.61a}$$

$$T(s,\, v) = \frac{\delta}{R} \left(e_0 + \frac{a}{v_0}\right) \left(\frac{v_0 - b}{v - b}\right)^{\delta} \exp\left(\delta \frac{s - s_0}{R}\right), \tag{3.61b}$$

Equations (3.61), combined together, give the well known equation of state for the van der Waals gas

$$\left(P + a\,\rho^2\right)(1 - b\,\rho) = \rho\, R\, T, \tag{3.62}$$

that by itself, however, does not represent all the gas thermodynamic characteristics.

3.3.1 Isentropic transformations

In an isentropic transformation, since entropy remains constant, the variation of pressure can be obtained evaluating equation (3.61a) for $s = s_0$

$$P_{is}(v) = P(s_0,\, v) = -\frac{a}{v^2} + \delta \left(e_0 + \frac{a}{v_0}\right) \left(\frac{v_0 - b}{v - b}\right)^{\delta} \frac{1}{v - b}. \tag{3.63}$$

Evaluating now equation (3.61a) in the reference gas state, the reference value for pressure can be obtained

$$P_0 = -\frac{a}{v_0^2} + \delta \left(e_0 + \frac{a}{v_0}\right) \frac{1}{v_0 - b}. \tag{3.64}$$

Dividing equation (3.63) by equation (3.64), the isentropic relation between P and v is obtained

$$\left(P_{is} + \frac{a}{v^2}\right)(v - b)^{\gamma} = \left(P_0 + \frac{a}{v_0^2}\right)(v_0 - b)^{\gamma}. \tag{3.65}$$

By changing the variable v to ρ, equation (3.65) can be rewritten as

$$P_{is}(\rho) = \left(P_0 + a\,\rho_0^2\right) \left(\frac{\rho}{\rho_0} \frac{1 - b\,\rho_0}{1 - b\,\rho}\right)^{\gamma} - a\,\rho^2. \tag{3.66}$$

3.3.2 Enthalpy

To solve the 2–fields full potential problem (2.23) it is necessary to compute the enthalpy function $h_{is}(\rho)$ for an isentropic transformation. Given that

$$h_{is}(\rho) = e_{is}(\rho) + \frac{P_{is}(\rho)}{\rho}, \tag{3.67}$$

computing e_{is} from equation (3.60) and imposing $s = s_0$, the final expression for $h_{is}^{vdW}(\rho)$ results

$$h_{is}^{PvdW}(\rho) = \left(P_0 + a\,\rho_0^2\right)\left(\frac{\rho}{\rho_0}\,\frac{1-\rho_0\,b}{1-\rho\,b}\right)^{\gamma}\,\frac{\gamma-\rho\,b}{(\gamma-1)\,\rho} - 2\,a\,\rho. \tag{3.68}$$

3.3.3 Speed of sound

The speed o sound for the polytropic van der Waals gas model is given by

$$c^2(P,\,\rho) = \frac{\gamma\,(P + a\,\rho^2)}{\rho\,(1-\rho\,b)} - 2\,a\,\rho. \tag{3.69}$$

3.4 Dense gas dynamics

In 1971 Thompson introduced the fundamental derivative of gas dynamics [28] defined as

$$\Gamma(s,\,v) \triangleq \frac{v^3}{2\,c^2(s,\,v)}\left(\frac{\partial^2 P}{\partial v^2}\right)_s, \tag{3.70}$$

whose sign is univocally determined by the sign of the second derivative $(\partial^2 P/\partial v^2)_s$, that is the concavity of isentropes in the P–v plane.

Simply changing the variable v to ρ in equation (3.70), it can be shown that

$$\Gamma(s,\,\rho) = 1 + \frac{\rho}{c(s,\,\rho)}\left(\frac{\partial c}{\partial \rho}\right)_s, \tag{3.71}$$

and a simple application of the chain rule with $\rho = 1/v$ leads to the third form of the function Γ

$$\Gamma(s,\,v) = 1 - \frac{v}{c}\left(\frac{\partial c}{\partial v}\right)_s. \tag{3.72}$$

Basing on this function, three different fluid families can be defined depending on the value assumed by Γ in different thermodynamic regions:

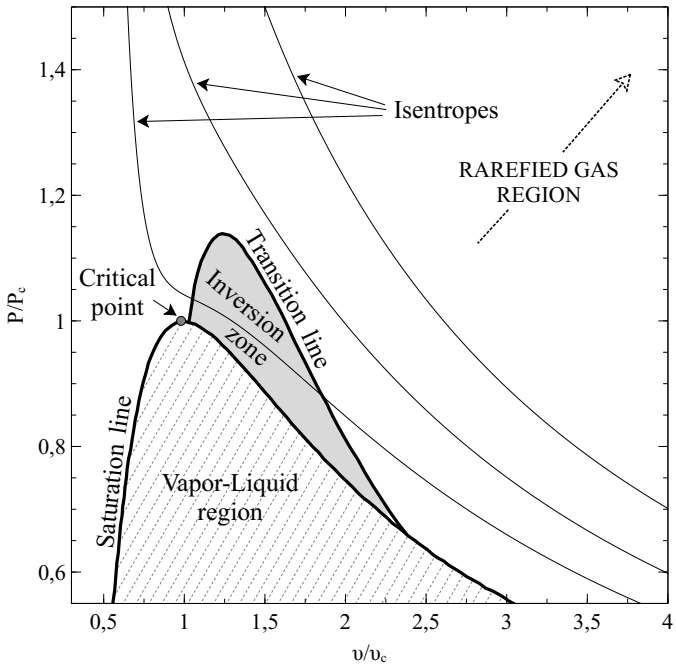


Figure 3.1: P–v diagram near dense gas region

- $\Gamma > 1$ everywhere: Low Molecular Complexity (LMC) fluid
- $\Gamma > 0$ and there exist a thermodynamic region where $0 < \Gamma < 1$: High Molecular Complexity (HMC) fluid
- $\Gamma < 0$ in some thermodynamic region: Bethe–Zel'dovich–Thompson (BZT) fluid

As it is clear from the expression (3.72), in HMC and BZT fluids, it could happen to have $(\partial c/\partial v)_s$ positive, that means that in an isentropic expansion the speed of sound would increase, possibly leading to a reduction in the Mach number. In the test case in section §6.3.1 such a case is presented.

Modeling a gas with the simple mathematical model of the polytropic ideal gas is a good strategy if the gas flow is always in a rarefied regime. However, when dense gas effects must be investigated this model becomes completely unappropriated. For example the value of Γ for the polytropic ideal gas model is always equivalent to $(\gamma + 1)/2$ (see appendix B) that is constant and greater than 1. This would lead to a wrong estimate of the rate of change of the speed of sound with density in isentropic perturbations, making this model incorrect to be used with HMC or BZT fluids.

Inversion region and BZT gases The region where the fundamental derivative Γ goes below zeros is commonly called inversion zone and the curve on the P–v diagram where $\Gamma = 0$ is called transition line. They are both sketched in figure 3.1.

The simplest thermodynamic model that allows the existence of such a region is the van der Waals gas. This model overestimates the extension of that region but it is well capable to represent all the BZT fluid qualitative behaviors. It is possible to show that with this model, the inversion region exists if

$$1 < \gamma \leq 1.06\,. \tag{3.73}$$

Such a value for the constant γ can be obtained using very heavy and molecularly complex fluids. This aspect can be better understood computing the value of c_v for the van der Waals fluid. Recalling equation (3.26), the easiest way to compute it, is to differentiate the function $e = e\,(T,\, v)$. This function can be obtained from equation (3.61b) recognizing in it the energy function

$$T\,(e,\, v) = \frac{\delta}{R}\left(e + \frac{a}{v}\right), \tag{3.74}$$

that solved for e gives

$$e\,(T,\, v) = \frac{R\,T}{\delta} - \frac{a}{v}, \tag{3.75}$$

and therefore

$$c_v = c_v^0 = \frac{R}{\delta} = \frac{R}{\gamma - 1}, \tag{3.76}$$

that is constant as expected since the model is polytropic. Solving equation (3.76) for γ gives

$$\gamma = \frac{R}{c_v} + 1, \tag{3.77}$$

and therefore to obtain a value for γ that tends to 1 it is necessary to use a fluid with a high value of constant volume specific heat, given generally by complex molecule fluids as said before.

As show in [22], the approximated entropy jump across a shock given by

$$\Delta s = -\left(\frac{\partial^2 P}{\partial v^2}\right)_s \frac{(\Delta v)^3}{12\,T} + \mathcal{O}\left((\Delta v)^4\right) = \frac{c^2}{6\,v^3\,T}\,\Gamma\,(\Delta v)^3 + \mathcal{O}\left((\Delta v)^4\right). \tag{3.78}$$

Therefore, across a shock near the transition line (where $\Gamma \approx 0$), as proved in [29], the first term is comparable to $\mathcal{O}\left((\Delta v)^4\right)$, i.e smaller than the PIG

case, leading to a better efficiency of ORC engines. Furthermore, this aspect plays in advantage of the isentropic full potential formulation when capturing shocks near the transition line.

Still referring to equation (3.78), as stated in [30], it is possible to see that where $\Gamma < 0$, given the convexity of the entropy function (that implies $\Delta s > 0$), compression shocks cannot exist and necessarily split into fans. Conversely expansion shock are physically admissible, and such an example is presented in the case of section §6.3.2.

Another interesting phenomenon is the existence of mixed shock–fan waves. Since the inversion zone has a relatively limited extension, it is highly possible that, during a transformation, a particle of fluid enters or exits it. In this case it is possible to observe a compression or an expansion through a 2–component wave (a fan adjacent to a shock) or a three component wave (fan–shock–fan or shock–fan–shock).

3.5 Other gas models

In litterature there exist many more complicated models such as the Martin–Hou, the Soave–Redlich–Kwong or the multi parametric ones. A possible efficient strategy to implement those models in the 2–fields full potential solver is to generate a precomputed thermodynamic table that gives the isentropic enthalpy in function of the fluid density. In problem (2.23), the function $h_{is}(\rho)$ is then substituted by the function $\bar{h}_{is}$ that is a piecewise linear interpolation of this table. To post process the data it could be necessary to have even other tabulated thermodynamic functions such as the pressure or the speed of sound. In figure 3.2 it is shown the error in function of the number of interpolating points.

The convergence is second order since the interpolation is piecewise linear, but the meaning of these graphics is to show that with a relatively small number of interpolating points, the relative error measured in L_2 norm is small enough to give acceptable solutions. When this strategy was adopted in the numerical simulations of this work, 5000 interpolating points were used, that is both cases of PIG or PvdW gas give an error of the order of 10^{-6}.

This approach, besides allowing the usage of every thermodynamic model, speeds up the program execution because instead of many heavy nonlinear computations such as exponentiations, it requires only the computing of products and sums to carry out the linear interpolation. For example, on a mesh with 34 662 elements the speed up (on CPU) obtained using the tabulated functions instead of the exact ones was 2.8x.

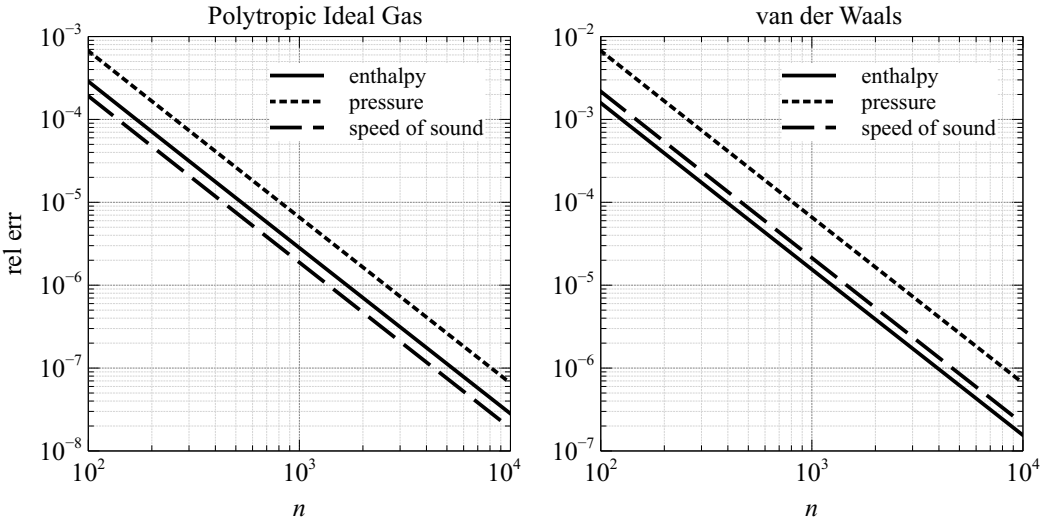


Figure 3.2: Thermodynamic functions interpolating errors. The PIG case is computed in rarefied gas conditions while the PvdW case is computed near the critical point.

Chapter 4

Parallel computing

Nowadays, thanks to the Internet, it is possible to find a lot of tutorials and articles that deal with parallel computing. It is even possible to find some guides that explain how to build a little cheap computer cluster connecting together some normal computers in a LAN network. What follows, is mainly taken from [31].

One of the main targets of this work is the parallelization of the aerodynamic solver ExPReS and the analysis of the speedup of the parallel version compared to the serial version.

Nowadays parallel computing represents a fundamental aspect in numerous research fields: physics, engineering, biology, economy, medicine. In these sectors many problems are intrinsically parallelizable: numerical example are vector sum and matrix multiplication. However, the growing interest in parallel computing in recent decades, both on software and hardware sides, has been particularly affected by the reaching of technological limits of serial architectures: after the race to the GHz of the CPUs in the 2000s, the technological and economical focus has moved to the multi–core processors approach. The technological limitations associated with serial processors are mainly concerned with the fact that the information in the CPU propagates at a speed of the order of light speed. It is therefore necessary to reduce the distance between its elements to increase the CPU performances . However there are physical and technological limits to the miniaturization of the elements inside the CPU. These limits are such that it is more economically advantageous to increase computing performance by focusing on parallel architectures.

There are several advantages in using parallel computers:

- time saving: first of all, throwing more resources at a task will shorten its time to completion, with potential cost savings;

- problem dimension: many problems are too large for their resolution with a single serial computer;
- concurrency capabilities: with a parallel architecture it is possible to do multiple things at the same time;
- distributed resources: in a parallel architecture it is possible to distribute tasks to resources that are not physically close.

Today, parallel architectures are practically everywhere, from supercomputers to small clusters, from personal computers to smart–phones.

4.1 Parallel architectures

4.1.1 Classification

One classification used in parallel computing is the Flynn's taxonomy [32]. In the Flynn's taxonomy parallel architectures are classified on the basis of their capabilities to manage instructions and data streams. Thus, there are two possible states for each type of stream: single or multiple. Four combinations are possible in this taxonomy: SISD (Single Instruction Single Data), SIMD (Single Instruction Multiple Data), MISD (Multiple Instruction Single Data), MIMD (Multiple Instruction Multiple Data).

SIS This is basically the simple serial architecture. In this architecture there is a single instruction stream and a single data stream. The serial version of ExPReS is based on this architecture. The execution is deterministic. Figure 4.1 shows the SISD approach: data A and B are loaded from memory, then C is computed and stored back.

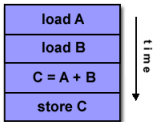


Figure 4.1: SISD approach [31].

SIMD In this parallel architecture there is a single instruction stream but there are multiple data streams. This means that it is possible to process multiple data at the same time but only with the same operation: each computational unit applies that operation on its data stream. Implementations of this type of parallel architecture inside modern CPUs are for example the

SSE (Streaming SIMD Extensions) and the AVX (Advanced Vector eXtensions) instruction sets extensions for CPUs. Even the architecture of the GPU can be seen as an example of SIMD. Figure 4.2 shows the SIMD approach: the same instruction stream (load, multiplication and store) is performed on different data sets (A(i), B(i), C(i) with i = 1 ... n).

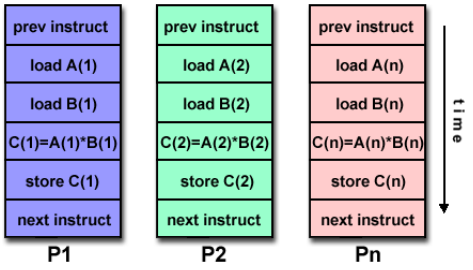


Figure 4.2: SIMD approach [31].

MISD In this parallel architecture there is a single data stream and multiple instruction streams: each processing unit operate with its instruction stream on data shared all the processing units. This means that each processing unit can generate different results from the shared data stream. This kind of parallel architecture is rarely used. Figure 4.3 shows the MISD approach: on the same data A(1) different multiplications are performed.

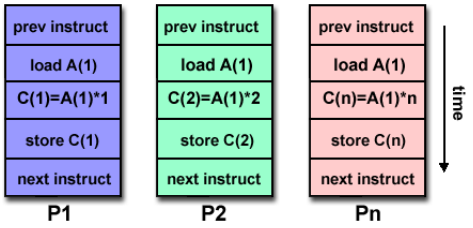


Figure 4.3: MISD approach [31].

MIMD This is the most popular parallel architecture: there are multiple data streams and multiple instruction streams. This means that different processing units can operate simultaneously with different instructions on their own data. The execution in this case can be non–deterministic: the order in which the different processing units finish their work is not defined. Examples of this architecture are multi–core CPUs where while a core is carrying out an operation on its data, another core can simultaneously perform a different operation on different data. Modern supercomputers and clusters use this architecture. Figure 4.4 shows the MIMD approach: completely

different instructions streams are performed in n different threads P1, P2 ... Pn.

The MIMD architecture can be further divided in SPMD(Single Program Multiple Data) and MPMD (Multiple Program Multiple Data).

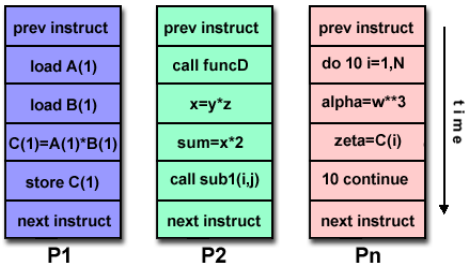


Figure 4.4: MIMD approach [31].

SPMD In this architecture different processors executes the same program on different data but, unlike the SIMD architecture, different processors may be in different points of the program at the same time, performing different instructions. When programming with OpenCL and GPUs, the architecture incorporates features from SIMD and SPMD. In this case NVIDIA called it SIMT (Single Instruction Multiple Thread) [33]. What happens is that GPU's threads are grouped together (in warps, explained in section §4.4) in order to execute the same instruction of a given program (kernel), similarly to what happens with the SIMD approach. However, the SIMT approach makes programming easier than the SIMD approach, avoiding the programmer to explicitly introduce some measure to execute the program properly.

MPMD In this architecture different processors executes differents programs on different data at the same time.

4.1.2 Parallel performances assessment

Parallel programs are more complex than serial programs because the programmer must deal with multiple data and multiple instruction streams. Mainly two performance indicators are used to assess benefits of parallel computing: Speed up measurement and scalability capabilities. It should be kept in mind that both of them depend on the software–hardware combination and not only on the program itself.

Speed up The primary goal of parallel computing is the reduction of the simulation time but this may lead to additional resources requirements.

In fact, in parallel applications, some hardware and software resources are needed to organize and distribute work between processors and in order to allow communications between them.

One of the most important concept in parallel computing is the Amdahl's law [34]. It states that, given the parallel fraction of the code P, given the serial fraction of the code S, and given the number of processors N, the potential program speedup of the parallel version in respect to the serial version is:

$$\text{speedup} = \frac{1}{\frac{P}{N} + S}$$

This means that, for example, if only 50% ($P = 0.5$) of the code can be parallelized it is impossible to obtain speedups greater than 2, not even with an infinite number of processors. Figure 4.5 shows the speed–up trend with different serial and parallel fractions.

Scalability Scalability is the ability of a problem and its solution algorithm to handle a growing amount of work efficiently. There are two types of scaling related to the solution time. In the strong scaling the total problem size is fixed as more processors are added: this means that it is possible to reduce the solution time by increasing the number of available processors. In the weak scaling the problem size per processors stays fixed that means that it is not possible to reduce the solution time but in the same time it is possible to solve more cases or a bigger problem by increasing the number of available processors. The type of scaling depends on both the hardware and the software used in the simulation.

4.1.3 Portability

Thanks to standardization in several API (Application Programming Interface), such as MPI, Posix Threads, OpenMP and OpenCL, nowadays portability issues with parallel programs are not as serious as in the past years. However, despite the existence of standards for APIs, implementations may be different. In some cases the differences in implementations require changes in the code to ensure portability. Portability can also be affected by hardware architectures.

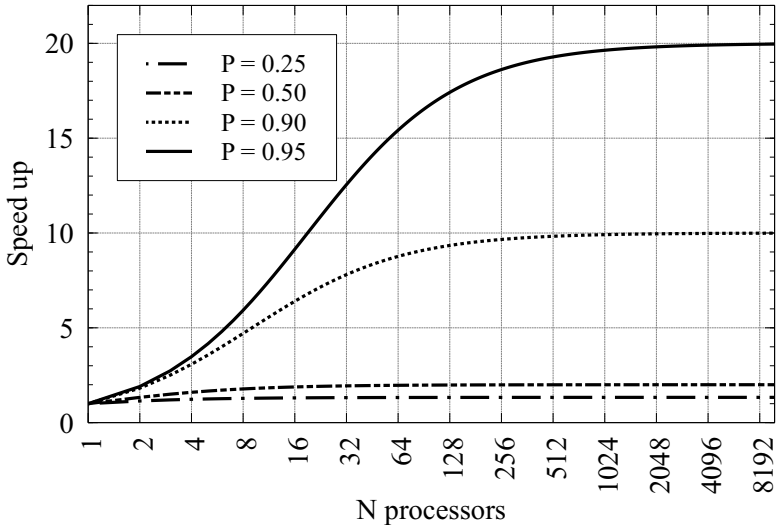


Figure 4.5: Speed–up trend.

4.1.4 SIMD instructions

Nowadays CPUs, even low–budget ones, have multiple cores that allow the processing of various data with different instructions at the same time: this is an example of MIMD approach. Moreover, each core is also able to perform the same operation on multiple data simultaneously using a set of SIMD instructions. That means, for example, that if the programmer writes a code that is capable to take advantage of SIMD instructions, it is possible to sum two vectors of 4 floats (i.e. single precision floating point numbers in C notation) with a single instruction using a single core. Actually there are many available SIMD instruction sets in modern CPUs, such as MMX, 3DNow!, SSE, SSE2, SSE3, SSE4, AVX, AVX2 and many others.

With the first version of the SSE the programmer can operate on four 32-bit single–precision floating point numbers simultaneously. With the second version, SSE2, the capabilities of the original SSE were extended introducing, for example, the possibility to operate on two double–precision floating point numbers simultaneously. Figure 4.6 shows the usage of the SSE to perform the same operation on multiple data at once.

When programming with SSE there are two choices. One choice is to explicitly tell the compiler how to use the SSE instructions. This strategy allows the programmer to have a low–level control of the use of the SIMD units in the CPU but it implies a greater effort to obtain an efficient implementation. The other choice is to leave to the compiler the task of generating the code needed to take advantage of SSE instructions. This approach is often called auto–vectorization and it is implemented in the GCC compiler

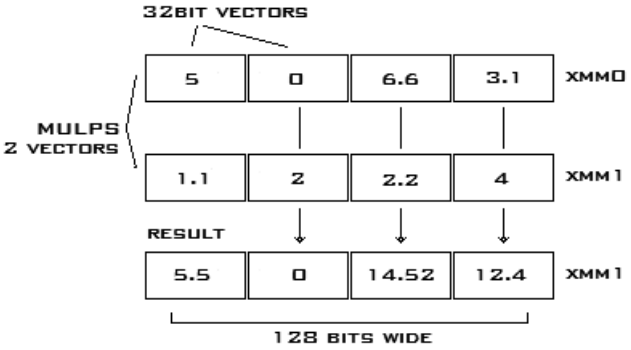


Figure 4.6: Using SSE to multiply four single–precision floating point numbers at once [35].

where this feature can be activated by the following compiler flags:

```
-O2 -msse -msse2 -ftree-vectorize
```

In this work it was decided to activate the auto–vectorization flags of GCC and leave the compiler the task of vectorizing for loops wherever possible.

4.1.5 Shared memory systems and multithreading

Shared memory parallel computers processors have the ability to access all memory as a global address space. In this architecture, processors can perform different tasks but share the same memory resources. This means that if one processor modifies a variable in the shared memory, the change is visible to all the other processors. There are two important type of shared memory systems: UMA (Uniform Memory Acess) and NUMA (Non Uniform Memory Access).

UMA The UMA shared memory systems are often called SMP (Symmetric Multiprocessor). In this architecture all processors have equal access priorities and access time to the shared memory. When processors use cache memory, a memory coherence problem arises. In fact if a processors updates a shared variable that is temporary stored in its cache memory, that change will be only visible to that processors, but the same variable stored in the shared memory will remain outdated. This problem is solved by the CC-UMA (Cache Coherent UMA). Cache Coherent systems use an hardware mechanism to ensure that all processors will see the updated variable. Examples of this architecture are the earlier dual–core dual–processors computers. Figure 4.7 shows the topology of a UMA system.

During this work some dual–core and quad–core single–processor computers that follows the SMP architecture were used. In fact each core of the single CPU shares the system RAM with a SMP approach. This is the architecture used for the execution of the multithreaded version of ExPReS.

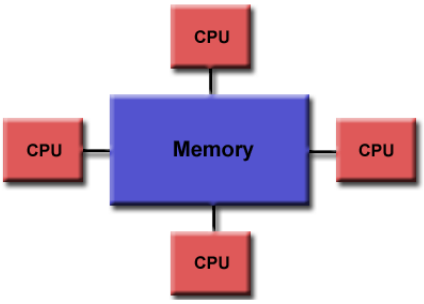


Figure 4.7: UMA system [31].

NUMA The NUMA architecture is made by phisically linking more SMPs. Each SMP in this topology has a local memory that can be accessed quickly. However each SMP can access to the local memory owned by other SMPs using an interconnect bus, but access times to other local memory becomes longer. The name NUMA arises because of these differences in access times. Examples of this architecture are the modern multiprocessor desktops where each multi–core processors is associated with some RAM banks but can access other banks using high–speed serial bus such as Intel QPI or HyperTransport. In CC-NUMA systems the cache coherency is maintained. In figure 4.8 it is shown the topology of a typical NUMA system.

The main advantages of the shared memory systems are the fast data sharing between different tasks and a user–friendly perspective to the memory given by the global address space. One of the main disadvantages of this architecture are the lack of scalability between CPUs and memory: adding mode CPUs increases traffic on shared memory–CPU path and the solutions of this problem can be too expensive. Another disadvantage of this architecture is that the programmer has the responsibility to guarantee the correct access to the shared memory from different tasks.

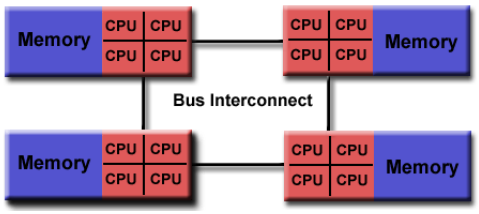


Figure 4.8: NUMA system [31].

Multithreading From the programming point of view it is possible to take advantage of the shared memory system architecture by using multithreaded programming model.

A thread is the smallest sequence of programmed instructions that can be managed independently by an operating system scheduler. A thread can be seen as a light–weight process: a single heavy–weight process can be subdivided in numerous threads that can be executed independently. Each thread of a process, for example, can be executed by each core in a multi–core CPU simultaneously. Each thread in a process has its own private memory but, following the shared memory architecture, every thread of the same process shares the process resources. Thus, different threads of the same process can communicate using the global shared memory. However, threads of different processes don't share resources and can't communicate directly through the shared memory. In this case different processes can communicate by using a message passing mechanism, such as MPI, typical of distributed memory systems. One of the advantages of the multithread approach over the multiprocess approach is that creating and managing threads in a process has a very low cost compared, for example, to the fork of a process typical of the multiprocess approach. That is because when a new thread is created the process shares with it the shared memory, but when a process is forked it is necessary to duplicate the entire memory allocated by the original process.

The two most important implementation that give the programmer the ability to parallelize the code with the multithreading approach are POSIX Threads (or pthreads) and OpenMP.

POSIX Threads POSIX Threads provide a library with a set of C types and procedure calls. With pthreads the programmer can create and terminate threads in a process by calling specific routines. The programmer, for example, can write a function in the program and then create a thread that execute that function. Pthreads gives the programmer a low–level management of threads, such as function to synchronize threads, mutexes and condition variables.

OpenMP OpenMP is an API that comprises three components: compiler directives, runtime library routines, environment variables. While pthreads uses a library–based approach to multithread parallelism, OpenMP uses a compiler directive–based approach. That means that the programmer has to tell the compiler how to parallelize the code mainly by using compiler directives instead of calling functions and creating objects. One of the main advantages of OpenMP API is that it is very easy to use: the programmer can incrementally parallelize its serial program by adding few compiler directives and leaving to the compiler the task to create the code that distributes the work among different threads. However the programmer must explicitly instruct the compiler how to behave when different tasks have to write to a shared resource, for example by using a critical region. The programmer can also synchronize threads by using barriers.

In listing 4.1 and 4.2 it is shown an example that explains how easy it is to parallelize a for loop in C using OpenMP:

Listing 4.1: serial for

```
for(int i = 0; i < size; i++)
{
        c[i] = a[i] + b[i];
}
```

Listing 4.2: OpenMP parallel for

```
#pragma omp parallel for
for(int i = 0; i < size; i++)
{
        c[i] = a[i] + b[i];
}
```

What happens during the execution of a program written with OpenMP directives is that when the program arrives to the parallel part of the code, the process is subdivided in two or more threads and the total work is subdivided and assigned to different threads. Figure 4.9 exemplifies the OpenMP model of parallel execution.

In the particular case shown in listing 4.2 when the process reaches the parallel region, it creates some new threads that share vectors a, b and c. The variable i, the index of the loop, is private to each thread and its value differs from thread to thread. That means that each thread works on different elements of the vectors a, b and c. Hence there is no possibility for one thread to write to the same memory position where another thread is reading. In more complex situations, such as when multiple threads need to write to the same location in the shared memory, it is needed a mechanism that allows only one thread to write at once to avoid unexpected behaviors. For this purpose OpenMP gives the programmer the possibility to explicitly indicate that a set of instructions in the code has to be executed by only one thread at once. This is done using the #pragma omp critical directive.

One important thing is that the OpenMP approach for executing for loops

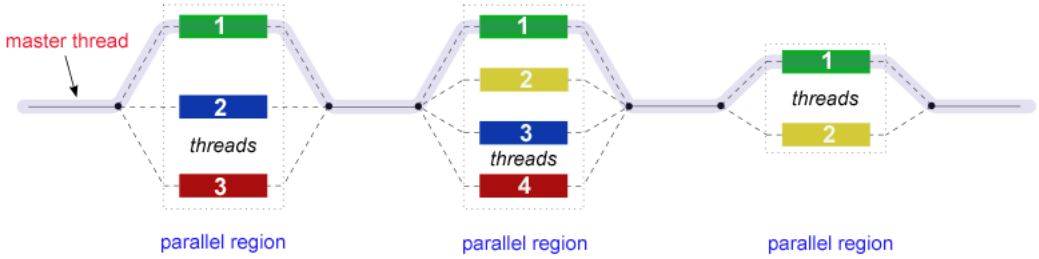


Figure 4.9: The fork–join model of OpenMP [31].

in parallel can be combined with the use of SIMD instruction (such as SSE), eventually by using the compiler autovectorization, to further accelerate the execution.

The first parallel version of ExPReS, the CPU multithreaded version is heavily based on OpenMP. Since the aerodynamic solver uses an explicit time advancing scheme where the update of the solution on one cell needs only to know the solution at the previous time step, it is natural to subdivide the updating task between threads. This means that at every time step a loop similar to the one shown in listing 4.2 is executed, but instead of adding two vectors the solvers updates the solution on each cell of the domain.

For a deepening of the OpenMP specification the reader is referred to [14].

4.1.6 Distributed memory systems and message passing

In this parallel architecture there is not any global shared memory and each processor can access directly only its local memory. This means that any change to the local memory of a processor is not visible by another processor. Thus in this architecture the concept of cache coherence is meaningless. However different processors can communicate to each other using a communication network to connect inter–processor memory. Usually communication networks used in distributed memory supercomputers and smaller clusters are based on technologies such as optical fiber, Infiniband or even the simple Ethernet. One important aspect in this architecture is that the programmer must explicitly define how and when processes executed on different processors can communicate to each other. This is done typically using a message passing mechanism such as MPI. Figure 4.10 shows schematically the distributed memory architecture.

There are some advantages and disadvantages in using a distributed memory architecture. The most important advantage is the scalability of the sys-

tem with respect to the distributed memory approach: it is possible to easily increase the number of total processors and the total memory by connecting new nodes to the network. Another advantage is that when different nodes don't need to communicate to each others they only need to access to their own local memory, thus avoiding any cache coherence problem.

There are two important disadvantages of this architecture in respect to the shared memory one. The first one is that the programmer must explicitly take care of communications between processes and the second is that communications between processors through the network slow down the entire execution.

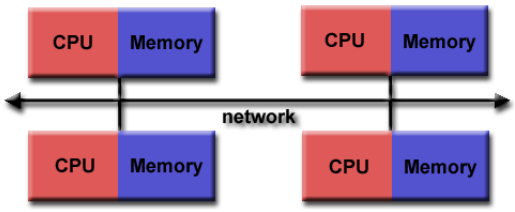


Figure 4.10: Topology of a distributed memory system [31].

MPI MPI (Message Passing Interface) is the "de facto" standard for message passing. There exist various implementations, such as OpenMPI and MPICH. MPI implementations provide types and a library of functions to the programmer. These functions allow to initialize and terminate the MPI environment and to manage the message passing between processes. However care must be taken only on the definition of how and when a communication between different processes is needed. The effective communication between processes on different processors, or even between different hardware architectures, is entirely managed by the implementation. For example some implementations such as OpenMPI use the SSH protocol to connect different node of the network. Since MPI is usually used in a distributed memory system, the parallel application has to support parallelism through a multiprocess approach instead of a multithread approach. This means that when the application is launched and the MPI environment is initialized the main process is forked in multiple processes which are distributed among different processors (or at least among different cores of the same CPU). Since every process generated from the master one has an uniquely identifier inside the program, each process can execute different operations on different data by going through different branches (MIMD).

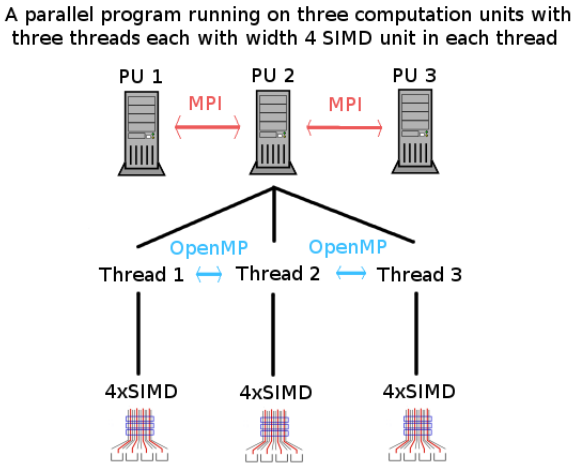


Figure 4.11: Programming with hybrid architecture [36].

4.1.7 Hybrid architecture

This architecture is based on the fusion of the distributed and the shared memory architecture. The main idea is to have multiple shared memory systems connected to each other through a network that follows the distributed memory topology, allowing the communications between different shared memory systems. This idea is used to exploit the advantages of both distributed and shared memory architectures. Nowadays small clusters and the fastest supercomputers are built with this scheme. The actual trend is to use both CPUs and GPUs in each shared memory system that compose the network to improve the performances of applications that can take advantage from the GPU computing. Figure 4.12 sketches the typical topology of a hybrid architecture.

From the programming point of view one of the best way to exploit the advantages of this architecture is to combine the use of MPI, OpenMP and even SIMD instructions as shown in figure 4.11.

The idea is to distribute tasks between each node in the distributed memory system network by using MPI and exploit the shared memory system advantages inside each node by using OpenMP. Moreover each core of each CPU inside a node can take advantage of the SIMD instruction parallelism. Each node can have one or more GPUs.

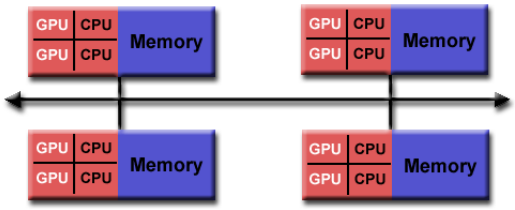


Figure 4.12: Hybrid architecture [31].

4.2 GPGPU computing

GPGPU (General Purpose Graphic Processing Units) indicates the ability of the modern GPUs to perform computations that are not strictly related to graphical rendering. In the last decade the two most important GPUs vendors, AMD and NVIDIA, spent a lot of money and time to develop products dedicated to heavy scientific computations. It is now possible to buy some GPU models that are specifically designed to perform general purpose computation, such as NVIDIA Tesla and AMD FireStream.

Nowadays almost every new GPU model, even a cheap one, has the ability to perform general purpose computations. However one of principal limitation of this technology is often the lack of software that can take advantage of the GPU computing. Even the fastest supercomputers in the world use a hybrid architecture that combine CPUs and GPUs. This approach allows great speedups using GPUs when the particular problem that must be solved allows that. However to exploit the highly parallel architecture of the GPUs the programmer must explicitly write a code that can be compiled and executed on those particular devices.

Before the birth of the actual GPU programming languages such as CUDA C and OpenCL C, it was still possible, by using OpenGL for example, to program a graphic card to perform computations that are not strictly related to visual rendering. However the direct use of OpenGL led to very complex codes. In fact the idea behind the first approaches to GPGPU, was to map the parallel computations needed for the problem to the parallel computations related to the visual elaboration of pixels and vertexes [6]. Now two of the most important GPGPU API and programming languages, CUDA and OpenCL, facilitate the writing of code by using syntaxes that are very similar to the other classical programming languages for CPUs such as C, Python and Fortran. OpenCL C, for example, is based on the C99 standard. However it must be underlined that to exploit the real potential of the GPGPU, the programmer that uses OpenCL and CUDA must have some knowledge of the typical GPU architecture. Recently a new programming

language for GPGPU called OpenACC has appeared: the main advantage of this programming language is that its syntax is very similar to the OpenMP one. This means that any programmer with a little knowledge of the GPUs architecture and memory model can easily adapt its code to be executed on the GPU with the simple addition of preprocessor's directives. Currently there are few compilers that supports OpenACC.

Using a GPU instead of a CPU for scientific computations can reduce of the execution time even by some order of magnitude. However the speedup that can be achieved with GPUs strongly depends on the nature and on the dimension of the problem to be solved. The causes of this behavior are strictly related to the differences between the CPU and the GPU architecture. GPUs in fact are composed by hundreds or thousands little cores that make them specially suitable for problems with very high data parallelism. There are many numerical problems that exhibit this type of parallelism and that can take advantage of the typical GPUs architecture. Vector sum and matrix multiplication, applied for example to aerodynamic and structural solvers are only few examples. On the contrary, when a problem exhibits mainly a task parallel type of work subdivision, the best way to improve performance is usually the execution on multiple CPUs.

4.3 OpenCL

OpenCL (Open Computing Language) is a framework to write programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other accelerators. This standard was originally proposed by Apple, and is now maintained by the Khronos Group consortium, supported by many important companies such as Intel, AMD and NVIDIA. Both CUDA and OpenCL provide the programmer an API (OpenCL API and CUDA API) and a programming language for the device (OpenCL C and CUDA C). They are very similar but the most important difference between the two APIs and languages is that while CUDA is compatible only with NVIDIA GPUs, OpenCL is an open standard implemented by many vendors. This means that a program written with OpenCL can run not only on GPUs but also on CPUs and accelerators with a proper OpenCL implementation. Thanks to the better compatibility of OpenCL, in this work it was decided to use this language to implement a GPU version of the solver ExPReS. For a more accurate comparison between OpenCL and CUDA APIs and languages see [37].

4.3.1 Host code and Device code, compilation and runtime

In the OpenCL and CUDA jargon the CPU is called host while the GPU is called device. The program executed on the device is composed by one or more kernels. To make a comparison with the C language, a kernel is a sort of function that is executed on the device exclusively. When programming with the OpenCL API and OpenCL C the programmer writes two codes. One is for the host and the other one, that contains the kernels, is for the device. The host program is written in normal C/C++, it includes the OpenCL header file (usually cl.h) and calls functions provided by the OpenCL API. The program for the device is written in the OpenCL C language, a language derived from the C99 standard with some restrictions. For example the standard input/output implementation (printf) lacks and consequently there is no possibility to print data on the standard output during the execution of the kernel. However the fact that OpenCL is based on C99 means that a programmer with experience in C can easily write code in the OpenCL C language.

Once the source code for the host and for the device have been written the next steps are the compilation and linking. The compilation of the host code is done in the same way as usual and can be easily performed with GCC. During the linking of the compiled objects, it is necessary to tell the compiler to link the OpenCL library. Since now the source code written for the device has not yet been compiled. In fact its compilation takes place at runtime. When the compiled and linked host program is executed, it calls the OpenCL just–in–time compiler that compiles the device code allowing its execution on the device.

The host and the device are usually physically separated on the motherboard and connected with the PCI–Express bus. Thus also the device memory and the host memory are physically separated. This means that when the host submits commands like the execution of a kernel on the device, the host and the device have to communicate to each other. Then, once the kernel and its arguments are loaded from the host memory to the device memory, the kernel can execute its instructions and access its own memory without interactions with the host. Data transfers between the host and the device are accomplished through the PCI–Express bus that is much slower if compared to the link between the host and its memory or between the device and its memory. When programming with OpenCL, it is thus important to minimize the communications between the host and the device. In fact the order of magnitude of the transfer speeds are: $\sim 10\,Gbit/s$ for the PCI–Express bus, $\sim 100\,Gbit/s$ for the link between the GPU and the device

memory. The simplest strategy is to structure the host code in three steps:

1. transfer all the input data from the host to the device
2. launch all the kernels needed for the simulation minimizing communications between the host and the device, i.e. try to reduce the read back of intermediate results
3. read back the final results from the device to the host when the execution on the device is completed

4.3.2 Parallel approach

Thanks to the many–cores GPUs architecture it is possible to execute the same kernel for hundreds or thousands threads concurrently. Since each thread has an unique ID, the idea is to use this identifier to allow different threads to read from different data location, execute different instructions and write to different locations. With this model the job of the programmer is simplified because he can work as if he was programming one thread. This idea is better explained in listing 4.3, where it is shown the implementation of the vector sum already discussed in section §4.1.5.

Listing 4.3: OpenCL vector sum

```
__kernel void vecAdd(__global float *a,
                     __global float *b,
                     __gloabl float *c)
{
        /* Firstly the global ID of the thread is obtained */

        int i = get_global_id(0);

        /* The programmer can think as he is programming one
           thread, so there are no for loops: each thread
           works on its elements of the three vectors */

        c[i] = a[i] + b[i];

        /* The problem of the vector's dimension is solved by
            launching the kernel with a total number of
           threads equal to the vector's dimension */

}
```

In OpenCL terminology threads are called work–items and are grouped in work–groups organized in NDRanges. In CUDA these concepts correspond to threads, thread blocks and grids. Each work–item in a NDRange has its unique global ID but it has also a work–group ID inside its work–group. Work–items of different work–groups in the same NDRange can have the same work–group ID but their global IDs are different.

4.3.3 Memory model

The host memory is the memory associated with the CPU (i.e. RAM memory). The device memory associated with the GPU in the OpenCL terminology can be subdivided in private memory, local memory, global memory and constant memory. Each work–item has a private memory accessible only by it and thus invisible by other work–items. Work–items in the same work–group shares a memory that is called local memory (or shared memory in the CUDA terminology). Since the local memory is only accessible by work–items in the same work–group, work–items of different work–groups cannot share data through the local memory. However work–items from every work group and from every kernel shares a memory that is called global memory (both in the OpenCL and CUDA terminology). Constant memory is implemented in the global memory, and it is used for a faster access to constant variables shared between all the work–items in a NDRange. However constant memory is associated to a particular kernel and thus a work–item from a kernel cannot access to the constant memory of another kernel. As explained previously, before a kernel can execute its instructions on data, the input data must move from the host to the device through the PCI–Express bus. Data transfer between host and device is strictly related to the concept of buffer (see section §4.3.5). Figure 4.13 shows the device memory topology with OpenCL terminology.

4.3.4 Restrictions

There are some important properties of the GPU memory hierarchy that must be kept in mind. First, the global memory is slower than the local and the private memory even by orders of magnitude. However not all of the algorithms in numerical simulations can be written to take advantage of the local memory. When data are transferred from the host to the device, it can be stored in the global memory only. The local and private memory can be filled with data only by the device as specified in the kernel's instructions. The same occurs when reading back results from the device to the host. It is only possible to transfer data stored in the global memory.

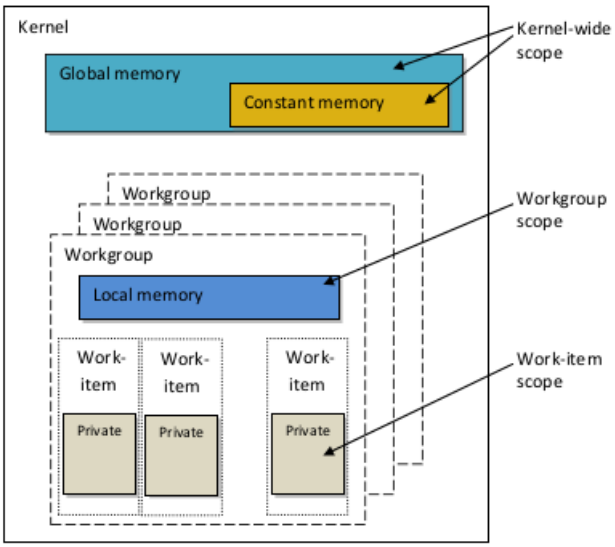


Figure 4.13: OpenCL memory model

Another problem is the memory consistency between the global and the local memory. OpenCL defines a relaxed consistency model. For the local memory, the value seen by work–items inside a work–group is guaranteed to be consistent only at work–group synchronization points. This means that when a work–item inside a work–group writes on a local memory location and then another work–item inside the same work group wants to read from the same location after the write operation, in order to ensure the correct order of the write and read operations, a work–group barrier inside the kernel is needed. The same problem arises with the global memory. A work group barrier is necessary inside a kernel to ensure the correct use of the global memory from different work–items inside the same work–group when load and store operations are performed. In both cases the problem is solved with the use of a synchronization barrier inside the kernel. However this problem can't be solved this way when it interests work–items from different work–groups that try to read and write in the same global memory location. In this case, in order to ensure memory consistency of the device global memory, it is necessary to terminate the kernel. This operation guarantees that all loads and stores from the global memory are completed before moving on with the next kernel (which could be the same as the previous).

Another synchronization problem concerns the communications between the host and the device. Most of the routines in OpenCL are non–blocking. This means, for example, that when the host calls the routine that launches the execution of a kernel on the device, that routine can return before the

device finishes the execution of the kernel. Thus, if the host tries to read the results of the kernel before the kernel end, it may read data that is not yet updated. To avoid this behavior a synchronization point, such as a barrier, is needed inside the host code to ensure that the kernel finishes its execution on the device before the host tries to read back the results.

Another problem with the use of OpenCL on the GPU device that may lead to poor performance is the thread divergence. The explanation of this problem is postponed to section §4.4 because some other important concepts must be introduced in order to better understand this phenomenon.

4.3.5 OpenCL concepts

When programming with OpenCL it is needed to deal with concepts such as platform, context, devices, queue, kernel, buffers etc. These concepts are needed because OpenCL creates an abstracted hardware model that frees the programmer from thinking about the detailed hardware architecture of the target device where the code will execute.

OpenCL platform For an intuitive explanation of what a platform is in OpenCL, it is possible to describe it as a vendor–specific implementations of the OpenCL API. The devices that a platform can target are thus limited to those with which a vendor knows how to interact. In the platform model there is a single host that coordinates the execution of kernels on one or more devices.

OpenCL device A device represents the hardware where the kernels executes. It is possible to have one or more device in the same platform, but, different devices can be included in the same platform only if they are all similar form the OpenCL API implementation point of view. This means for example that if a desktop computer has 2 AMD GPUs, the AMD OpenCL API implementation will see 2 devices in the same platform. On the contrary if inside the system 1 NVIDIA GPU and 1 AMD GPU are installed, two OpenCL API implementations are needed and thus two platforms formed by one device each are seen by the host.

OpenCL context A context defines the environment in which kernels are defined and executed. A context consists of:

- The collection of the OpenCL devices available for the host. These can be CPUs, GPUs or accelerators. Accelerators are hardware specifically designed to perform some computation faster than CPUs.

- A list of kernels that execute on the available devices. It is possible, for example, to run different kernels on different devices concurrently.
- Program objects: device program source code and executables that implement the kernels. Program objects are built from the device source code at runtime within the host program, by calling the OpenCL compiler.
- Memory objects: objects in the device memory only visible and accessible within a kernel that executes on the device. These are explicitly defined on the host and explicitly moved between the host and the OpenCL devices. Memory objects are needed for compatibility reasons because different devices may have different memory architectures.

While different devices from the same platform can stay in the same context, different devices from different platforms cannot stay in the same context. Thus if communications between devices from different platforms are needed, it is necessary to pass through the host.

Platforms, devices and contexts are managed by the host program using functions from the OpenCL API.

OpenCL queue Host and device are connected thanks to a bus such as PCI–Express. Communications with a device occur by submitting commands to a command queue. At least one command queue is necessary for the communication between the host and the device, but multiple command queues for the same device are allowed. A command queue can be in–order or out–of–order. In an in–order command queue, commands are executed by the device in the exact order as they are submitted to the queue. On the contrary, with an out–of–order command queue it is possible for two commands to be executed not int the order they were enqueued. This strategy is useful because it allow the OpenCL implementation to search for commands that can possibly be rearranged and executed more efficiently. The programmer can always force an order between specific commands using the so–called events.

OpenCL kernel As previously anticipated, kernels represent a sort of functions that execute on the device. Kernels are written in OpenCL C, a language derived from the C99 standard with some restrictions such as the lack of recursive functions, pointers to functions, standard libraries. Kernels can call other kernels or functions during their execution on the device. OpenCL C language provides many built–in functions and vector data types that can take advantage of the SIMD architecture.

OpenCL buffer As previously explained, for compatibility reasons, data transfers between the host and the device are achieved through the use of memory object. Memory objects are visible from all the devices in the same context. Communications between devices from different contexts must be managed more explicitly by the host. The most important type of memory objects are buffers and images. During this work only buffers were used. A buffer is a contiguous block of memory made available to the kernels. When an array assembled on the host has to be moved to the device, the programmer has to create a buffer and copy the content of the array from the host to the global memory of the device. After the buffer has been changed by the kernel, the host submits a request to read back the results.

For a deepening of the OpenCL API and OpenCL C specifications the reader is referred to [38], [33], [39], [15], [40], [41], [42].

4.4 Hardware architecture of a GPU

GPUs technology evolves very quickly and even the same vendor introduces numerous changes and improvements between successive architectures. However every GPU architecture has in common a basic structure: they contain hundreds, even thousands, cores and a hierarchy of memory. Thanks to their enormous number of cores, GPUs can be orders of magnitude faster than CPUs when the problem exhibit high data parallelism. Figure 4.14 better explain the gap between GPUs and CPUs floating point performance.

All tests with OpenCL in this work were conducted on the hardware available: NVIDIA GPUs based on the Fermi architecture. Here follows a brief description of the GPUs hardware architecture and focus is given on the Fermi one. Figure 4.15 shows the architecture of a Fermi GPU.

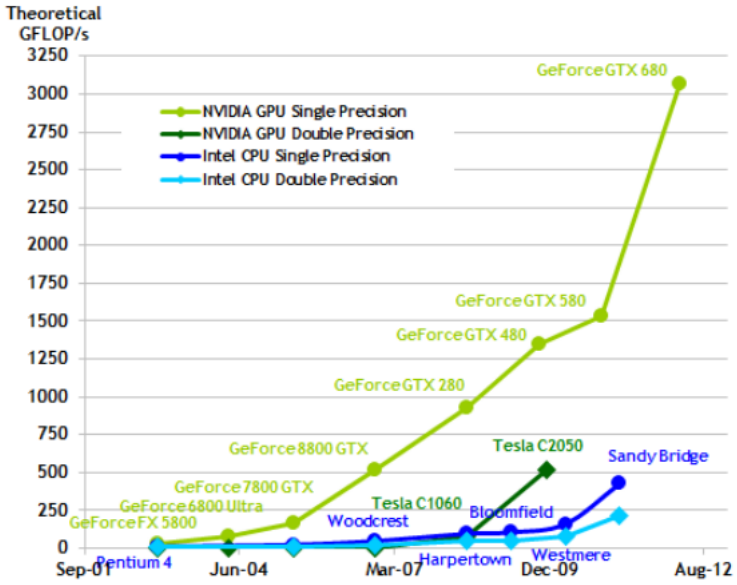


Figure 4.14: trending of CPUs and GPUs performance in GFLOPS

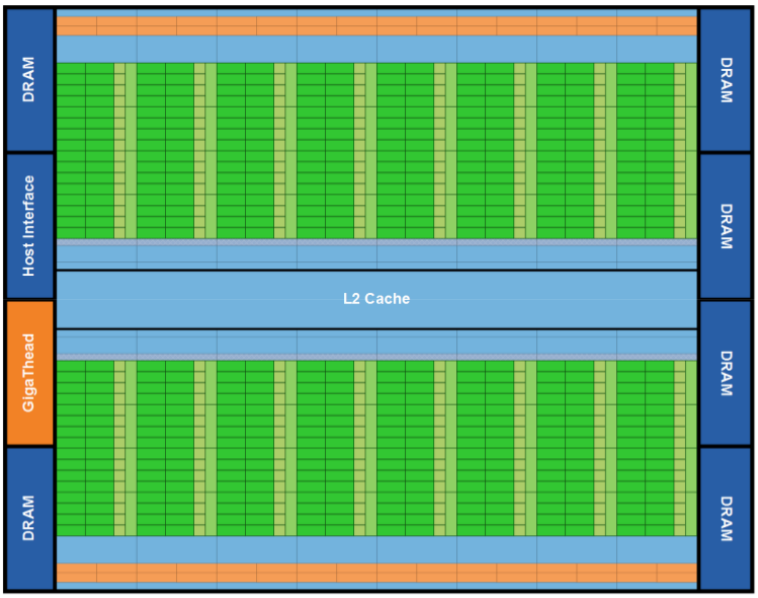


Figure 4.15: Fermi GPU. Fermi's 16 SM are positioned around a common L2 cache. Each SM is vertical rectangular strip that contain an orange portion (scheduler and dispatch), a green portion (execute units), and a light blue portions (register file and L1 cache).

In the Fermi architecture GPU cores, also called CUDA cores in CUDA

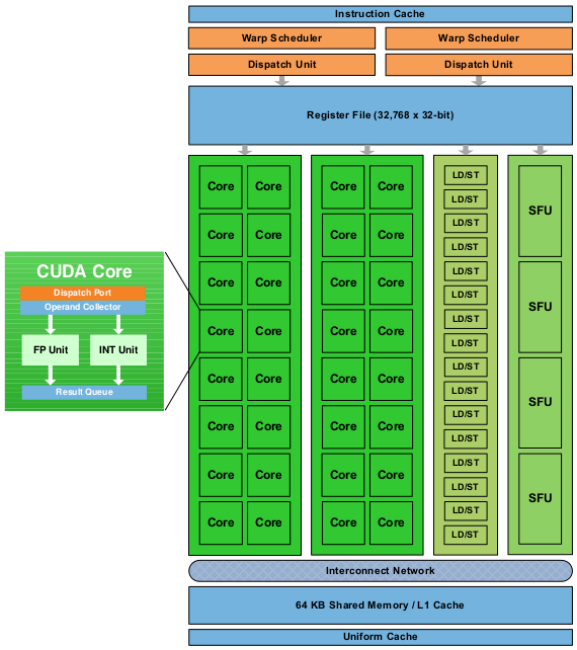


Figure 4.16: Detail of Fermi SM and CUDA core

terminology or Processing Element (PE) in OpenCL terminology, are organized in Streaming Multiprocessors (SM) of 32 core each. In OpenCL terminology, each SM represents a Compute Unit (CU). The host interface connects the host and the device via PCI–Express. GigaThread is the scheduler that distribute thread blocks to each SM scheduler.

Figure 4.16 shows the architecture of a Stream Multiprocessor in detail. Each SM features:

- 32 CUDA cores (or PE), one PE execute one thread. Each CUDA core has a fully pipelined integer arithmetic logic unit (ALU) and a floating point unit (FPU);
- 16 Load/Store units (LD/ST) for the calculation of the source and destination addresses in cache or DRAM;
- 4 Special Function Units (SFU) that execute transcendental instructions such as square root;
- Capability of computing 16 double–precision operations per clock;
- Two warp scheduler and two instruction dispatch units allowing two warp to be issued and executed concurrently: instructions of one warp are issued to a group of 16 CUDA cores, 16 LD/ST units or 4 SFU. Warps are defined later in section §4.4.1;

- 64 KB configurable shared memory (local memory in OpenCL terminology) and L1 cache. This memory can be used as local memory or L1 cache depending on the requests of the kernel in execution. This is an important aspect when a kernel doesn't use local memory. In this case in fact the configurable memory can be set as 48 KB of L1 cache and 16 KB of local memory;

- Registers for the private memory of PEs.

Fermi also featured 768 KB of L2 cache that is accessible from all SMs and that services load and store requests.

The GPU architecture is highly scalable. Since work–groups are distributed among the Streaming Multiprocessors, it is possible to reduce the execution time of a kernel by increasing the number of available SMs on the device. It is thus possible to span a wide market and performance range with the same architecture by simply increasing the number of SMs and the amount of device memory as depicted in figure 4.17.

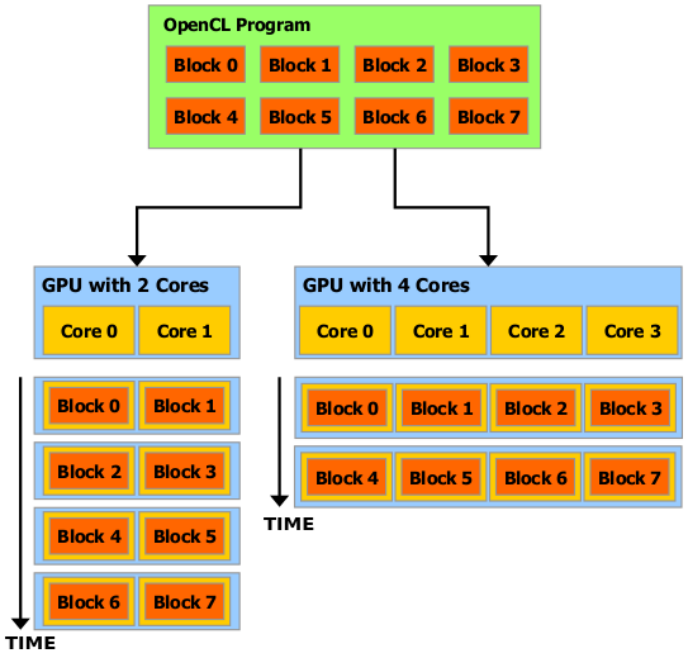


Figure 4.17: GPUs architecture scalability. A multithreaded device program is partitioned into blocks of threads that execute independently from each other, so that a GPU with more cores will automatically execute the program in less time than a GPU with fewer cores.

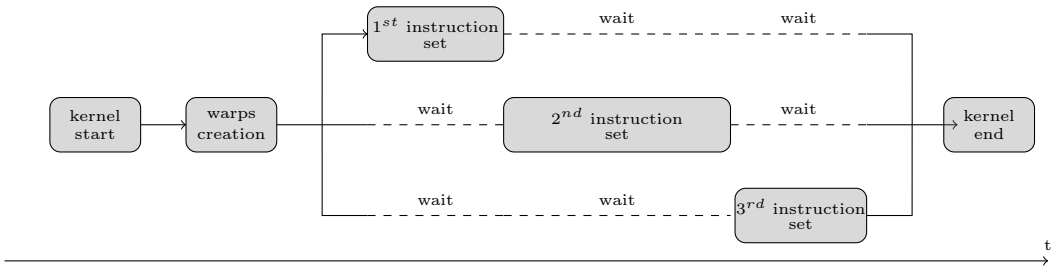


Figure 4.18: Branch divergence.

Fermi revision It must be noted that the GPUs used in this work are based on a revision of the Fermi architecture that introduce some improvement over the original Fermi architecture, such as the increase of the number of CUDA cores in each SM up to 48, as stated in [43]. However, since OpenCL creates an abstraction of the hardware, these differences between hardware architectures are not so critical.

For a deepening of the NVIDIA Fermi Architecture and the newer Kepler architecture the reader is referred to [44], [45], [33]

4.4.1 Branch divergence

When the kernel doesn't contain any conditional statement, each work–item executes the same instructions on a different data thanks to the global ID. This behavior is typical of the SIMD parallel approach. However when different work–items inside a kernel follow different branches they can execute different operations, but all of them are executing the same "program" (i.e. the same kernel). This leads to a SPMD (Single Program Multiple Data) parallel approach. NVIDIA called this SIMD/SPMD approach SIMT (Single Instruction Multiple Threads). When a SM is given a thread block to execute, since the number of thread in a thread block can be bigger than the number of cores in a SM (32 in the original Fermi architecture), thread blocks are subdivided in group of 32 threads called warps that execute concurrently. A warp execute one common instruction at time, so full efficiency is achieved when all 32 threads in a warp follow the same execution path. If threads of a warp diverge due to a conditional branch the warp serially executes each branch path taken, disabling all the threads that are not on that path. When all the paths are completed, all the threads converge back to the same execution path. This serialization of the execution is the cause of the thread divergence performance reduction. It is therefore very important to avoid that different work–items behave differently depending on their global id.

Chapter 5

Numerical discretization

In this chapter the numerical discretization of the 2–fields full potential flow governing equations is introduced.

The general finite volume method is first presented and applied to this particular problem. After this brief introduction, the cell–centered and the node centered formulations are exposed and the implemented choice is discussed in view of the GPU architecture peculiarities. Subsequently some schemes for the reconstruction of gradients are presented and briefly analyzed. The spatial stabilization problem is finally dealt, by presenting some possible upwinding strategies.

Once the problem has been approximated in the space domain, some time discretization schemes are presented. The attention is focused on explicit time stepping schemes to obtain an efficient GPU solver. The integration schemes stability is then investigated by a von Neumann analysis. Those schemes are very suitable for the computing of transient solution, while they result quietly inefficient when the steady state is sought for. Therefore two different strategies to accelerate the computing of stationary solution are presented.

Finally a procedure to impose boundary conditions is described.

5.1 The finite volume discretization

In order to solve the system (2.23) (rewritten here for clarity)

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot (\rho \, \boldsymbol{\nabla} \phi) = 0 \\ \\ \dfrac{\partial \phi}{\partial t} + \dfrac{1}{2} \boldsymbol{\nabla} \phi \cdot \boldsymbol{\nabla} \phi + h_{is} (\rho) = \dfrac{1}{2} V_{\infty}^2 + h_{\infty} \end{cases}$$

In the case of aerodynamic flows with shock waves, the most suitable numerical solution technique is the finite volume (FV) approach. In fact, this technique, on the contrary of the finite element method (FEM), allows to easily deal with discontinuous solutions.

5.1.1 Mass conservation approximation

The first step required to obtain the finite volume formulation, is to convert the differential form of the mass conservation equation to it's integral counterpart. To do that let's integrate it over a fixed control volume V

$$\int_V \frac{\partial \rho}{\partial t} + \int_V \boldsymbol{\nabla} \cdot (\rho \, \boldsymbol{\nabla} \phi) = 0. \tag{5.1}$$

It is underlined that the integral form is valid even in case of discontinuous solution (shocks) while the differential one exists only for differentiable solutions.

Being the control volume not dependent from time, it is possible to pull the time derivative out of the volume integral

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_V \rho + \int_V \boldsymbol{\nabla} \cdot (\rho \, \boldsymbol{\nabla} \phi) = 0. \tag{5.2}$$

Note that the time derivative is now an ordinary and no more a partial derivative because, having integrated $\rho(\boldsymbol{r}, t)$ over the volume V, the quantity $\int_V \rho$ does no more depend on the space coordinate $\boldsymbol{r}$.

The divergence theorem is then applied to the second term in order to find the final form of the integral mass conservation law

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_V \rho + \oint_{\partial V} \rho \, \boldsymbol{\nabla} \phi \cdot \hat{\boldsymbol{n}} = 0. \tag{5.3}$$

To compute an approximation of the scalar field $\rho(\boldsymbol{r}, t)$ it is necessary to solve this equation on a set of small control volumes obtained by a decomposing the whole domain. This decomposition can be obtained tessellating the domain as in figure 5.1 (a more precise definition of the control volumes is given in section §5.2). Meshes used in this work are all Delaunay triangulations obtained with the software Gmsh.

Once the domain has been split, equation (5.3) is locally solved over each control volumes, obtaining a discretized approximation of the solution. To do that, the finite volume method approximates the field $\rho(\boldsymbol{r}_i, t)$ (with $\boldsymbol{r}_i \in V_i$) with the piecewise constant function $R_i(t)$ defined as follows:

$$R_i(t) = \frac{1}{|V_i|} \int_{V_i} \rho(\boldsymbol{r}, t). \tag{5.4}$$

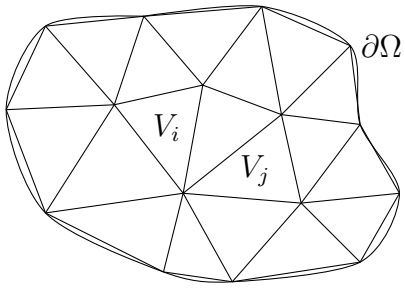


Figure 5.1: Domain Tessellation.

It is highlighted that R_i converges to ρ (if ρ is regular enough) in the limit of the primal elements size approaching 0.

It is therefore possible to write the approximated mass conservation law as

$$\frac{\mathrm{d}R_i}{\mathrm{d}t} = -\frac{1}{V_i} \oint_{\partial V_i} \rho \, \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}}. \tag{5.5}$$

5.1.2 Bernoulli theorem approximation

With regard to the Bernoulli theorem, there is no necessity to integrate it on a control volume because it is a local relation and an integration process will not reduce the order of its derivatives. Exactly as in the case of density, the velocity potential is approximated by a piecewise constant function

$$\phi\left(\boldsymbol{r},\, t\right) \approx \sum_{i}^{N_v} \Phi_i\left(t\right)\ I_i\left(\boldsymbol{r}\right), \tag{5.6}$$

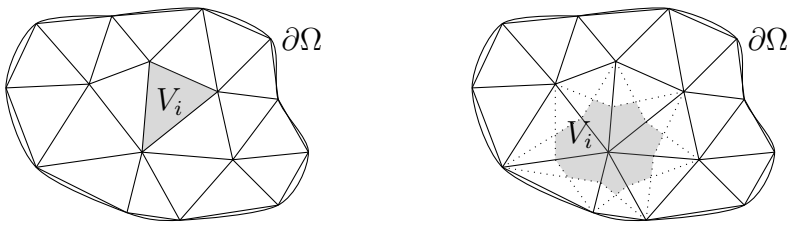
where N_v is the number of control volumes and $I_i\left(\boldsymbol{r}\right)$ is a function defined as follows

$$I_i\left(\boldsymbol{r}\right) = \begin{cases} 1 & \text{if} \quad \boldsymbol{r} \in V_i \\ \\ 0 & \text{elsewhere} \end{cases} \tag{5.7}$$

Introducing this approximation in the Bernoulli theorem a first spatial semi–discretized form is obtained

$$\frac{\mathrm{d}\Phi_i}{\mathrm{d}t} + \frac{1}{2}\ \boldsymbol{\nabla}\phi\,|_{V_i} \cdot \boldsymbol{\nabla}\phi\,|_{V_i} + h_{is}\left(R_i\right) = \frac{1}{2}\,V_\infty^2 + h_\infty, \tag{5.8}$$

where it is evidenced that this is a mixed numerical–analytical expression because of the presence of the term $\boldsymbol{\nabla}\phi\,|_{V_i}$ that will be approximated in



Cell-Centered Node-Centered

Figure 5.2: Cell-Centered and Node-Centered.

section §5.3.1.

5.2 Cell–Centered vs. Node–Centered formulations

One aspect that must be clarified is the definition of the control volume. Two different discretization methodologies are known in the literature. The Cell–Centered (CC) and the Node–Centered (NC).

With reference to figure 5.2, these two schemes differ from how the control volumes are defined. The Cell–Centered method adopts directly the primary elements of the grid (triangles in figure 5.2 left), while the Node–Centered scheme uses a geometric construction based on triangles medians to build the control volumes (see figure 5.2 right).

This last strategy could seem to be unnecessary over–complicated, but as proved in [46], it allows to use many of the finite element techniques in the world of the finite volumes. In fact, in the NC formulation, the unknowns are defined on the triangles nodes. It is thus absolutely natural to interpolate the solution with the same shape functions used in the finite element method. This allows to compute easily all the necessary derivatives at the cell interfaces. Using this shrewdness it is possible to assemble all the residuals and matrices (if present) exactly with the same procedure as it would be done using the finite element method (FEM).

An aspect against the CC formulation is that, as stated in [47], for an arrangement of the cells sketched in figure 5.3, an interface averaging does not provide the correct value at the midpoint of a face even for a linearly varying function. The consequence is that on a grid with slope discontinuity the discretization error will not be reduced even when the grid is infinitely refined. Such zero–order errors manifest themselves as oscillations or kinks in isolines, whereas a node–centered scheme experiences no problems in the

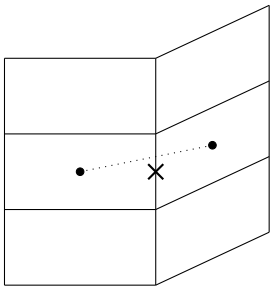


Figure 5.3: Grid with slope discontinuity.

same situation.

GPGPU computing All the aspects illustrated above seem to be in favor of the Node–Centered formulation, but some big disadvantages arise when the solver is going to be implemented on a GPU. Two possible algorithms to solve the 2–fields full potential flow problem in the NC formulation are here presented. They are outlined in figure 5.4 and 5.5 and they are going to be explained in this paragraph later on. These two procedures are presented for the 2D case, since all the peculiar aspects related to GPGPU computing are not different for the 3D case.

Consider the algorithm outlined in figure 5.4. In this procedure two kernels are subsequently enqueued to compute fluxes and to update the solution of the mass conservation equation. The first one is a 1D kernel of dimension equal to the number of triangles. It computes all fluxes on the three interfaces of every triangle. For example, work–item 1, that works on triangle E, computes the fluxes through the tree interfaces $E - M_{1,5}$, $E - M_{1,6}$ and $E - M_{5,6}$, where with $M_{x,y}$ is intended the mean point of the $x - y$ side.

Successively, a second kernel is launched in order to update the solution. It is a 1D kernel with the same dimension of the number of triangles. It updates the value of density on the three nodes of the triangle it's working on summing to them their fluxes. For example, work–item 1, that works on triangle E, updates the value of density on the nodes 1, 5 and 6. Work–item 2, that works on triangle A, updates the value of density on nodes 1, 2 and 6. To do that, work–item 1 must write to the memory positions `rho[1]`, `rho[5]` and `rho[6]` and work–item 2 must write to the memory positions `rho[1]`, `rho[2]` and `rho[6]`. It must be noticed that in this example, more work–items write in the same memory location. Because of the relaxed memory consistency model (explained in section §4.3.3 and in section §4.3.4), this will result in wrong values of density on the nodes at the end of this kernel, making this algorithm not suitable for an implementation on a GPU.

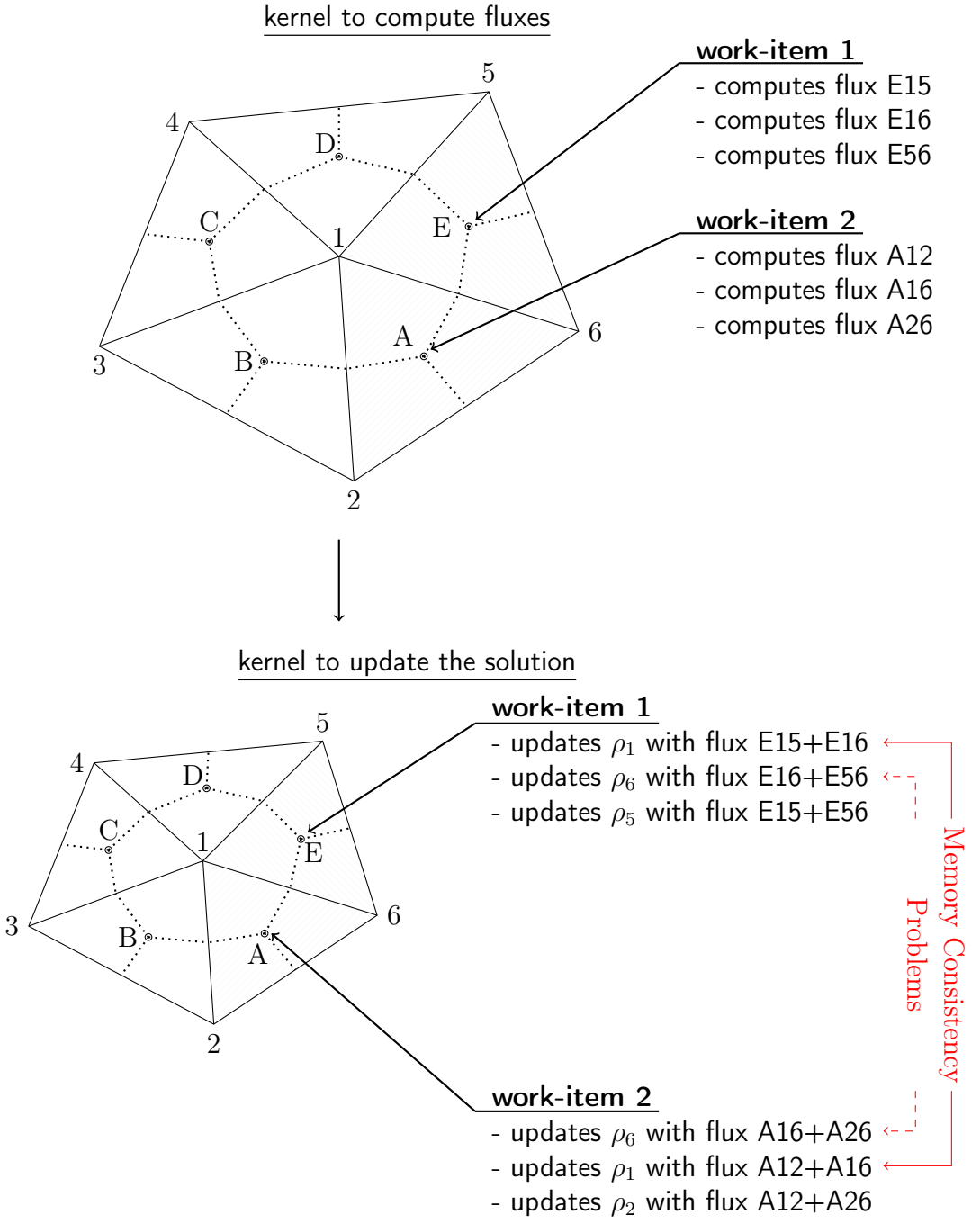


Figure 5.4: Algorithm 1: memory consistency problem.

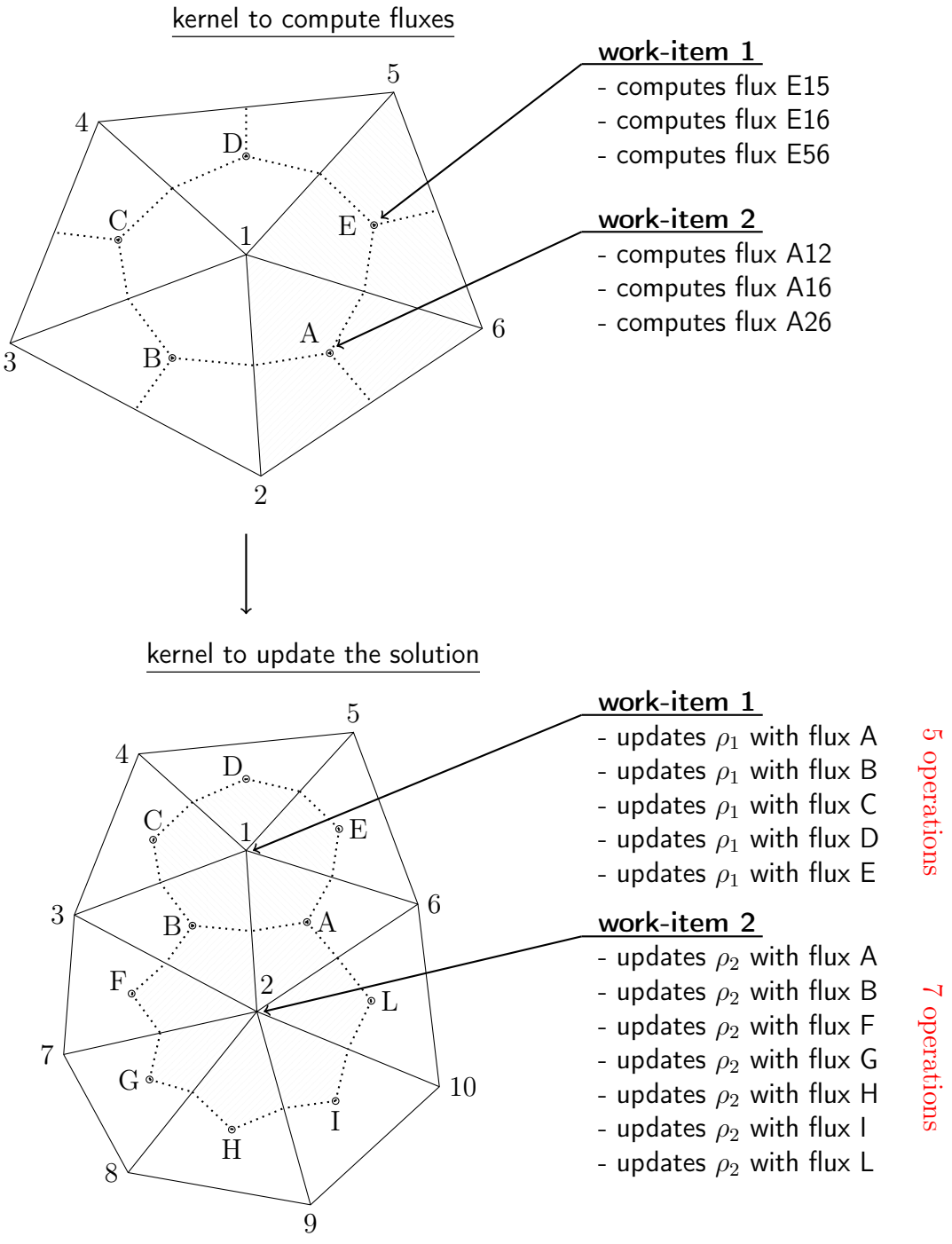


Figure 5.5: Algorithm 2: branch divergence.

Consider now a second algorithm, sketched in figure 5.5, and used in [48]. As for the above procedure, two kernels are subsequently enqueued to compute fluxes and to update the solution of the mass conservation equation. The first kernel is identical to the first kernel of algorithm 1. On the contrary, the second kernel works now on the nodes and no more on the mesh triangles. Every work–item updates the value of its node, therefore avoiding problems with memory consistency. In the example of figure 5.5, work–item 1 updates the value of `rho[1]` and work–item 2 updates the value of `rho[2]`. The problem now is that, to perform this task, work–item 1 executes five operations while work–item 2 executes seven operations. This difference comes from the fact that, given a general unstructured mesh, the number of nodes connected to one node is not equal all over the grid. For example, referring to figure 5.5, the node number 1 is connected to 5 nodes and node number 2 is connected to 7 nodes. This implies that, in the NC formulation, every control volume could have a different number of interfaces, leading to a different behavior of different work–items performing the same task. This aspect is called branch divergence and it is not a problem when parallelizing a code on a CPU, but in GPU programming (see section §4.4.1) it can seriously affects the computing performances, making this algorithm not recommended.

These two algorithms are only two of the many possible implementations of a NC finite volume scheme, but, to the best knowledge of the authors, all other possible strategies lead to the same kind of problems.

Finally, in figure 5.6, a Cell–Centered algorithm is outlined. It consists of two subsequent kernels. The first ones is a 1D kernel of dimension equal to the number of interfaces (internal interfaces only because boundaries are handled separately) and it computes fluxes through them. The second kernel then updates the value of density. It is a 1D kernel of dimension equal to the number of cells and every work–item operates on one cell only. In the example in figure 5.6, work–item 1 updates `rho[E]` and work–item 2 updates `rho[A]`. This algorithm does not suffer from memory consistency issues because every work–item writes only in one memory location. Moreover all of them perform the same number of operations, avoiding any problem of branch divergence.

These two fundamental aspects brought to the choice of the cell–centered scheme. It should be noticed that this approach is quite different from the standard methods adopted to parallelize CFD codes on CPU clusters. In fact the classical way relies on a domain decomposition where the entire volume is split in relatively few sub–domains. These are composed of many cells and they are assigned to different processors. In the GPU approach used in this work, on the contrary, the domain decomposition is much finer, assigning every elementary cell to a single thread.

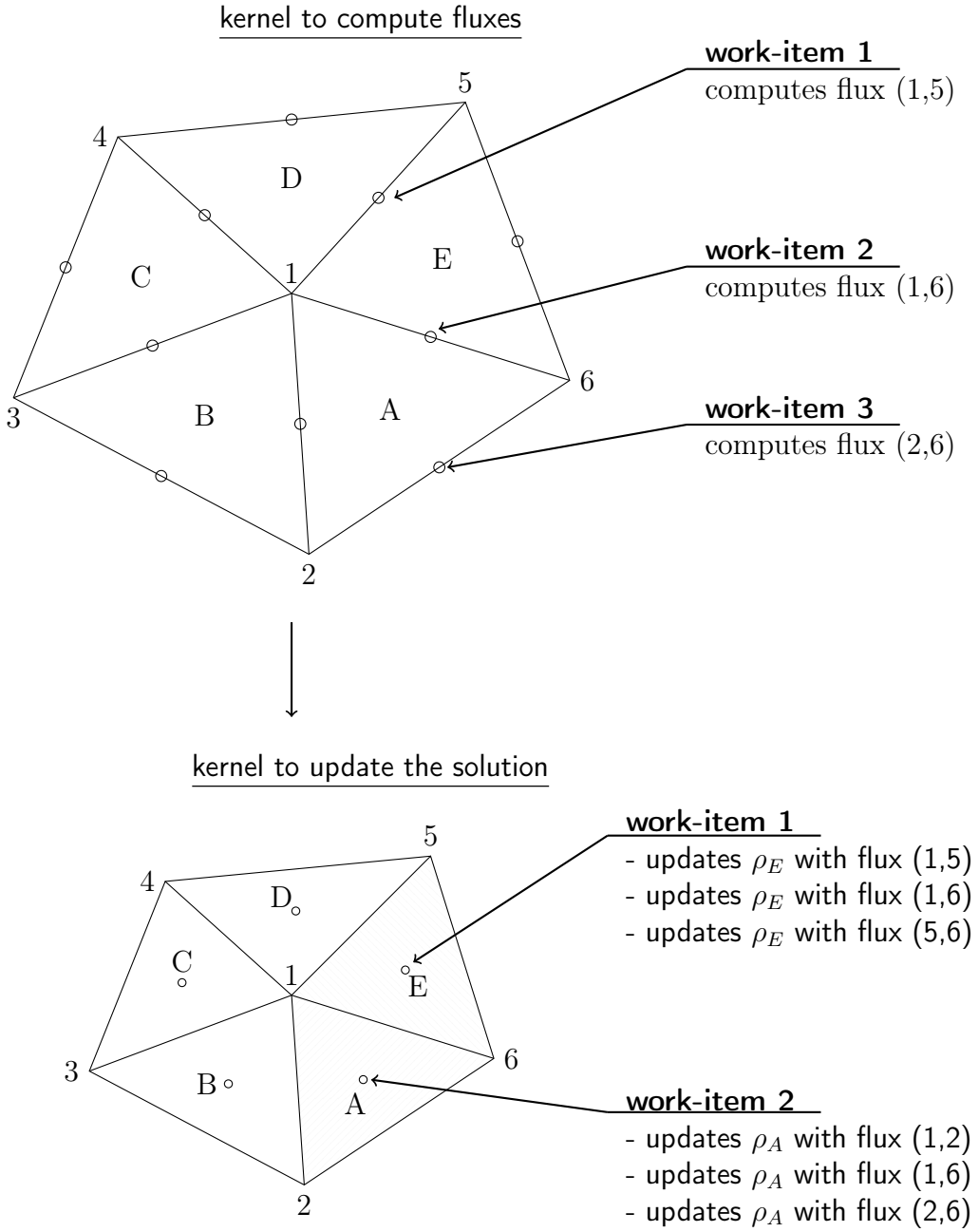


Figure 5.6: Cell-Centered algorithm

5.3 Spatial discretization

To compute the right hand side of equation (5.5)

$$\oint_{\partial V} \rho \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}}, \tag{5.9}$$

it is necessary to introduce a numerical scheme that reconstructs the interface fluid velocity ($\boldsymbol{\nabla}\phi$).

Moreover, the term of the Bernoulli theorem

$$\frac{1}{2} \boldsymbol{\nabla}\phi \cdot \boldsymbol{\nabla}\phi \tag{5.10}$$

must be computed on cells and not on interfaces. Therefore, a numerical procedure to obtain the gradient of the velocity potential over every cell of the computational domain must be introduced.

5.3.1 Gradient reconstruction

Independently from the equation to be resolved, the finite volume discretization schemes usually require to compute the gradient of the numerical solution.

As stated in [49], the choice of the method used to compute the gradients depends at least on two factors. The first is whether the discretization is a Node–Centered or Cell–Centered type. Secondly different algorithms have to be used if the gradient is to be evaluated on the cell centers or on an interface between two cells. Consider a Node–Centered discretization in which an interface gradient reconstruction is needed: the most natural technique is to interpolate the numerical solution using the same basis functions that are used in finite elements method and to differentiate this interpolation to obtain the gradient. This approach cannot however be used to compute the derivatives on the grid nodes because the interpolated solution is not, in general, $\mathcal{C}^1$ between adjacent elements.

As it was discussed in section §5.2, the formulation adopted in this work is the Cell–Centered. Therefore some commonly used gradient reconstruction algorithms are here described and briefly analyzed.

The full potential problem strongly relies on the accuracy of the gradient reconstruction schemes, much more than what is needed in viscous fluxes computing. In fact in a full potential simulation a poor accuracy can easily destabilize the numerical integration, while in viscous fluxes it would imply an error only in the viscous term without deeply affecting the stability.

Cell gradient (C–Gradient) The first method is a direct application of the Green–Gauss theorem

$$\int_{\Omega} \boldsymbol{\nabla} u = \oint_{\partial\Omega} u\, \hat{\boldsymbol{n}} \tag{5.11}$$

that, with reference to figure 5.7, can be used to compute the mean of the gradient of the generic function u over the volume V_i as follows:

$$\text{mean}\left(\boldsymbol{\nabla} u\right)\big|_{V_i} = \frac{1}{|V_i|} \oint_{\partial V_i} u\, \hat{\boldsymbol{n}}. \tag{5.12}$$

The circuitation integral is computed summing up the product of the out–warding unit vector multiplied by the interface value of u and by the size of the interface. The first problem is how to compute the value of u over the interface and it can be solved taking the linear interpolation between the two cell centers. In the 2D case where cells are build from a triangulation of the domain, as in figure 5.7, this discretization results in the following reconstructed cell gradient $\boldsymbol{G}_u^C$

$$\boldsymbol{\nabla} u\,|_{V_i} \approx \boldsymbol{G}_u^{C_i} = \frac{1}{|V_i|} \sum_{j=1}^{N_f} \bar{U}_{i,j}\, \hat{\boldsymbol{n}}_{i,j}\, \Delta L_{i,j}\,, \tag{5.13}$$

where N_f is the number of faces of a cell, $\bar{U}_{i,j}$ is the linear interpolation of u along the line connecting the two cell centers straddling interface j, $\hat{\boldsymbol{n}}_{i,j}$ is the out–warding unit vector and $\Delta L_{i,j}$ is the size of the interface.

A difficulty arises when a face of the cell lies on a boundary. In that case the flux is not known from the solution alone. A possibility is to consider an extern ghost cell with a zero order extrapolation value for the quantity u or to use, if known, the boundary conditions. However, in the implementation relative to this work, this scheme did not prove to have the necessary accuracy to obtain a stable numerical scheme.

The second method examined is a Least Square interpolation of the solution over the cell. Usually a polynomial fit is computed on a stencil of points surrounding the cell. Using a linear interpolation a first order convergence scheme is obtained but, depending on the least square method used, different levels of accuracy can be achieved. In fact using a weighted least square (WLSQ) procedure gives a smaller error than using a simple unweighted least square (ULSQ) approach. However the choice of the weighing factors is not straightforward and in order to obtain an improvement in accuracy those parameters are strongly dependent from grid characteristics such as the triangles aspect ratio.

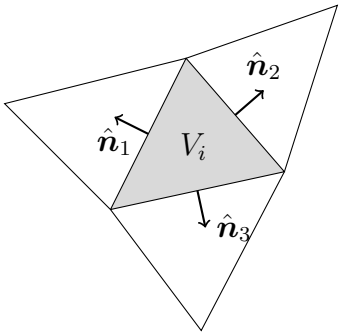


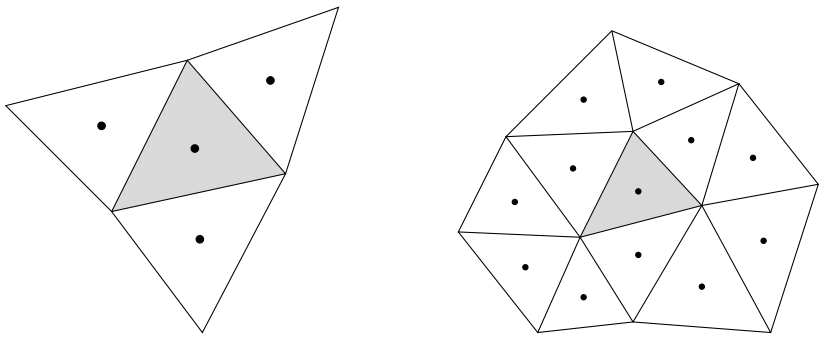
Figure 5.7: Cell-Centered C-Gradient Green Gauss theorem

Two example stencils are shown in figure 5.8. To compute the linear interpolation, the simple stencil uses all the cells sharing a face with the primary cell, while the augmented stencil uses all the cells that share with it at least one node. Obviously it is possible to adopt every intermediate approach between the two that have been presented here, but it is necessary to define an appropriate criterion to chose the interpolating points.

Moreover, the choice of the stencil should rely not only on the desired degree of accuracy, but even on algorithm constraints. In fact it should be noticed that the augmented stencil implies that there is not a fixed number of points to deal with. This aspect causes a different behavior of the function that computes the gradients depending on the nodes connectivity, leading straightforwardly to the branch divergence phenomenon when the solver is implemented on a GPU. Instead, the simple stencil is not affected by this problem. For this reason the simple stencil is the one adopted in this work.

The last technicality is how to proceed if a cell has a face on the boundary. In this case it is possible to use an outside ghost cell with a zero order extrapolation of the solution or to simply reduce the stencil to tree points instead of four. In this last case there is no more the necessity to solve a problem in the least square sense, because the linear interpolating function is strictly determined knowing its value in three different points. For a deeper explanation on gradient reconstruction stencils the reader is redirected to [50].

The last numerical scheme presented here is the node averaging (NA) technique. It is based on a reconstruction of the solution at the nodes from the local neighboring cells by a weighted mean procedure. The simplest node average is performed by an area weighted mean, but in literature many other more accurate schemes exist as the one presented in [51, 52] that is based on an averaging procedure subjected to some laplacian optimization constraints. It must be pointed out, however, that, as proved in [53], this



simple stencil augmented stencil

Figure 5.8: Least Square C-Gradient stencils

NA technique is not different from a Least Square one. Once the node averaging has been performed, the usual node centered techniques can be used to compute gradients.

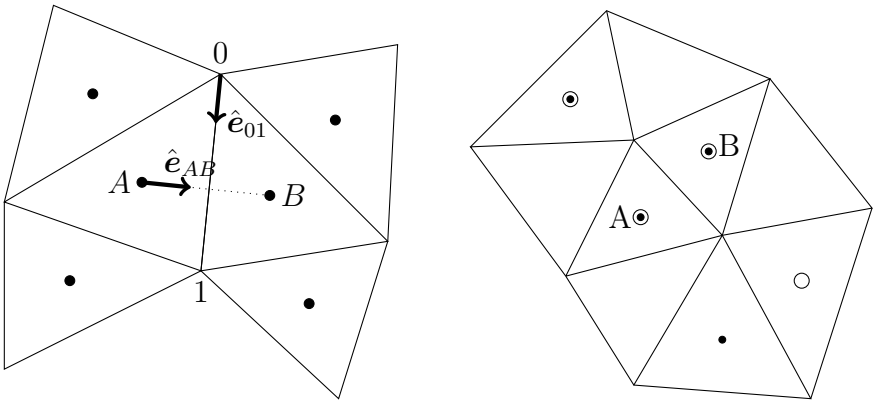
This last technique was not even taken into consideration because the node mean depends on how many elements are connected to the master node on which the mean is going to be computed. Being this number not equal on every grid node in the case of unstructured meshes, this approach would cause the very undesired branch divergence phenomenon in the kernel that computes gradients.

Face gradient (F–Gradient) The simplest way to evaluate the gradient of a variable on an interface is to take the mean of the C–Gradients on the two neighboring cells. In order to obtain a higher accuracy on stretched grids, it should not be used the arithmetic mean, but a weighted one computed using as weights the distances of the cells centroids from the interface.

Other algorithms directly compute the gradient over the cell face solving a small system in the least square sense. In this case the problem is split in two phases. The first one approximates the directional derivative of the solution along two (in the 2D case) linearly independent directions. Referring to figure 5.9, these two directions are:

- $\boldsymbol{\nabla} u \cdot \hat{\boldsymbol{e}}_{AB}$: directional derivative along the line connecting the two centroids of the cell A and B;
- $\boldsymbol{\nabla} u \cdot \hat{\boldsymbol{e}}_{12}$: directional derivative along the interface between cell A and B.

In a second phase, system (5.14) is solved in the least square sense in order to obtain the reconstruction of the gradient on the face $\boldsymbol{G}_u^F = \left(G_{u\,x}^F,\ G_{u\,y}^F\right)^T \approx$



(a) (b)

filled circle: 1^{st} possible stencil, empty circle: 2^{nd} possible stencil

Figure 5.9: F-Gradient reconstruction

$$\left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right)^T$$

$$\begin{cases} G^F_{u\,x}\, e_{AB\,x} + G^F_{u\,y}\, e_{AB\,y} = \boldsymbol{\nabla} u \cdot \hat{\boldsymbol{e}}_{AB} \\ G^F_{u\,x}\, e_{12\,x} + G^F_{u\,y}\, e_{12\,y} = \boldsymbol{\nabla} u \cdot \hat{\boldsymbol{e}}_{12} \end{cases} \tag{5.14}$$

Every cell centered method uses a simple centered difference formula to reconstruct the derivative across the face ($\boldsymbol{\nabla} u \cdot \hat{\boldsymbol{e}}_{AB}$)

$$\boldsymbol{\nabla} u \cdot \hat{\boldsymbol{e}}_{AB} = \frac{U_A - U_B}{|\vec{r}_A - \vec{r}_B|}. \tag{5.15}$$

On the contrary, many methods are available to compute the derivative along the face. They are all based on a polynomial least square approximation of the solution. Once it has been computed, it's gradient is multiplied by the unit vector in the face direction so as to obtain the term $\boldsymbol{\nabla} u \cdot \hat{\boldsymbol{e}}_{12}$.

Different interpolation techniques can be used. They differ from each other for the choice of the computational stencil that goes from a set of nodes comprising all the cell centroids represented in figure 5.9 (a), to a reduced stencil of just four points composed of the two cells sharing the interface and other two cells sharing with them a node but not an interface, as shown in figure 5.9 (b).

It's moreover possible to opt for a weighted or unweighted least square approximation. In the first case a natural choice for the weighing factors is to

take the inverse of the distance between the cell barycenters and the center of the interface.

In the implementation of ExPReS the right balance between computational efficiency and accuracy was achieved using the simple stencil for C–Gradient reconstruction and averaging those value to obtain the gradient approximation on interfaces. In fact, since the gradients over the cell are anyway required by the Bernoulli theorem, the face mass flux was computed using the average method, that proved to be accurate enough and computationally efficient.

5.3.2 Spatial stabilization: upwind

In order to guarantee the stability of the numerical scheme, as will be further demonstrated in section §5.4.4, it is necessary to introduce some dissipation in the integration method. This can be done by upwinding the flux at the cell interface, taking a value for the density that is not its exact value on the interface, but its value in a point slightly upstream in the wind direction. This strategy can be seen just as a way to ensure the numerical stability, but it's not all that. In fact it has a physical meaning, that is the representation of the spatial causality in advection dominated flows. Physically speaking, in potential flows the upwinding should be necessary only in supersonic regions, where a point can be affected only from its upstream region inside its Mach cone. However, if an explicit time stepping technique is adopted, it's anyway necessary to upwind the interface density value in order to obtain a numerically stable algorithm.

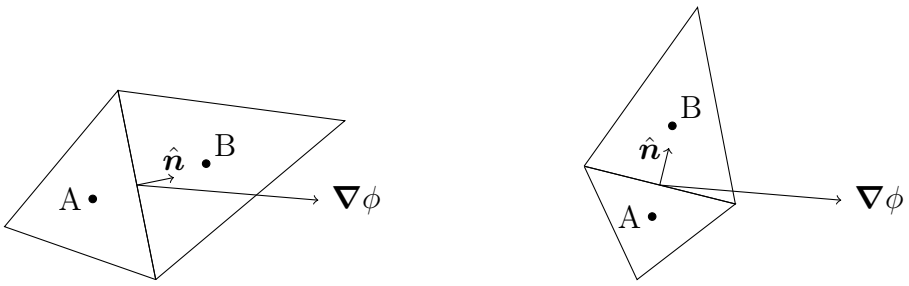
There are many ways to upwind the value of a variable in a point and here some different methods are presented. The first one is the simplest and consists of taking the density value of the upstream cell of the interface. Referring to figure 5.10, it means to use ρ_A to compute the flux through the interface in both cases (a) and (b).

This algorithm could be implemented as described below;

- compute $\boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}}$;
- enter in the function of figure 5.11 (a) to obtain the value of the upwinded density $\bar{\rho}$.

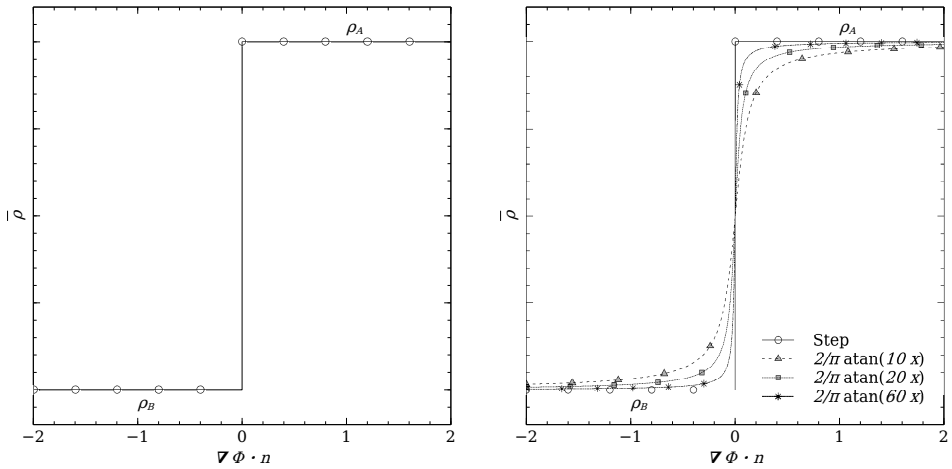
It easy to understand that this strategy works greatly for the case of figure 5.10 (a), but it doesn't in the case (b).

This first algorithm can then be improved using one of the functions of figure 5.11 (b) to compute the flux density value. It could be useful to have a function that can change it's slope in function of a parameter, in order to adjust the dissipation as needed.



(a) (b)

Figure 5.10: Upwinding. First technique



(a) (b)

Figure 5.11: Upwinding function

A more sophisticated upwinding technique was presented for example in [5], and consists of taking as density value the quantity

$$\bar{\rho} = \rho + \Delta\rho, \tag{5.16}$$

where

$$\Delta\rho = \boldsymbol{\nabla}\rho \cdot \boldsymbol{l} \tag{5.17}$$

and $\boldsymbol{l}$ lies in the wind direction and its modulus is a mix of a user adjustable parameter ε and of the size of the local cell h_i

$$|\boldsymbol{l}| = \varepsilon\, h_i\,. \tag{5.18}$$

In an implicit scheme, where no dissipation is needed in subsonic regions, this approach allows to upwind the solution only where needed

$$\boldsymbol{l} = \begin{cases} 0 & if\ M < 1 \\ \\ \varepsilon\, h_i & if\ M > 1 \end{cases} \tag{5.19}$$

This technique is less dissipative than the ones presented above, due to the fact that it dissipates flow perturbations only in the streamline direction and not in the transverse wind direction. Therefore it is more convenient to use this approach when computing transient solutions when high time accuracy is desired, while it could be better to use the first technique when the stationary solution is desired.

5.3.3 Numerical flux

Once the gradient and the upwinded density have been computed, it is possible to proceed with the approximation of the right hand side of equation (5.5)

$$\oint_{\partial V_i} \rho\, \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}} \approx \sum_{j=1}^{N_f} \mathcal{F}_{i,j} = \sum_{j=1}^{N_f} \bar{\rho}_{i,j}\ \boldsymbol{G}_\Phi^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j}\ \Delta L_{i,j}\,, \tag{5.20}$$

where the numerical flux $\mathcal{F}_{i,j}$ through interface F_j has been introduced

$$\mathcal{F}_{i,j} = \bar{\rho}_{i,j}\ \boldsymbol{G}_\Phi^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j}\ \Delta L_{i,j}\,. \tag{5.21}$$

The space discretized form of the mass conservation equation is thus obtained

$$\frac{\mathrm{d}R_i}{\mathrm{d}t} = -\frac{1}{V_i} \sum_{j=1}^{N_f} \bar{\rho}_{i,j}\ \boldsymbol{G}_\Phi^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j}\ \Delta L_{i,j}\,. \tag{5.22}$$

5.3.4 Space discretized 2–fields potential flow problem

It is now possible to give a full meaning to equation (5.8), having now better specified the term $\boldsymbol{\nabla}\phi\,|_{V_i}$.

The space–discretized Bernoulli theorem finally results

$$\frac{\mathrm{d}\Phi_i}{\mathrm{d}t} + \frac{1}{2}\,\boldsymbol{G}_\Phi^{C_i} \cdot \boldsymbol{G}_\Phi^{C_i} + h_{is}\,(R_i) = \frac{1}{2}\,V_\infty^2 + h_\infty. \tag{5.23}$$

Thus the space discretized 2–fields potential flow problem is

$$\begin{cases} \dfrac{\mathrm{d}R_i}{\mathrm{d}t} = -\dfrac{1}{V_i}\displaystyle\sum_{j=1}^{N_f} \bar{\rho}_{i,j}\ \boldsymbol{G}_{\Phi}^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j}\ \Delta L_{i,j} \\ \dfrac{\mathrm{d}\Phi_i}{\mathrm{d}t} + \dfrac{1}{2}\,\boldsymbol{G}_{\Phi}^{C_i} \cdot \boldsymbol{G}_{\Phi}^{C_i} + h_{is}\left(R_i\right) = \dfrac{1}{2}\,V_{\infty}^2 + h_{\infty} \end{cases} \tag{5.24}$$

This is a system of two equations in two unknowns. To obtain a full solvable problem, boundary and initial conditions must be specified. They will be discussed in section §5.5.

5.4 Time discretization

To discretize the problem in the time domain it is possible to choose among various methods. Some time stepping schemes are briefly presented with an outlook on their efficiency, their stability properties and the possibility to efficiently parallelize the solution algorithm.

Consider the generic implicit initial value problem (5.25)

$$\begin{cases} \dot{\underline{x}} = \underline{f}\left(\underline{x},\, \dot{\underline{x}},\, t\right) \\ \underline{x}\left(t_0\right) = \underline{x}_0 \end{cases} \tag{5.25}$$

In order to integrate it in the time domain, two main time stepping methods are available:

- explicit time stepping, that can be written as

$$x_{k+1} = \sum_{j=0}^{p} x_{k-j} + \Delta t\ g\left(x_k,\, x_{k-1},\, \ldots\right), \tag{5.26}$$

  where g is a method–dependent function.

  examples: Linear Multi Step (LMS) methods of the Adams–Bashforth family or the explicit Runge–Kutta methods;

- implicit time stepping, that can be written as

$$x_{k+1} = \sum_{j=0}^{p} x_{k-j} + \Delta t\ h\left(x_{k+1},\, x_k,\, \ldots\right), \tag{5.27}$$

  where h is a method–dependent function.

  examples: the LMS method of Adams–Moulton family, the BDF methods or the scheme presented in [54].

Implicit methods have generally a bigger stability region than explicit ones and if the method is A–Stable any value of the time step ensures that the time integration will not diverge due to numerical instability. However this advantage is payed in therms of computational effort. In fact implicit schemes require the solution of a linear system (if $\underline{f}$ is linear) or many systems (if $\underline{f}$ is not linear) at every time step.

On the contrary explicit time schemes do not require the solution of any system, but are limited in the time step amplitude for stability reasons. In the case of the 2–fields full potential formulation, the mass conservation equation has a hyperbolic nature and is subjected to the Courant–Friedrichs–Lewy (CFL) stability condition

$$\Delta t < \min_i \left(\frac{h_i}{c_i} \right), \tag{5.28}$$

where h_i is the cell j dimension and c_i is the speed of sound in the cell i.

This restriction is very limiting when the stationary solution is sough for, because in this case there is no reason to require a high time accuracy but the CFL condition forces anyway explicit methods to proceed with very little time steps. On the other side, the implicit schemes can reach the stationary solution in few iterations of the Newton–Raphson algorithm.

When computing transient solution this disadvantage is no more a problem, because even implicit schemes are forced to use a relatively small time step to achieve the desired accuracy.

The last advantage of adopting an explicit scheme is the easiness and efficiency of parallelization. These methods in fact do not require the solution of a system, but only the computing of residuals. On the contrary, using an implicit method, it would be necessary to parallelize the assembly and the solution of large sparse systems of equations. The best way to do that could be the use of an iterative method such as GMRES or the conjugate gradient algorithm (CG), that require an efficient preconditioner to converge in an acceptable number of iterations. This could certainly be a future development, but up to now it is beyond the scope of the present work. Moreover the implicit formulation is much more memory consuming than the explicit one due to the matrix factorization.

5.4.1 Explicit Euler time integration

The first possible choice to integrate the system (5.24), is to use the explicit Euler method as follows

$$\begin{cases} \dfrac{R_i^{n+1} - R_i^n}{\Delta t} = -\dfrac{1}{V_i} \displaystyle\sum_{j=1}^{N_f} \bar{\rho}_{i,j}^{\,n} \, \boldsymbol{G}_{\Phi^n}^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j} \, \Delta L_{i,j} \\ \dfrac{\Phi_i^{n+1} - \Phi_i^n}{\Delta t} = \dfrac{1}{2} V_\infty^2 - \dfrac{1}{2} \boldsymbol{G}_{\Phi^n}^{C_i} \cdot \boldsymbol{G}_{\Phi^n}^{C_i} + h_\infty - h_{is}\left(R_i^n\right) \end{cases} \tag{5.29}$$

where the apex n indicates the time instant.

This system is then solved for the two unknowns R_i^{n+1} and Φ_i^{n+1}

$$\begin{cases} R_i^{n+1} = R_i^n - \dfrac{\Delta t}{V_i} \displaystyle\sum_{j=1}^{N_f} \bar{\rho}_{i,j}^{\,n} \, \boldsymbol{G}_{\Phi^n}^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j} \, \Delta L_{i,j} \\ \Phi_i^{n+1} = \Phi_i^n + \Delta t \left(\dfrac{1}{2} V_\infty^2 - \dfrac{1}{2} \boldsymbol{G}_{\Phi^n}^{C_i} \cdot \boldsymbol{G}_{\Phi^n}^{C_i} + h_\infty - h_{is}\left(R_i^n\right) \right) \end{cases} \tag{5.30}$$

This solution, as shown in section §5.4.4, has a very small stability region, making this approach unfeasible.

5.4.2 Staggered time integration

The first idea to stabilize the numerical discretization is to use a staggered integration, that is to integrate the first equation in a first step and to use the obtained result to integrate the second one. This idea originated from the semi implicit (or symplectic) Euler method, that integrates the generic system (5.31)

$$\begin{cases} \dfrac{\mathrm{d}v}{\mathrm{d}t} = f\left(t,\, x\right) \\ \dfrac{\mathrm{d}x}{\mathrm{d}t} = g\left(t,\, v\right) \end{cases} \tag{5.31}$$

with the numerical scheme

$$\begin{cases} v^{n+1} &= v^n + \Delta t \, f\left(t_n,\, x^n\right) \\ x^{n+1} &= x^n + \Delta t \, g\left(t_n,\, v^{n+1}\right) \end{cases} \tag{5.32}$$

This scheme is still a first order integrator, but it is symplectic, i.e. it conserves energy[1] (if Δt is constant). This feature gives the semi implicit method better stability and accuracy properties.

In the 2–fields full potential case, the system is not of the same type as that of system (5.31), but it is slightly more complex:

$$\begin{cases} \dfrac{\mathrm{d}R}{\mathrm{d}t} &= f\left(R,\, \Phi\right) \\ \\ \dfrac{\mathrm{d}\Phi}{\mathrm{d}t} &= g\left(\Phi,\, R\right) \end{cases} \tag{5.33}$$

To solve this system with the staggered integrator, the first equation that has to be resolved is the mass conservation equation

$$R_i^{n+1} = R_i^n - \frac{\Delta t}{V_i} \sum_{j=1}^{N_f} \bar{\rho}_{i,j}^{\,n}\, \boldsymbol{G}_{\Phi^n}^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j}\, \Delta L_{i,j} \tag{5.34}$$

and the second one is the Bernoulli theorem,

$$\Phi_i^{n+1} = \Phi_i^n + \Delta t \left(\frac{1}{2} V_\infty^2 - \frac{1}{2} \boldsymbol{G}_{\Phi^n}^{C_i} \cdot \boldsymbol{G}_{\Phi^n}^{C_i} + h_\infty - h_{is}\left(R_i^{n+1}\right) \right), \tag{5.35}$$

where the value of enthalpy at the right hand side is computed using the new value of density.

This method, provided with some spatial dissipation, results to be stable and therefore is the one implemented in the solver ExPReS.

5.4.3 Convergence acceleration techniques

As already mentioned in section §5.4, obtaining the steady state solution with an explicit method is a time consuming activity. In literature many methods exist to accelerate this process at the expense of the accuracy of the initial transitory. In this work two strategies have been tested to speed up the convergence.

Local time stepping As suggested in [47], the first strategy implemented is the *local time stepping* technique. It consists in using the largest possible time step allowed by the local stability constraint on every cell. As reported in [55], this trick should assure that disturbances are propagated at the far field boundary of the domain in a number of time steps of the same order as the number of elements in the radial direction of the mesh.

[1] Symplecticness is a very important aspect, for example, in long time orbit calculations

Relaxation method A simple way to improve convergence is to compute a new solution at every time step based on the predicted solution and on the solution at the previous time step.

This new solution is computed according to the following algorithm

$$\begin{cases} \tilde{R}_i^{n+1} = R_i^n - \dfrac{\Delta t}{V_i} \displaystyle\sum_{j=1}^{N_f} \bar{\rho}_{i,j}^{\,n} \, \boldsymbol{G}_{\Phi^n}^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j} \, \Delta L_{i,j} \\ \tilde{\Phi}_i^{n+1} = \Phi_i^n + \Delta t \left(\dfrac{1}{2} V_\infty^2 - \dfrac{1}{2} \boldsymbol{G}_{\Phi^n}^{C_i} \cdot \boldsymbol{G}_{\Phi^n}^{C_i} + h_\infty - h_{is}\left(\tilde{R}_i^{n+1}\right) \right) \\ R_i^{n+1} = \beta \, \tilde{R}_i^{n+1} + (1-\beta) \, R_i^n \\ \tilde{\Phi}_i^{n+1} = \beta \, \tilde{\Phi}_i^{n+1} + (1-\beta) \, \Phi_i^n \end{cases} \tag{5.36}$$

where β is the relaxation factor:

- $\beta < 1$ leads to under–relaxations;
- $\beta > 1$ leads to over–relaxation.

In figure 5.12 the effect of this scheme on a general oscillating damped signal is presented.

It is possible to see that under–relaxation dumps down oscillations in the solution while over–relaxation accentuates them. In testing, it was found that $\beta > 1$ generally speeds up the convergence to the steady state in detriment of stability while $\beta < 1$ allows to use higher CFL numbers but the steady state is obtained with a greater number of iterations.

A more rigorous stability analysis for this scheme will be carried out in section §5.4.4.

5.4.4 Stability analysis

The stability properties of those scheme are analyzed performing a von Neumann analysis (as in [56]) on the 1D linearized problem for the PIG gas. In order to obtain results that do not depend on the asymptotic condition, a dimensionless formulation is used.

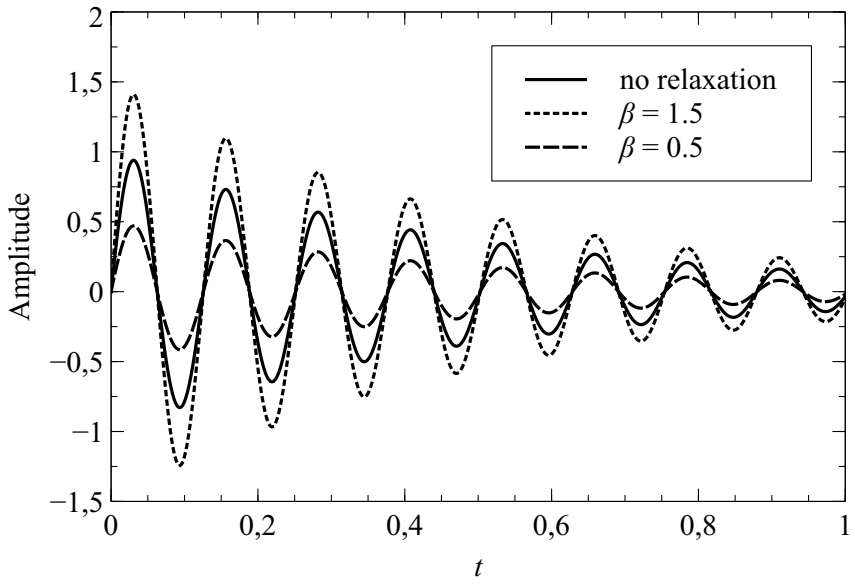


Figure 5.12: Relaxation Method

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \dfrac{\partial}{\partial x}\left[\left(\rho - \varepsilon\, h\, \dfrac{\partial \rho}{\partial x}\right)\, \dfrac{\partial \phi}{\partial x}\right] = 0 \\ \dfrac{\partial \phi}{\partial t} + \dfrac{1}{2}\left(\dfrac{\partial \phi}{\partial x}\right)^2 + h_{is}\left(\rho\right) = \dfrac{1}{2}\, V_\infty^2 + h_\infty \end{cases} \tag{5.37}$$

where h is the space discretization length and ε is the upwind parameter.

Expressing the two unknowns as

$$\rho = \rho_\infty + \delta\rho \tag{5.38a}$$
$$\phi = V_\infty\, x + \delta\phi \tag{5.38b}$$

and substituting equations (5.38) in system 5.37, and neglecting the second order terms, the dimensional linearized formulation is obtained

$$\begin{cases} \dfrac{\partial\, \delta\rho}{\partial t} + \rho_\infty\, \dfrac{\partial^2\, \delta\phi}{\partial x^2} + V_\infty\, \dfrac{\partial\, \delta\rho}{\partial x} - \varepsilon\, h\, V_\infty \dfrac{\partial^2\, \delta\rho}{\partial x^2} = 0 \\ \dfrac{\partial\, \delta\phi}{\partial t} + V_\infty\, \dfrac{\partial\, \delta\phi}{\partial x} + \gamma\, \dfrac{P_\infty}{\rho_\infty^2}\, \delta\rho = 0 \end{cases} \tag{5.39}$$

To convert this system in the dimensionless form the following quantities are introduced

$$\tilde{\rho} = \frac{\rho}{\rho_\infty}, \tag{5.40a}$$
$$\tilde{\phi} = \frac{\phi}{V_\infty L}, \tag{5.40b}$$
$$\tilde{x} = \frac{x}{L}, \tag{5.40c}$$
$$\tilde{t} = \frac{V_\infty t}{L}, \tag{5.40d}$$

where L is a characteristic length of the domain.

A direct application of the chain rule gives

$$\frac{\partial}{\partial t} = \frac{\mathrm{d}\tilde{t}}{\mathrm{d}t}\frac{\partial}{\partial \tilde{t}} = \frac{V_\infty}{L}\frac{\partial}{\partial \tilde{t}}, \tag{5.41a}$$
$$\frac{\partial}{\partial x} = \frac{\mathrm{d}\tilde{x}}{\mathrm{d}x}\frac{\partial}{\partial \tilde{x}} = \frac{1}{L}\frac{\partial}{\partial \tilde{x}}. \tag{5.41b}$$

Substituting relations (5.40) and applying the transformations (5.41), the linearized dimensionless formulation is finally obtained

$$\begin{cases} \dfrac{\partial\,\delta\tilde{\rho}}{\partial\tilde{t}} + \dfrac{\partial^2\,\delta\tilde{\phi}}{\partial\tilde{x}^2} + \dfrac{\partial\,\delta\tilde{\rho}}{\partial\tilde{x}} - \varepsilon\,\tilde{h}\,\dfrac{\partial^2\,\delta\tilde{\rho}}{\partial\tilde{x}^2} = 0 \\ \dfrac{\partial\,\delta\tilde{\phi}}{\partial\tilde{t}} + \dfrac{\partial\,\delta\tilde{\phi}}{\partial\tilde{x}} + \dfrac{1}{M_\infty^2}\,\delta\tilde{\rho} = 0 \end{cases} \tag{5.42}$$

where $\tilde{h} = \frac{h}{L}$. For the sake of simplicity the symbols δ and $\tilde{}$ will not be reported from now on.

This problem is then numerically discretized using centered second order finite differences for the two cases of explicit Euler time stepping and staggered time stepping.

- Explicit Euler

$$\begin{cases} R_i^{n+1} = & R_i^n - \dfrac{\Delta t}{h}\left[\dfrac{R_{i+1}^n - R_{i-1}^n}{2} - \varepsilon\left(R_{i+1}^n - 2\,R_i^n + R_{i-1}^n\right)\right. \\ & \qquad\qquad \left. + \dfrac{1}{h}\left(\Phi_{i+1}^n - 2\,\Phi_i^n + \Phi_{i-1}^n\right)\right] \\ \Phi_i^{n+1} = & \Phi_i^n - \dfrac{\Delta t}{h}\left[\dfrac{\Phi_{i+1}^n - \Phi_{i-1}^n}{2} + h\,M_\infty^{-2}\,R_i^n\right] \end{cases} \tag{5.43}$$

- Staggered

$$\begin{cases} R_i^{n+1} = & R_i^n - \dfrac{\Delta t}{h}\left[\dfrac{R_{i+1}^n - R_{i-1}^n}{2} - \varepsilon\left(R_{i+1}^n - 2\,R_i^n + R_{i-1}^n\right)\right. \\ & \qquad\qquad \left. + \dfrac{1}{h}\left(\Phi_{i+1}^n - 2\,\Phi_i^n + \Phi_{i-1}^n\right)\right] \\ \Phi_i^{n+1} = & \Phi_i^n - \dfrac{\Delta t}{h}\left[\dfrac{\Phi_{i+1}^n - \Phi_{i-1}^n}{2} + h\,M_\infty^{-2}\,R_i^{n+1}\right] \end{cases} \tag{5.44}$$

To perform the von Neumann analysis the generic solution u (assumed periodic) is expanded in the Fourier series

$$u_i^n = \sum_{\omega=-\infty}^{+\infty} \alpha_\omega \mathrm{e}^{\mathrm{j}\omega i h}\left(\gamma_\omega\right)^n, \tag{5.45}$$

where γ_ω is the amplification factor and j the imaginary unit.

A numerical method results stable if

$$|\gamma_\omega| < 1 \quad \forall\omega. \tag{5.46}$$

Explicit Euler stability Substituting (5.45) in the system (5.43), given that the relation (5.46) must hold for every ω, the following relation is obtained for the first time step.

$$\left\{ \begin{array}{c} R_i^1 \\ \Phi_i^1 \end{array} \right\} = \left[A\left(\omega\right)\right] \left\{ \begin{array}{c} R_i^0 \\ \Phi_i^0 \end{array} \right\}, \tag{5.47}$$

where

$$A(\omega) = \begin{bmatrix} 1 - \dfrac{\Delta t}{h}\left[f(\omega h) + 2\,\varepsilon\, g(\omega h)\right] & 2\dfrac{\Delta t}{h^2} g(\omega h) \\ -\Delta t\, M_\infty^{-2} & 1 - \dfrac{\Delta t}{h} f(\omega h) \end{bmatrix}, \tag{5.48}$$

with $f(\omega h) = \mathrm{j}\sin(\omega h)$ and $g(\omega h) = (1 - \cos(\omega h))$.

Applying relation (5.47) recursively, the solution at the generic time t_n is

$$\begin{Bmatrix} R_i^n \\ \Phi_i^n \end{Bmatrix} = [A(\omega)]^n \begin{Bmatrix} R_i^0 \\ \Phi_i^0 \end{Bmatrix}. \tag{5.49}$$

Therefore the numerical method is stable if the modulus of each eigenvalue of $A(\omega)$ is less than 1.

Staggered time integration stability Repeating the same analysis for the staggered time integration the following relations are obtained.

$$\begin{bmatrix} 1 & 0 \\ \Delta t\, M_\infty^{-2} & 1 \end{bmatrix} \begin{Bmatrix} R_i^1 \\ \Phi_i^1 \end{Bmatrix} = [B(\omega)] \begin{Bmatrix} R_i^0 \\ \Phi_i^0 \end{Bmatrix}, \tag{5.50}$$

where

$$B(\omega) = \begin{bmatrix} 1 - \dfrac{\Delta t}{h}\left[f(\omega h) + 2\,\varepsilon\, g(\omega h)\right] & 2\dfrac{\Delta t}{h^2} g(\omega h) \\ 0 & 1 - \dfrac{\Delta t}{h} f(\omega h) \end{bmatrix}. \tag{5.51}$$

In order to have a stable scheme it necessary that all the eigenvalues of matrix (5.52)

$$C(\omega) = \begin{bmatrix} 1 & 0 \\ \Delta t\, M_\infty^{-2} & 1 \end{bmatrix}^{-1} B(\omega) \tag{5.52}$$

have their modulus inferior to 1.

Stability regions In figure 5.13 the stability regions for the forward Euler scheme and for the staggered scheme are presented

It is evident that the staggered method presents advantages both in terms of minimum needed dissipation and maximum allowed CFL.

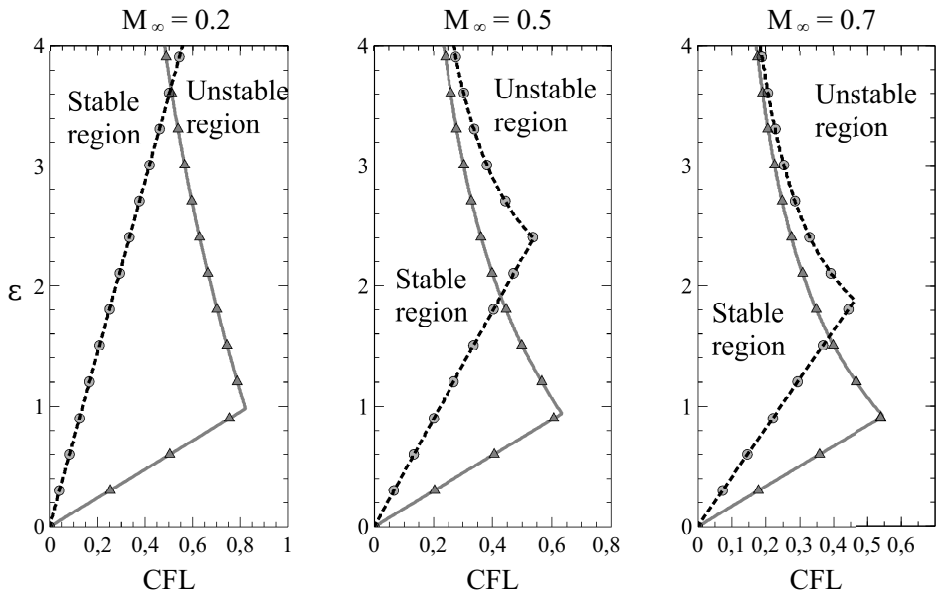


Figure 5.13: Stability regions for forward Euler and staggered methods. Forward Euler: Dashed line with circle markers; Staggered method: Continuous line with triangle markers.

Relaxation method The stability properties of the relaxation method presented in section §5.4.3 are now analyzed. Doing the same steps of the previous paragraphs, the solution at time instant t_1 is obtained from the initial condition as

$$\begin{Bmatrix} R_i^1 \\ \Phi_i^1 \end{Bmatrix} = \begin{bmatrix} 1-\beta & 0 & \beta & 0 \\ 0 & 1-\beta & 0 & \beta \end{bmatrix} \begin{Bmatrix} R_i^0 \\ \Phi_i^0 \\ \tilde{R}_i^1 \\ \tilde{\Phi}_i^0 \end{Bmatrix} = \begin{bmatrix} 1-\beta & 0 & \beta & 0 \\ 0 & 1-\beta & 0 & \beta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ [C(\omega)] \end{bmatrix} \begin{Bmatrix} R_i^0 \\ \Phi_i^0 \end{Bmatrix}, \quad (5.53)$$

where $C(\omega)$ has been defined in equation (5.52).

As before, to obtain a stable integration, relation (5.54) must hold

$$\left| \text{spec} \left(\begin{bmatrix} 1-\beta & 0 & \beta & 0 \\ 0 & 1-\beta & 0 & \beta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ [C(\omega)] \end{bmatrix} \right) \right| < 1 \quad \forall \omega, \quad (5.54)$$

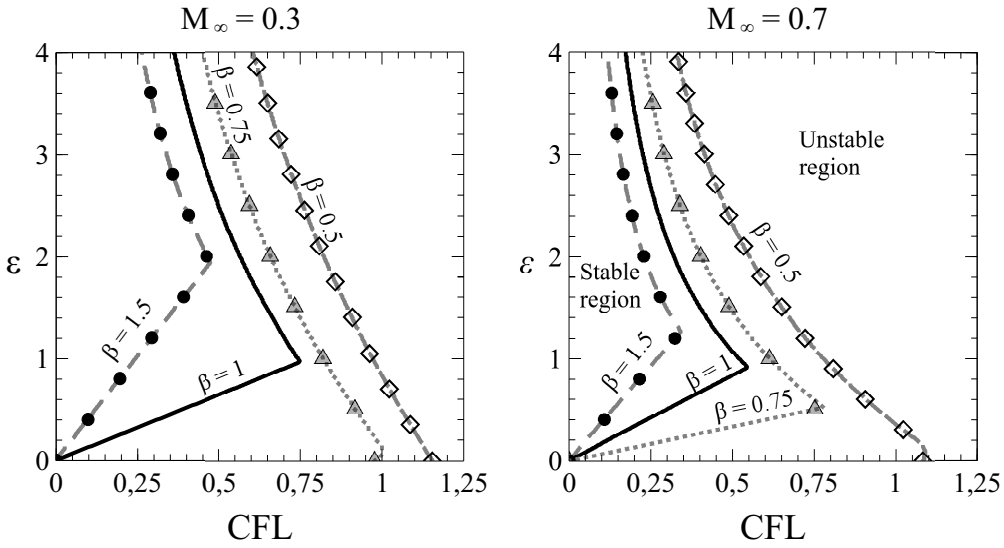


Figure 5.14: Relaxation method. Stability regions

where spec indicates the spectrum of the matrix.

In figure 5.14, different stability regions for different values of the relaxation factor are shown.

It is therefore possible to use a higher CFL number with $\beta < 1$ for the first time instants, when the initial strong oscillations must be damped and gradually increase the relaxation factor (and consequently decrease the CFL) as the simulation proceeds so as to speed up the convergence.

5.5 Boundary and initial conditions

In order to solve the problem (2.23) it is necessary to specify initial and boundary conditions.

Initial conditions In the case of aerodynamic external flows, the initial conditions on the velocity potential is simply

$$\phi\left(\boldsymbol{r},\, t=0\right) = \phi_0\left(\boldsymbol{r}\right) = \boldsymbol{V}_\infty \cdot \boldsymbol{r} \tag{5.55}$$

and the initial discretized velocity potential is obtained sampling equation (5.55) in the centroids of every cell.

This initial condition is not compatible with the non penetrability condition. However this is not a problem because the flow is compressible.

The same sampling procedure is carried out to obtain the initial discretized density from its initial free stream value

$$\rho\left(\boldsymbol{r},\, t=0\right) = \rho_0\left(\boldsymbol{r}\right) = \rho_\infty. \tag{5.56}$$

Boundary conditions The standard method to impose boundary conditions in finite volume approximation is to split the circuitation integral of equation (5.3) in the sum of two parts. The first one is the union of the control volume faces that lie in the internal part of the domain and the second one is the set of faces that lie on the considered boundary

$$\oint_{\partial V} \rho\, \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}} = \int_{\partial V_{\text{internal}}} \rho\, \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}} + \int_{\partial V_{\text{boundary}}} \rho\, \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}}. \tag{5.57}$$

To impose boundary conditions it is necessary to compute the flux given by the value of the solution on the boundary. In this work three kind of boundaries have been considered:

- far field boundary, where asymptotic flow conditions are imposed;
- wall boundary;
- wake, for lifting bodies only.

5.5.1 Far field boundary

Using the particular scheme adopted in this work, i.e. 2–fields potential flow with explicit staggered time stepping and Cell–Centered finite volumes, external boundary conditions must be imposed to the mass conservation equation only[2]. This is done by adding to the value of density inside every cell that has an interface lying on the external boundary the mass flux through that interface given by asymptotic conditions. In doing that, it must be taken into consideration that the mass conservation equation has a hyperbolic nature and the imposition of boundary conditions in this kind of equations is not trivial. In fact, it is necessary to distinguish between inflow and outflow boundaries. Where the flow is entering the domain, external boundary conditions can be used to compute the boundary flux, and where the flow is exiting the domain internal conditions must be used.

For example, with reference to figure 5.15, the flux through the interface 1–2 (that is an inflow interface) of cell A is computed as

[2]Using a 1–field formulation other types of boundary conditions must be imposed on the velocity potential

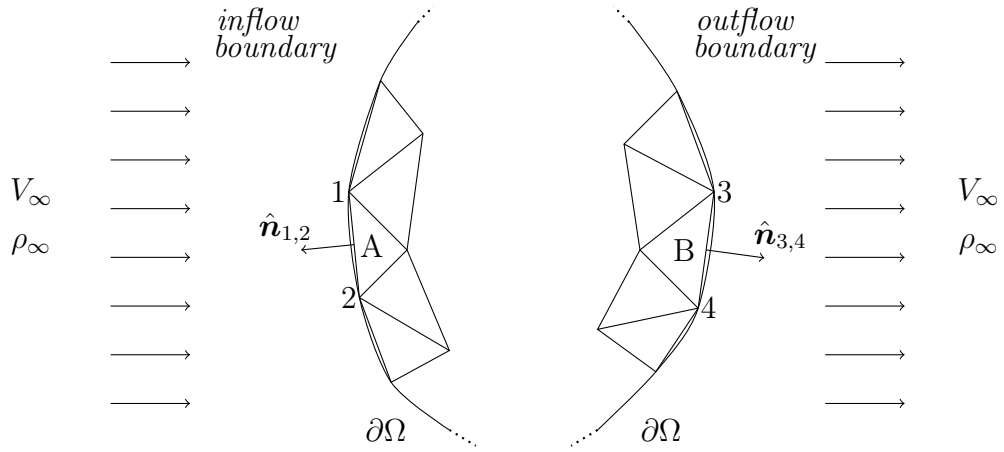


Figure 5.15: Far Field Boundary Conditions

$$\oint_{\partial V_{\text{inflow}}} \rho \, \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}} \approx \mathcal{F}_{1,2} = \rho_\infty \, \boldsymbol{V}_\infty \cdot \hat{\boldsymbol{n}}_{1-2} \; \Delta L_{1-2} \,, \tag{5.58}$$

while the flux through interface 3–4 (that is an outflow interface) of cell B is computed as

$$\mathcal{F}_{3,4} = \rho_B \, \boldsymbol{V}_\infty \cdot \hat{\boldsymbol{n}}_{3-4} \; \Delta L_{3-4} \,. \tag{5.59}$$

For what concerns the Bernoulli theorem, being it a local and not an integral relation, no boundary conditions must be imposed.

5.5.2 Wall boundary

Through walls there is no mass flux

$$\oint_{\partial V_{\text{wall}}} \rho \, \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}} = 0 \tag{5.60}$$

and therefore, in the case of fixed wall, there is no necessity to compute any flux.

Oscillating airfoil: transpiration conditions If it is necessary to find a solution for a body moving through the fluid and if the amplitude of the movement is small, it is possible to account for the effect of this motion by imposing the so called transpiration conditions. These conditions realize the same effect as if the mesh was actually moved but without the necessity to use an ALE scheme. To do that it is necessary to define a wall speed

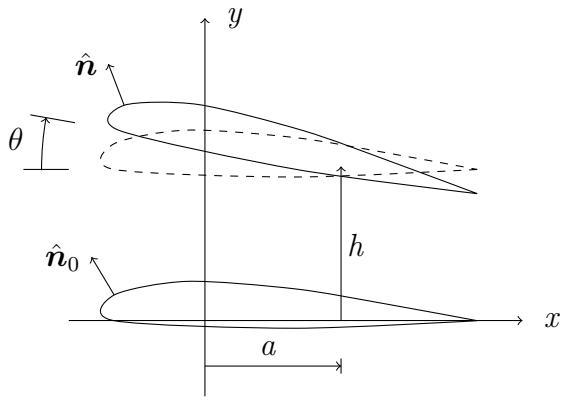


Figure 5.16: Transpiration Boundary Conditions

with which to impose a mass flux through the boundary. With reference to figure 5.16, this velocity is

$$V_{\text{n wall}}(\boldsymbol{r}) = \underbrace{-\boldsymbol{V}_\infty \cdot (\hat{\boldsymbol{n}}(\boldsymbol{r}) - \hat{\boldsymbol{n}}_0(\boldsymbol{r}))}_{\text{geometric effect}} + \underbrace{\left(\dot{h}\,\hat{\boldsymbol{y}} - \dot{\theta}\,\hat{\boldsymbol{z}} \times (\boldsymbol{r} - a\,\hat{\boldsymbol{x}})\right) \cdot \hat{\boldsymbol{n}}_0}_{\text{kinematic effect}}, \tag{5.61}$$

where $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{z}}$ are the reference system unit vectors.

To impose the transpiration boundary condition the following wall flux must be used

$$\mathcal{F}_{\text{transpiration}} = \rho\, V_{\text{n wall}}\, \Delta L, \tag{5.62}$$

where ρ is the value of density inside the boundary cell and ΔL is the length of the interface.

5.5.3 Wake

The most critical aspect of lifting potential flows is how to impose the correct value of circulation. As stated in section §2.1.1, it is necessary to impose that density is continuous through the wake. This is done by imposing the mass conservation over a control volumes obtained by merging all triangles that share an interface on the wake. Referring to 5.17, the resulting control volumes are the ones hatched with the same pattern, in which the value of density is uniform across the wake.

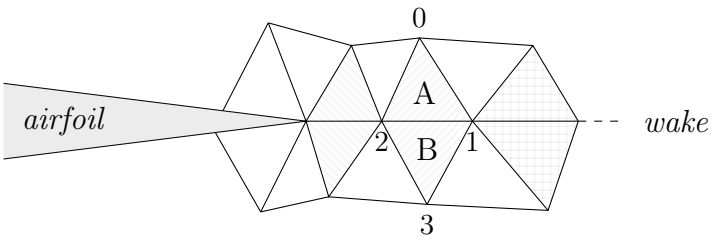


Figure 5.17: Wake Boundary Conditions. *Control Volumes*

It should be underlined that volumes above the wake are not physically merged to their counterparts below, because when the circulation is not null, the velocity potential is discontinuous across the wake. Therefore the algorithm to update the solution on the wake is

- update the value of density in cell A

$$\rho_A \quad += \quad -\frac{\Delta t}{V_A + V_B} \left(\mathcal{F}_{0-1} + \mathcal{F}_{0-2} + \mathcal{F}_{2-3} + \mathcal{F}_{1-3}\right), \tag{5.63}$$

  where the symbol $+=$ must be intended in the C syntax meaning, i.e. "add to ρ_A";

- copy the value of ρ_A in the cell B;

- update the value of the velocity potential via the standard formula (5.35).

Chapter 6

Solver and results

In the first part of this chapter the structure of the program ExPReS is presented.

The second part of the chapter presents some results obtained with ExPReS. Section §6.2 is essentially a validation of the program and presents some aerodynamic flows around the airfoil NACA0012 at different Mach numbers and angles of attack compared to the results given by the solver S^T (implicit 2–fields full potential solver, [5]) and by the solver rhoCentralFoam (Euler equation solver) distributed with the software openFoam [57]. Section §6.3 presents some results for flows near the critical point, where BZT peculiar phenomena appear in the flow field, such as expansion shocks and mixed waves. These results are compared with those available in literature or computed with the solver S^T.

In section §6.5 some results for unsteady flows around moving bodies are presented. These solutions are computed by the transpiration boundary conditions technique and by ALE simulations. Comparisons are made between these two approaches and with solutions provided by S^T in order to validate the results.

In section §6.7 the speed up results that where achieved using GPGPU computing instead of traditional CPU software are outlined and briefly commented. Finally a comparison between the speed–ups obtained with the solver ExPReS are compared to those declared by other commercial software developers.

6.1 The aerodynamic solver

The structure of the aerodynamic software is depicted in figure 6.1. The first argument of the executable must be the mesh in the proprietary format of ExPReS and the second optional argument is a binary file containing a previ-

ously computed solution written by ExPReS itself. All the numerical schemes implemented in the solver can be activated or deactivated by commenting or uncommenting some preprocessor directives contained in a configuration file and by recompiling the code. These directives allow to:

- choose different numerical schemes to compute the gradient (Least Square (LS) or Gauss theorem (see section §5.3.1);
- choose different upwinding techniques (all the ones presented in section §5.3.2);
- choose different time stepping schemes (constant time step, adaptive time step limited by a maximum allowed CFL number, local time stepping and relaxation method, see section §5.4);
- Activate transpiration boundary conditions (see section §5.5.2) or ALE formulation (see appendix A).

The first part of the program initializes the OpenCL environment through the use of the OpenCL API functions to identify the platform, get system devices (only GPUs are sought for), create the context and initialize the queue. OpenCL C sources that contain the code that will run on the graphic card is then read and compiled. The host code reads the mesh file and creates memory buffers where all the metrics will be stored by the first kernels that are executed. After setting numerical parameters and the asymptotic conditions, the unknown fields ρ and ϕ are initialized and copied to their corresponding device buffers. Optionally, these fields can be initialized from a previously stored solution that the solver writes in a binary format every time a simulation ends. The time loop then starts and the kernels that compute the velocity field $\boldsymbol{\nabla}\phi$ is enqueued. If necessary, depending on the upwinding scheme, the gradient of the density field is computed. Internal domain mass fluxes are then computed by a kernel that works on all the interfaces. External boundary fluxes are computed eventually in parallel (in the sense that these two kernels are executed out of order) with the mass conservation enforcement across the wake. All the density fluxes are then used to update the density field. Subsequently the velocity potential is updated by the Bernoulli equation where the enthalpy is computed with the updated value of density as prescribed by the staggered time integration method (section §5.4.2). If the current time step corresponds to a write time step, buffers are read back in the host memory and VTK files containing density ρ, velocity $\boldsymbol{\nabla}\phi$, Mach number M, pressure coefficient C_P and fundamental derivative Γ are written on disk. The coefficient of lift, drag and moment are computed and printed on file. If the time has reached the final time, the time loop ends

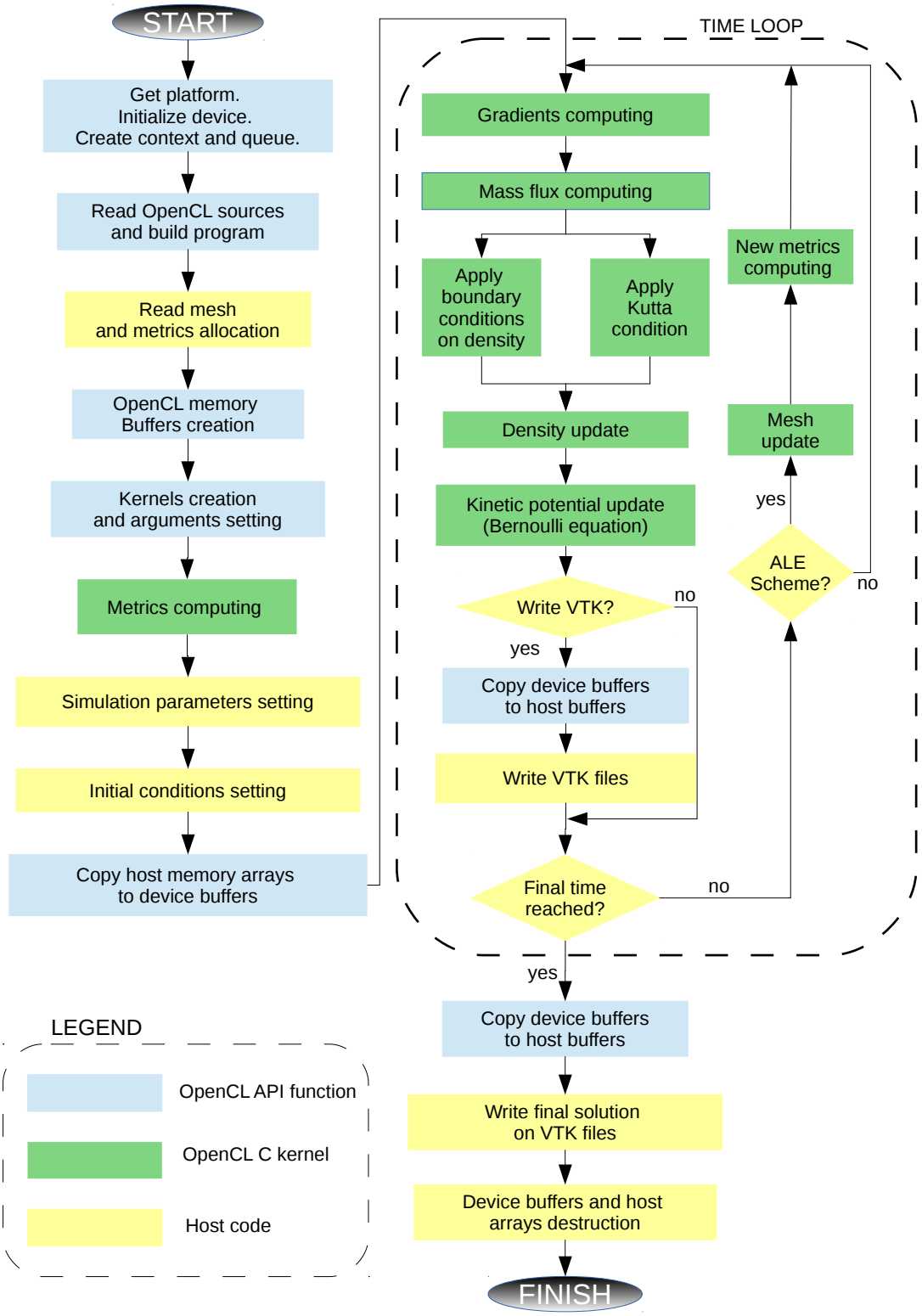


Figure 6.1: Aerodynamic solver scheme.

and the final solution is written to disk. The distribution of the pressure coefficient on the surface of the body is computed and written in a data file. The aforementioned binary file containing the final solution is written and all buffers and arrays are destructed. VTK files can be easily visualized with the free program ParaView [19] and the surface pressure coefficient with the free program gnuplot [58].

6.2 Ideal gas flows

The first test analyses are performed in a ideal gas regime to validate results provided by ExPReS with those computed with the solver S^T and with other results available in literature.

A polytropic ideal gas model is used with the constant $\gamma = 1.4$, that well represents common aerodynamic flows. Three solutions are computed around the NACA0012 airfoil in different asymptotic conditions and angles of attack. The mesh geometry is depicted in figure 6.2 beside an example mesh[1]. The used mesh has two regions with different spatial discretization. The internal zone is much more refined than the external one in order to improve the accuracy of the solution near the airfoil and to dissipate acoustic waves near the far field.

All simulations were tested for grid convergence.

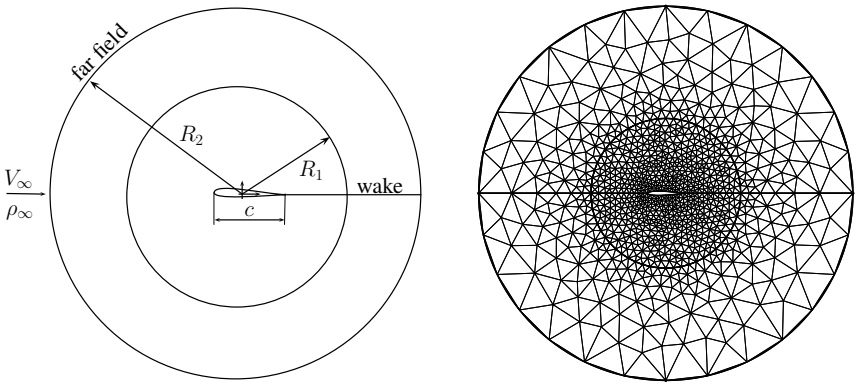


Figure 6.2: Naca 0012 mesh.

6.2.1 Subsonic airfoil

For this simulation the free stream Mach number is $M_\infty = 0.5$ and the asymptotic density value is $\rho_\infty = 1.225\,\mathrm{kg/m^3}$. With reference to figure 6.2

[1] This mesh is for presentation purposes only since the actually used meshes are much larger and finer.

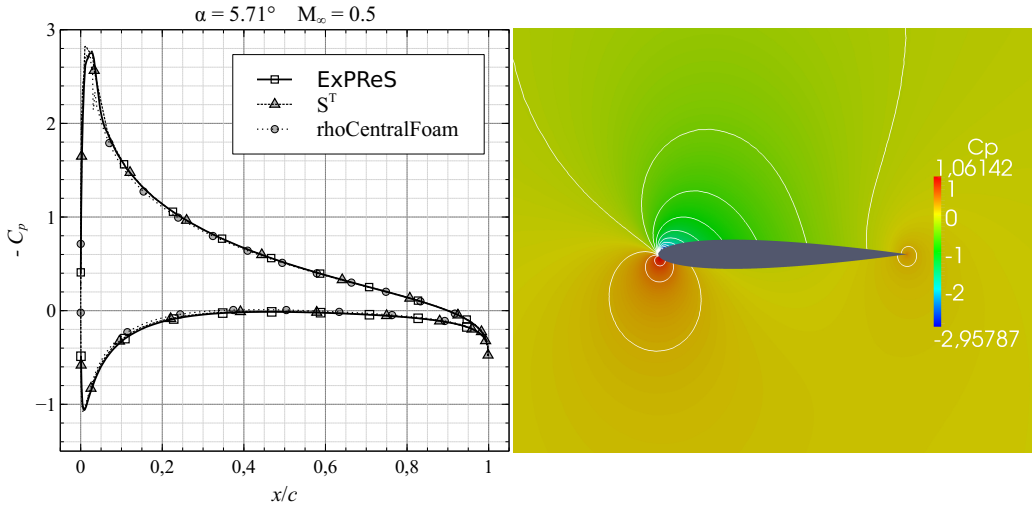


Figure 6.3: Subsonic NACA0012. Polytropic ideal gas. C_P surface distribution and contour plot.

this computation is carried out on a domain with $R_1 = 5\,c$ and $R_2 = 20\,c$ discretized with a mesh with 77 819 nodes and 154 765 triangles.

The solution for the angle of attack $\alpha = 5.71°$ is depicted in figure 6.3. The value of the pressure coefficient is shown in figure 6.3 where the solution computed with ExPReS is compared to the one provided by the solver S^T [5] and by the solver rhoCentralFoam [57]. There is a very good agreement between these three solutions. The small discrepancy near the leading edge between the solutions computed with ExPReS and with S^T could be due to a different mesh refinement.

It is underlined that there are no practical differences between the full potential solutions and the Euler solution. In fact, in the subsonic case, full potential solvers allow to compute the aerodynamic field very precisely with a much smaller complexity and computational effort than with Euler solvers.

6.2.2 Transonic airfoil

One of the aims of this work is to develop a solver capable to compute transonic solutions.

The first test case in this regime is a non lifting flow around a NACA0012 airfoil at Mach number $M_\infty = 0.85$. Since the airfoil is symmetric, to obtain no lift the angle of attack is $\alpha = 0°$. The asymptotic density is $\rho_\infty = 1.225\,\mathrm{kg/m^3}$ and the mesh is the same of the case presented in section §6.2.1. In this regime two identical shock waves form on the upper and lower part of the airfoil. The contour plot of the pressure coefficient and its distribution on

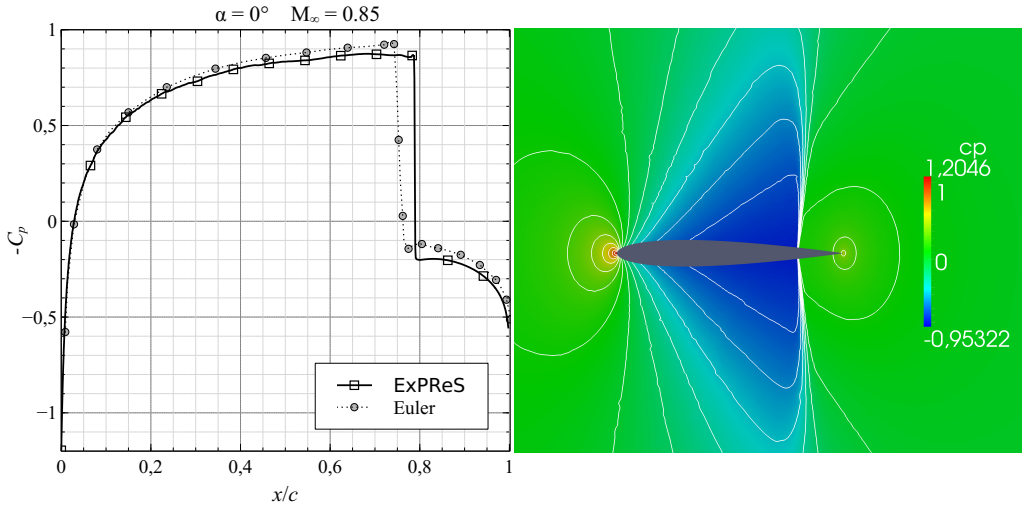


Figure 6.4: Transonic non lifting NACA0012. Polytropic ideal gas. C_P surface distribution and contour plot.

the airfoil surface are depicted in figure 6.4. The surface pressure coefficient distribution is compared to that computed by an Euler equation solver [59]. In this case the two solutions are in good agreement and the shock position is well captured by the full potential solution.

The second test case in the transonic regime is a lifting flow around a NACA0012 airfoil at Mach number $M_\infty = 0.7$ and angle of attack $\alpha = 4.09°$. The asymptotic density is $\rho_\infty = 1.225\,\mathrm{kg/m}^3$ and the mesh is the same of section §6.2.1. In this regime a strong compressive shock wave forms on the upper surface of the airfoil. The solution is compared with the ones provided by S^T and by rhoCentralFoam. In the case of lifting flows, solutions provided by full potential solvers present generally an error in the shock position. This error is due to the isentropic assumption that makes jump conditions not satisfied, making the shock move towards the trailing edge. In the Euler case the shock is at the 40% of the chord, while it is placed at $0.55\,c$ in the full potential solutions. This aspect is depicted in figure 6.5, where it is anyway possible to note that the two full potential solutions provided by ExPReS and S^T agree on the shock position. The value of lift coefficient and aerodynamic moment coefficient (computed with respect to a point at $x/c = 0.25$) are

$$C_L^{ExPReS} = 0.975; \quad C_M^{ExPReS} = 0.0287;$$
$$C_L^{Euler} = 0.731; \quad C_M^{Euler} = -0.004$$

with a bigger relative error on the moment coefficient due to the fact that in the Euler case the low pressure area is more advanced than in the full

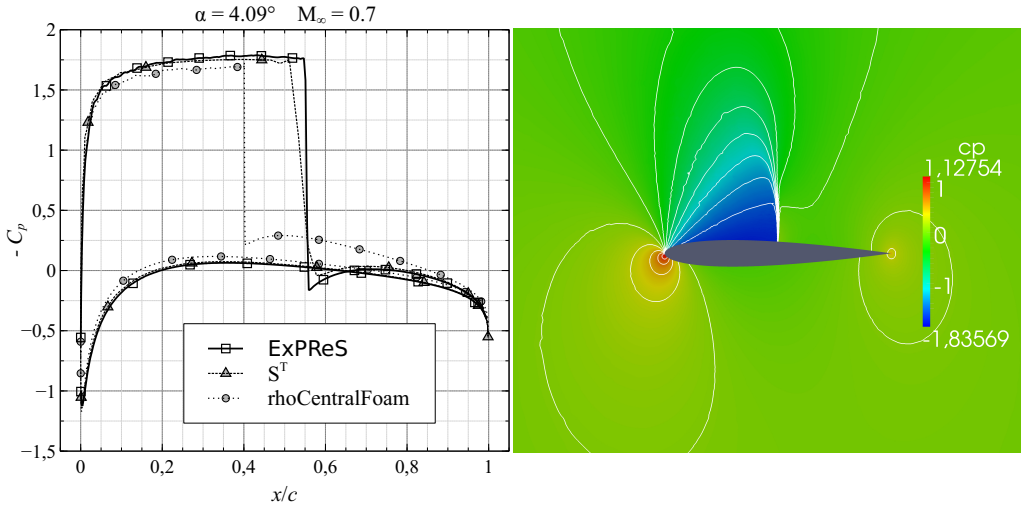


Figure 6.5: Transonic lifting NACA0012. Polytropic ideal gas. C_P surface distribution and contour plot.

potential case.

It is finally reported that the shock position can be better captured by non isentropic full potential solutions as in the original work by Parrinello [5].

6.3 Dense gas flows

Dense gas flows are computed in order to analyze some of the phenomena described in section §3.4. Results are compared to some cases available in literature computed with an Euler solver [59] and with solutions provided by S^T.

The gas model is a polytropic van der Waals gas with $\gamma = 1.0125$, therefore capable to express BZT fluids behaviors (see section §3.4). All the thermodynamic variables have been normalized by their corresponding critical values. For this normalized gas, the van der Waals constant a and the covolume constant b are respectively

$$a = 3 \tag{6.1}$$

and

$$b = 1/3. \tag{6.2}$$

The following solutions are computed around the widely used NACA0012 airfoil on various meshes with the geometry described in figure 6.2.

6.3.1 Subsonic airfoil in dense gas flow

In the dense gas region, it could happen that in an isentropic expansion the speed of sound increases, leading to a reduction in the Mach number. This is exactly what happens in this case. For a PIG flow around the NACA0012 airfoil the critical Mach number[2] is approximately 0.8 and for angles of attack different from zero, shock waves appear even at lower Mach numbers. In this section three different solutions are computed with a free stream Mach number $M_\infty = 0.85$ and at different angles of attack equal to $\alpha = 1°$, $\alpha = 3°$ and $\alpha = 5°$. In all of them however no shocks are present due to the fact that the fundamental derivative of gas dynamics Γ becomes negative in the expansion region after the leading edge as depicted in figures 6.6, 6.7 and 6.8. Thus, as explained in section §3.4, the speed of sound increases even if the flow is expanding, keeping the stream subsonic.

The asymptotic conditions for this cases are

$$\begin{aligned} M_\infty &= 0.85, \\ \frac{P_\infty}{P_C} &= 1.07, \\ \frac{\rho_\infty}{\rho_C} &= 0.920, \end{aligned}$$

where the subscript C denotes the critical conditions. This asymptotic state will be denoted through the text with the initials DG1 (dense gas 1).

The grid on which these computations are made is composed by 154 765 triangles and 77 819 nodes. The obtained solutions are compared to those obtained by Cinnella [59].

It is possible to observe that in all these three cases there is a very good matching between solutions computed with the Euler model and those provided by the the two full potential solvers.

The last of the three cases presented in this section is a limit case for an inviscid solver. In fact, even if in a normal gas regime no separations are expected for a NACA0012 airfoil at an angle of attack of 5°, in the case presented in figure 6.8 there is a very strong adverse pressure gradient that could lead to the separation of the boundary layer.

[2]Lower free stream Mach number at which there is a sonic point on the the airfoil at a null angle of attack.

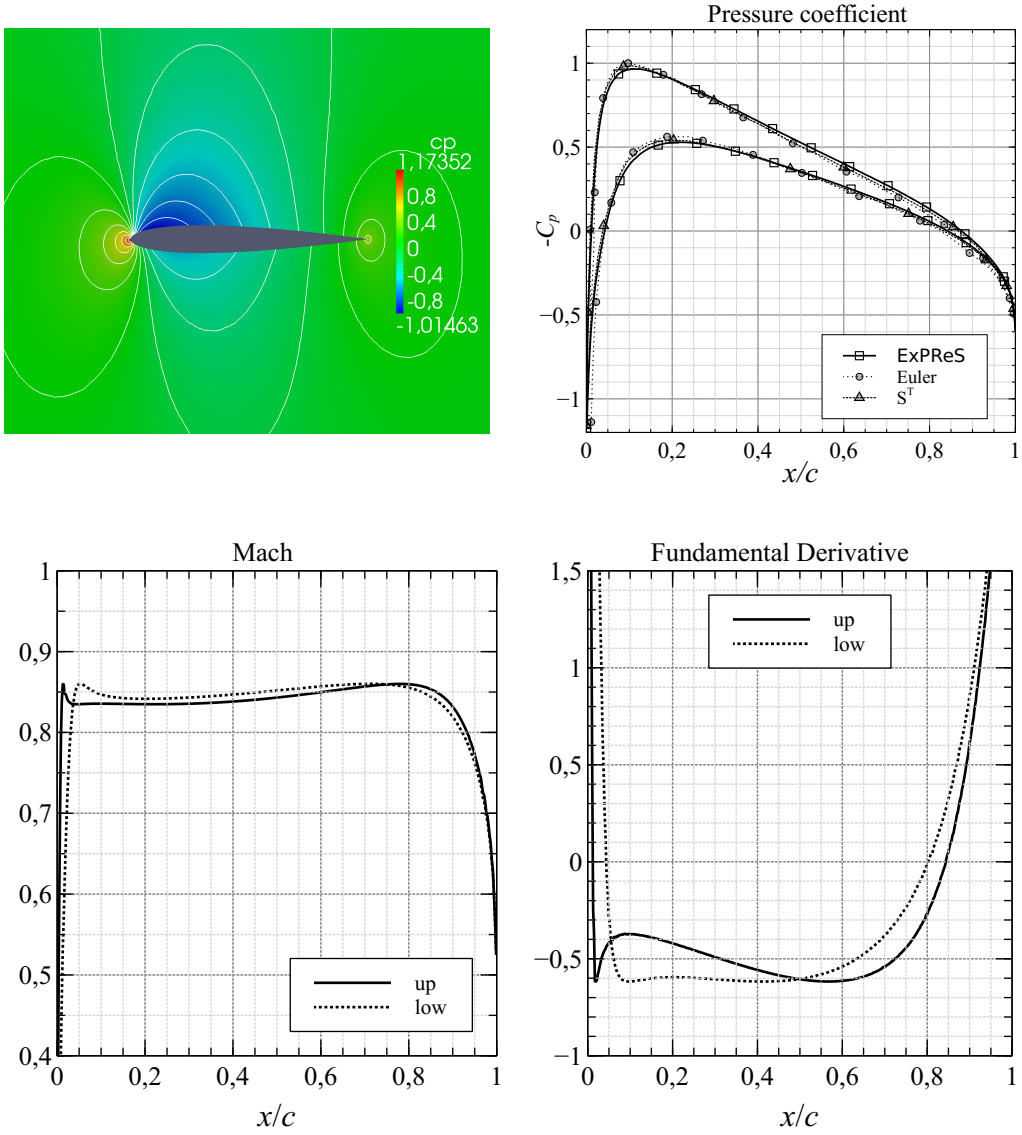


Figure 6.6: DG1. Subsonic dense gas, $\alpha = 1$.

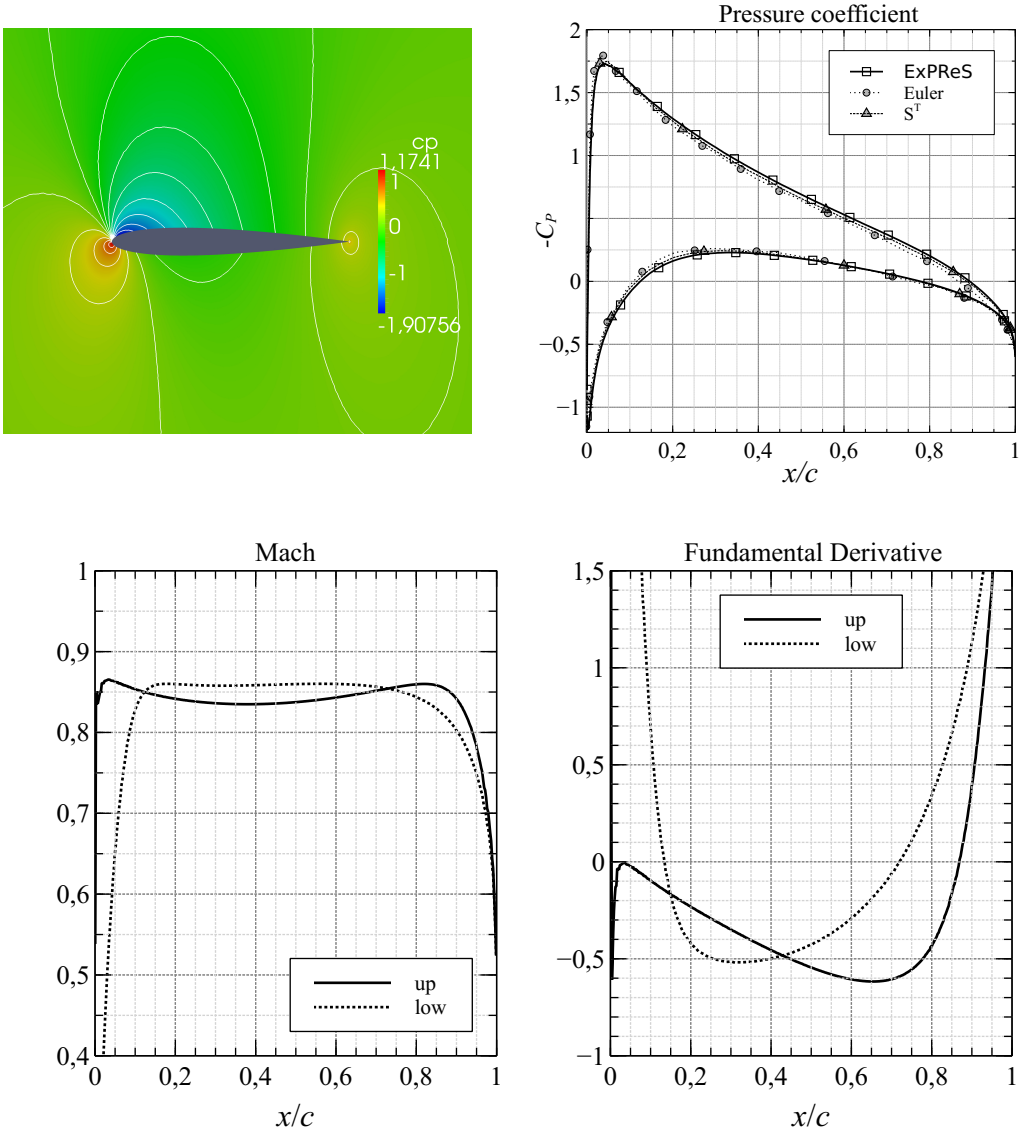


Figure 6.7: DG1. Subsonic dense gas, $\alpha = 3$.

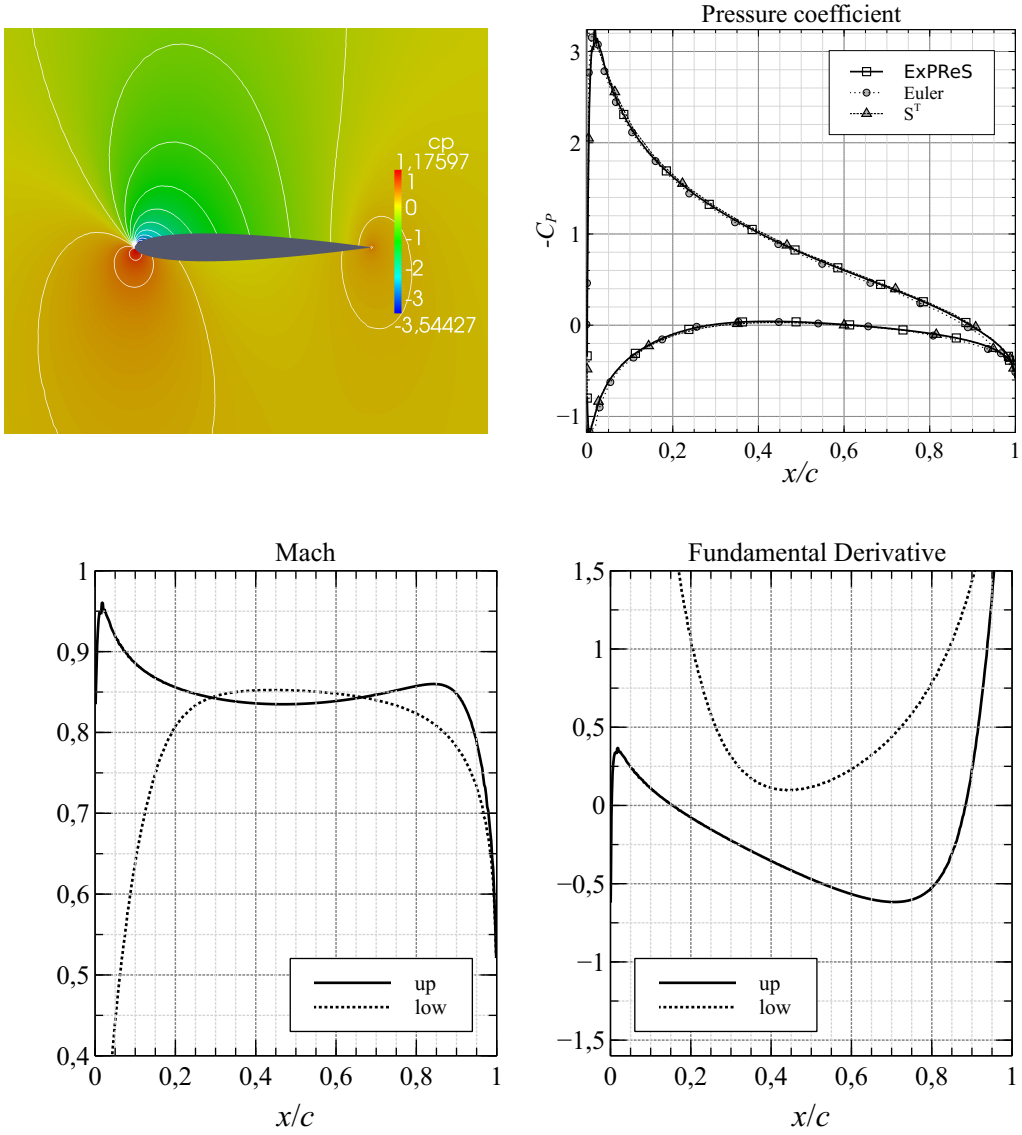


Figure 6.8: DG1. Subsonic dense gas, $\alpha = 5$.

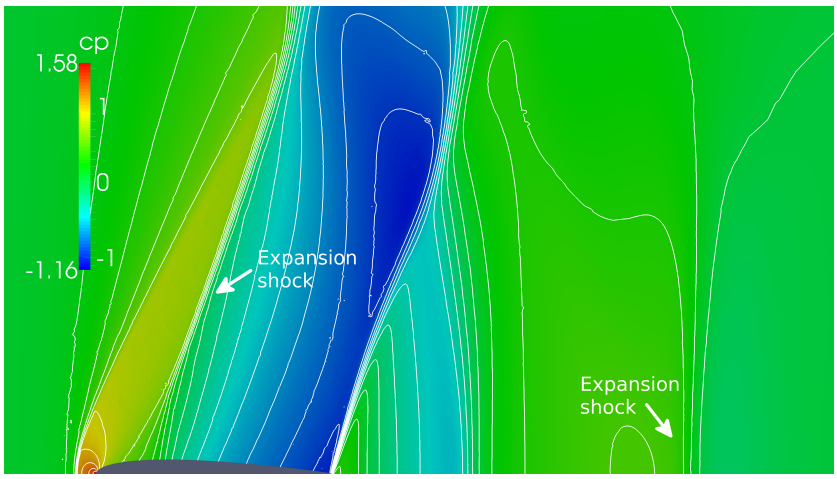


Figure 6.9: DG2. Pressure coefficient distribution and contour plot

6.3.2 BZT gas phenomena

A transonic flow in which many BZT gas peculiar phenomena are present can be obtained from the asymptotic conditions

$$\begin{aligned} M_\infty &= 0.998, \\ \frac{P_\infty}{P_C} &= 0.944, \\ \frac{\rho_\infty}{\rho_C} &= 0.600, \end{aligned}$$

that give a negative free stream value of the fundamental derivative of gas dynamics equal to $\Gamma_\infty = -0.0447$. This case will be denoted through the text with the initials DG2 (dense gas 2).

The solution for this flow is computed on a mesh with almost 764 052 triangles and 382 379 nodes. Such an high spacial resolution was used to obtain a high definition of shocks and fans. The solution interval time is $t \in [0,\ 93\,\mathrm{s}]$. On the airfoil surface, steady state convergence is rapidly obtained (in approximately 20 s), but downstream of the trailing edge the convergence is much slower, since the expansion shock that forms in that regions keeps moving backward slowly.

In figure 6.9 the pressure coefficient distribution and its contours are depicted. It is possible to observe a compressive bow shock in front of the airfoil even though the asymptotic free stream is subsonic. In fact, with reference to figure 6.12, it is possible to see that the flow becomes supersonic before reaching the leading edge due to a drop of the speed of sound. Behind

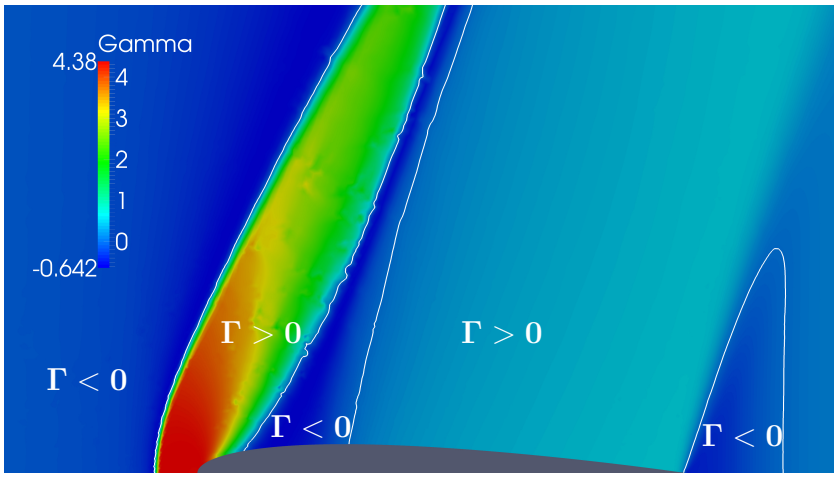


Figure 6.10: DG2. Fundamental derivative of gas dynamics. Contour plot for $\Gamma = 0$.

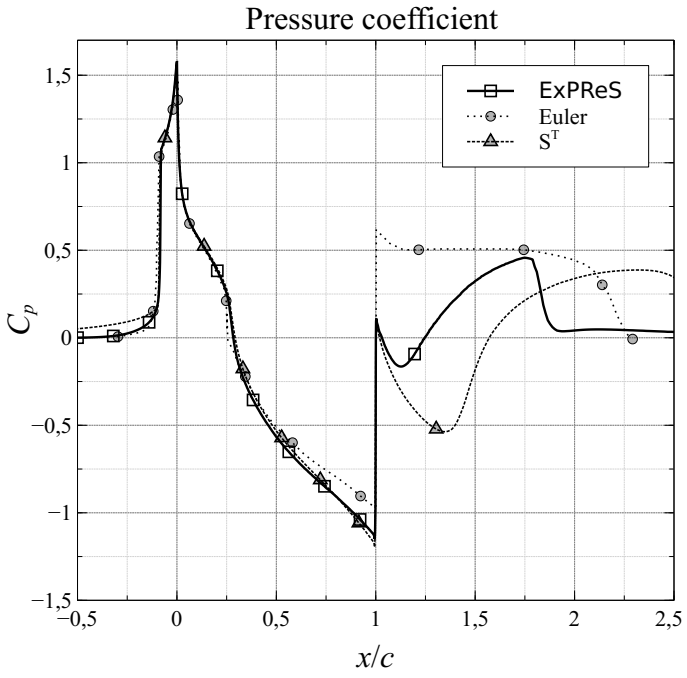


Figure 6.11: DG2. Pressure coefficient along the wall streamline. The airfoil begins in $x/c = 0$ and ends in $x/c = 1$.

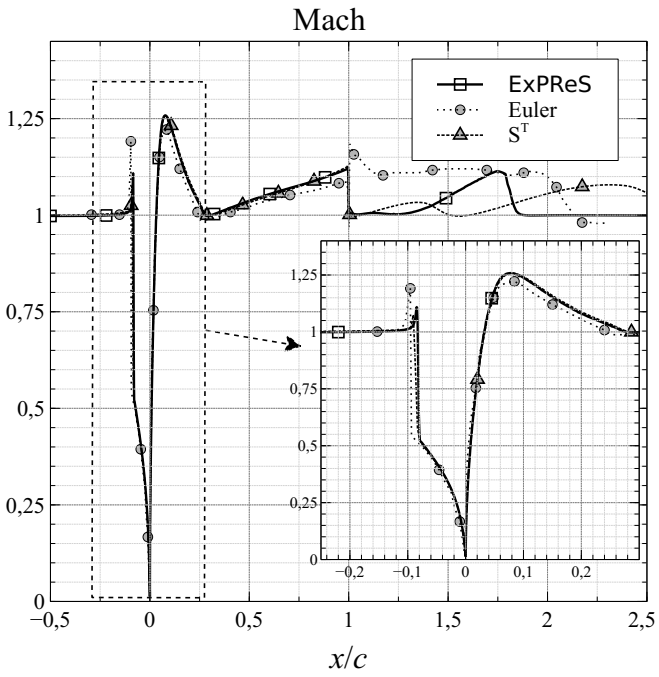


Figure 6.12: DG2. Mach distribution along the wall streamline. The airfoil begins in $x/c = 0$ and ends in $x/c = 1$.

this bow shock the flow becomes subsonic and its Mach number approaches zero at the stagnation point. The flow then undergoes a rapid expansion near the 25% of the chord. From figure 6.9 it is possible to notice that this rapid expansion region merges in an expansion shock above the airfoil. At the trailing edge a strong compression shock forms where the entropy jump is not negligible. In that region, the Euler equations require the formation of a mixed compressive wave composed by a shock followed by a fan, as discussed in [59]. The full potential solution is not capable to capture this fan probably due to the isentropic assumption that across such a strong shock is no more satisfied. Downstream to the trailing edge, the full potential solution is no more coherent with the Euler one, even if an expansion shock is captured by both of them.

In figure 6.13 the distribution of the fundamental derivative of gas dynamics is represented along the wall streamline. It is possible to note that before the bow shock its value decreases to approximatively -0.5. This is precisely the cause of the increase in the Mach number (depicted in figure 6.12) in this region that leads to the formation of the bow shock.

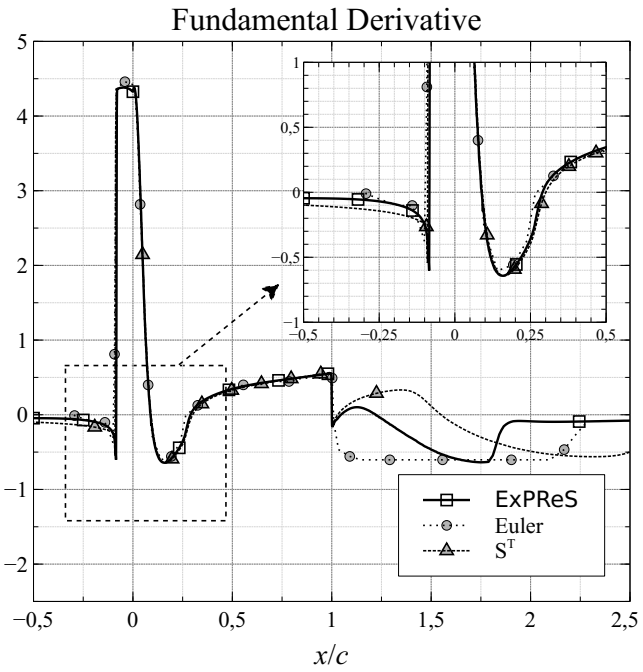


Figure 6.13: DG2. Fundamental derivative along the wall streamline. The airfoil begins in $x/c = 0$ and ends in $x/c = 1$.

6.3.3 Transonic lifting airfoil in dense gas flow

In this section a lifting transonic case in dense gas regime is presented. Results are compared to those published in [59]. The asymptotic conditions are

$$\begin{aligned} M_\infty &= 0.85, \\ \frac{P_\infty}{P_C} &= 1.116, \\ \frac{\rho_\infty}{\rho_C} &= 1.05. \end{aligned}$$

The solution in depicted in figure 6.14. On the upper surface, near the leading edge it is possible to observe an expansion shock. In that region the solution of the Euler equations is in good agreement with the one provided by ExPReS. On the lower side, the flow goes through a rapid expansion region where it is possible to observe a not negligible discrepancy between the two solutions.

The two compressive shocks near the trailing edge are captured by the full potential solution but the pressure recovery across them is smaller than

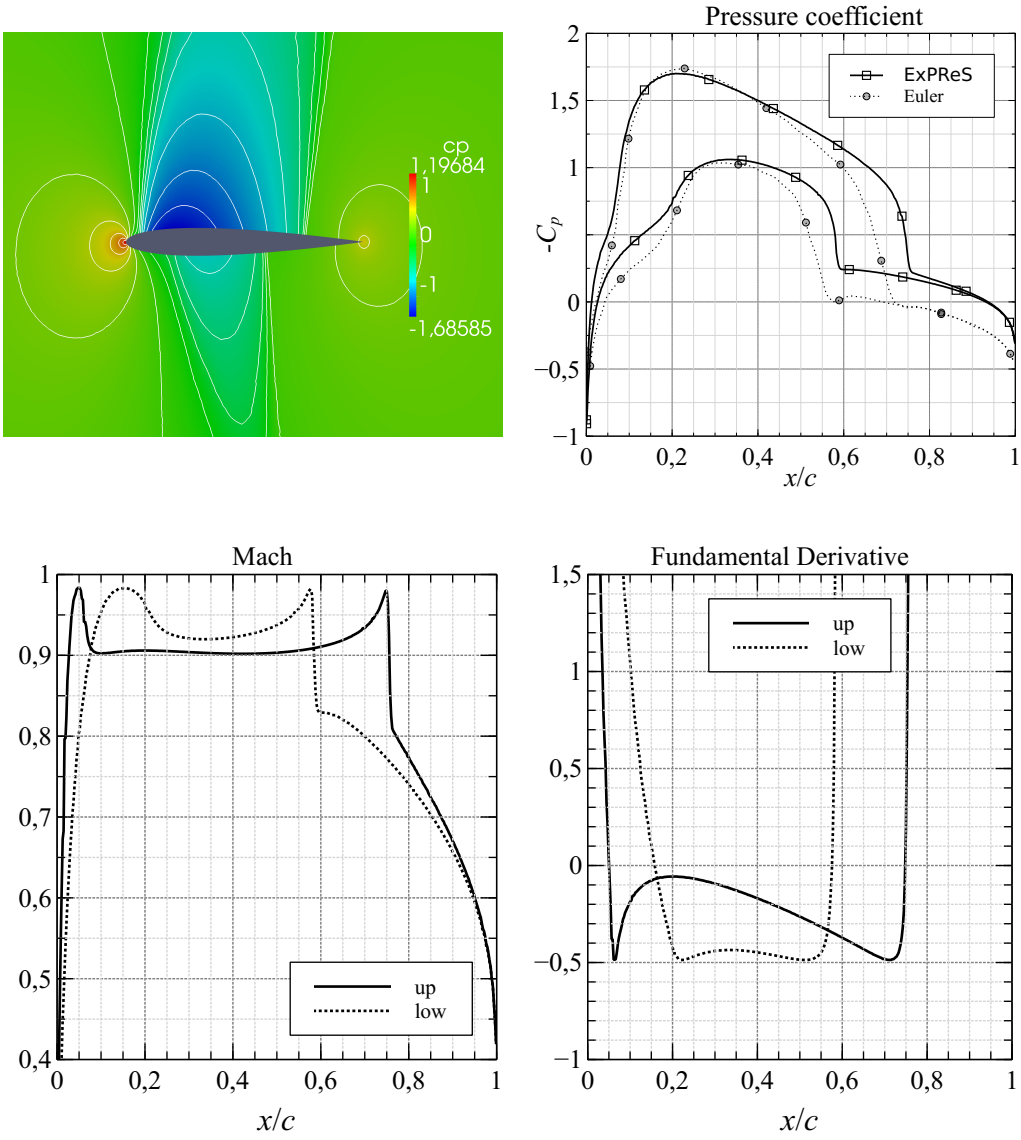


Figure 6.14: Transonic lifting NACA0012. Polytropic van der Waals gas. C_P surface distribution and contour plot.

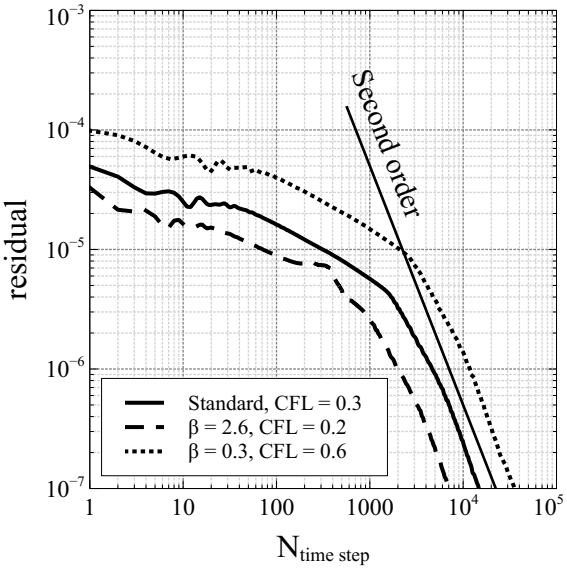


Figure 6.15: Residual convergence

the one provided by the Euler solution. A possible explanation of this error could reside in the isentropic assumption (2.1) that across a shock is never satisfied. The expansion shock on the upper surface of the airfoil is so well captured by the full potential solution since, as depicted in figure 6.14, the value of Γ in that region is very close to zero, hence leading to a small jump of entropy as explained in section §3.4.

It is interesting to note that in this regime shocks form even if the flow field is everywhere subsonic as depicted in the Mach distribution of figure 6.14.

6.4 Convergence acceleration techniques

The convergence acceleration techniques presented in section §5.4.3 are all implemented in ExPReS. In figure 6.15 the residual, computed as

$$Res = \frac{\sqrt{\sum_i \left(\frac{\Delta t}{V_i} \sum_{j=1}^{N_f} \bar{\rho}_{i,j}^{\,n} \, \boldsymbol{G}_{\Phi^n}^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j} \, \Delta L_{i,j} \right)^2 V_i}}{\sqrt{\sum_i R_i^2 \, V_i}} \tag{6.3}$$

is depicted. The index i and j are respectively the cell and the interface index. The numerator is a numerical approximation of the L_2 norm of the

mass conservation equation forcing term. The denominator is a numerical approximation of the L_2 norm of the density field.

The local time stepping did not proved to accelerate the convergence to the steady state, causing many spurious oscillations in the flow field. The over–relaxation procedure ($\beta > 1$) lead to a faster convergence even if the CFL number that guarantees the stability is smaller than that of the constant time stepping scheme. On the contrary, the under–relaxation procedure allows a higher CFL, but it slowed down the convergence, as predicted in section §5.4.3.

6.5 Unsteady simulations

To investigate unsteady solutions around moving bodies, the ALE formulation and transpiration boundary conditions were implemented in ExPReS. An essential parameter that characterizes the behavior of unsteady flows is the reduced frequency, defined as

$$k = \frac{\omega\, l_a}{V_\infty} \tag{6.4}$$

where ω is the structural motion frequency, l_a is an arbitrary aerodynamic length and V_∞ is the free stream speed. This dimensionless parameter measures the unsteadiness of the aerodynamic flow field subjected to the forcing of boundary conditions.

6.5.1 Subsonic oscillating airfoil

The first case presented is a comparison between the solutions computed with both transpiration boundary conditions and ALE simulation. To validate these results they are also compared to the solution obtained with the solver S^T. The polytropic ideal gas model is adopted and the asymptotic conditions are

$$\begin{aligned} M_\infty &= 0.5, \\ P_\infty &= 101325, \\ \rho_\infty &= 1.225\,. \end{aligned}$$

In this simulation only the pitching motion about the 25% of the chord was activated. The reduced frequency is $k = 1$ (computed using as aerodynamic reference length l_a the airfoil chord c) and the oscillation is

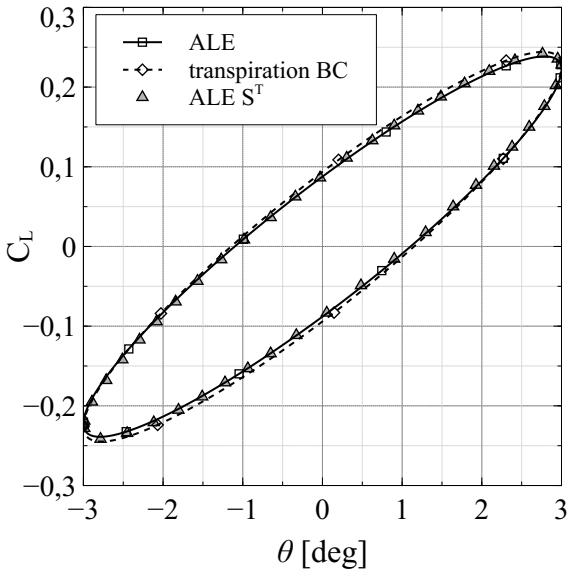


Figure 6.16: Lift coefficient vs. pitch angle. $M_\infty = 0.5$

$$\theta(t) = 0^\circ + 3^\circ \sin(\omega\, t)\,.$$

The mesh is rigidly moved, ensuring that the geometric conservation law (A.33) is satisfied.

The lift coefficient plotted against the pitch angle θ is depicted in figure 6.16. It is possible to observe a very good correspondence between the two implemented strategies and with data provided by S^T, even if this last solution was computed deforming the mesh.

6.5.2 Transonic oscillating airfoil

The second investigated case is an oscillating airfoil with shock waves moving on its surface. The asymptotic conditions are

$$\begin{aligned} M_\infty &= 0.755, \\ P_\infty &= 101325, \\ \rho_\infty &= 1.225\,. \end{aligned}$$

The pitching motion around the 25% of the chord is

$$\theta(t) = 0.02^\circ + 2.51^\circ \sin(\omega\, t)$$

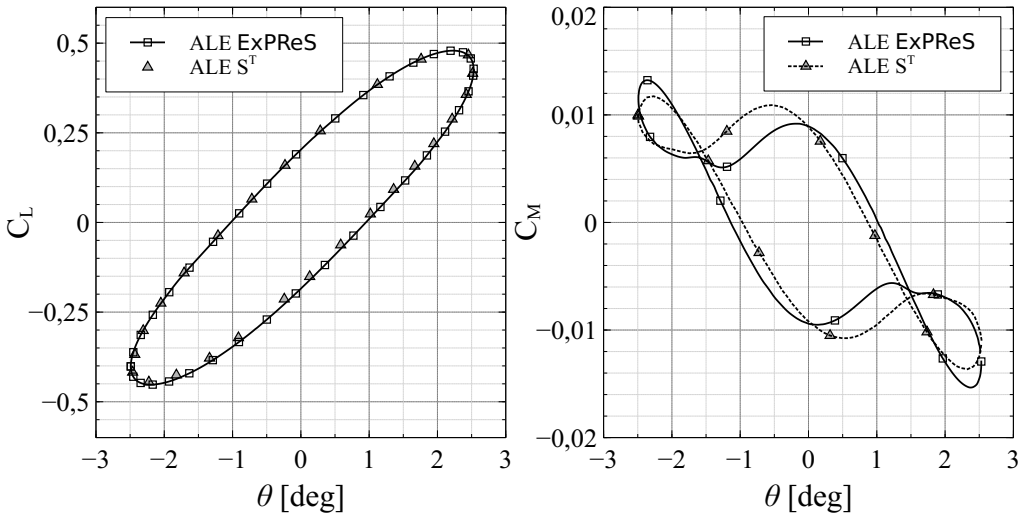


Figure 6.17: Oscillating transonic airfoil in ideal gas flow

and the reduced frequency is $k = 0.1$ (computed using as aerodynamic reference length l_a the airfoil chord c).

The mesh is rigidly moved. In figure 6.17 the lift and pitching moment coefficient are plotted against the pitch angle θ. Results are compared with those computed with the program S^T. It possible to observe a good correspondence in the lift coefficient loop, while there is a certain discrepancy in the moment coefficient. A possible explanation could be found in the fact that this last result is much more sensible to grid refinement and dissipation than the lit coefficient. The two solutions are computed on different grids and the solution provided by S^T is obtained deforming the mesh instead of moving it rigidly.

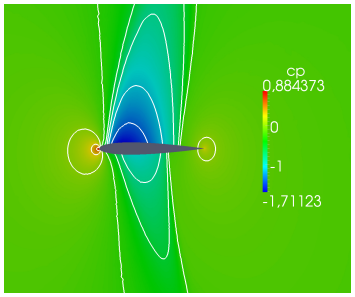
6.5.3 Transonic oscillating airfoil in dense gas flow

The unsteady behavior of an expansion shock is investigated by means of an ALE simulation in the dense gas transonic case of section §6.3.3. The airfoil pitch movement around the 25% of the chord has the sinusoidal shape

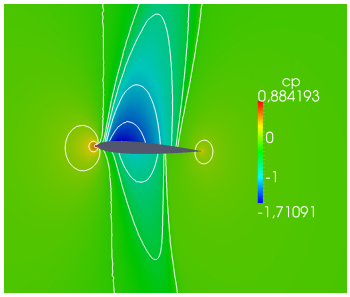
$$\theta(t) = 3^\circ \sin(\omega t)$$

with a reduced frequency $k = 0.5$ (computed using as aerodynamic reference length l_a the airfoil chord c). The two compressive shocks near the trailing edge assume an almost fixed position while the two expansion shocks near the trailing edge continue to form and disappear.

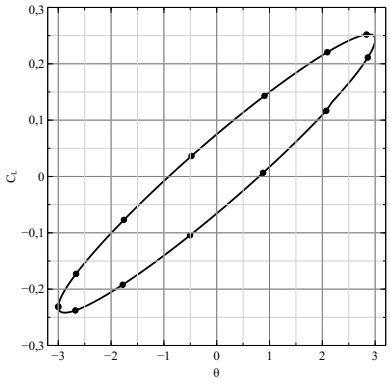
The solution is depicted in figure 6.18, where it is possible to observe the



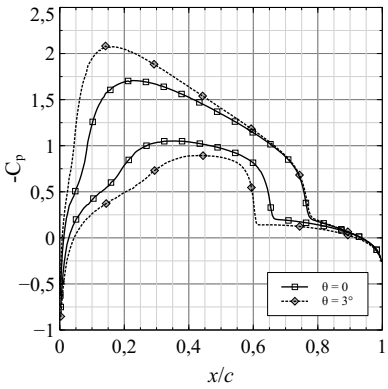
(a) $\theta = 0°$, downward pitching



(b) $\theta = 3°$



(c) Lift coefficient v.s. pitch angle



(d) Pressure coefficient distribution

Figure 6.18: Oscillating airfoil in dense gas flow

hysteresis on the lift coefficient. In figure 6.18a in fact the solution for $\theta = 0°$ is not symmetric.

6.6 3D flows

The solver ExPReS has been extended to compute 3D flows. Big problems were faced with gradient reconstruction schemes and they have not been completely resolved yet. In figure 6.19 a 3D subsonic stationary solution in ideal gas regime is depicted. It is a lifting flow around the ONERA M6 wing in the free stream conditions $M_\infty = 0.5$ and $\alpha = 2$. The wing span is $b = 1.196$ units and the mesh discretizes a semi-spherical domain with a radius of 20 units. It is composed by 334 963 tetrahedra and 69 510 nodes.

The pressure coefficient is plotted at three span wise stations, 30% 60% and 90% of wingspan.

The difficulties faced in the computing of 3D flows could be due to a programming error or to numerical stability reasons. Some other cell–center

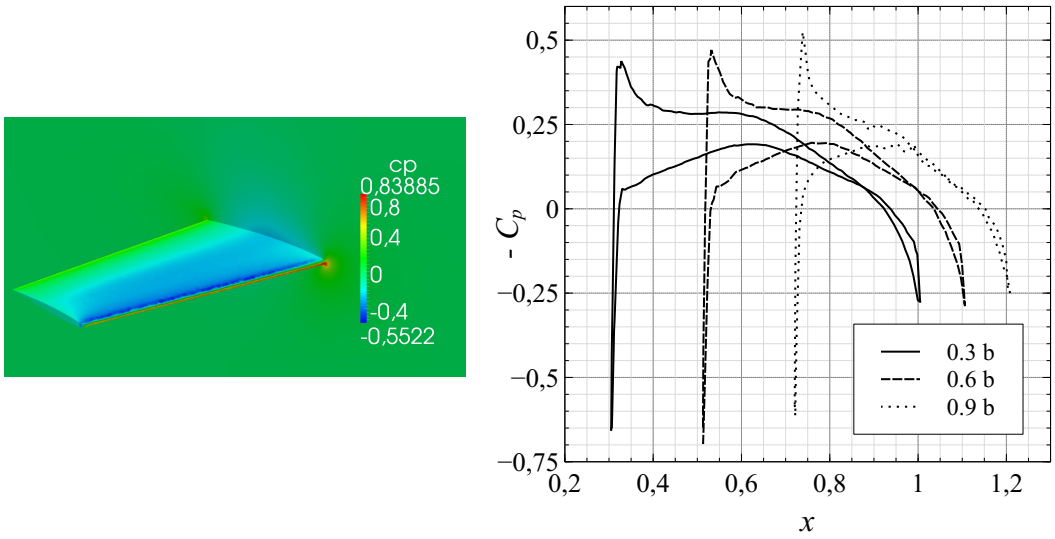


Figure 6.19: Onera M6 wing in subsonic regime

3D gradient reconstruction algorithms and stabilization techniques should be analyzed in order to further investigate the causes of these problems.

6.7 Speed up results

6.7.1 Hardware setup

All the benchmark tests that follow were performed with the hardware available to the authors of this work. Different CPU and GPU models were used to test the scalability capabilities of ExPReS. This hardware includes desktop quad–core CPUs, a notebook dual–core CPU with HyperThreading [3], a desktop GPU and a notebook GPU.

In computer world, hardware architectures and performances depend significantly from the time period they were developed and from their price category. Since one of the goals of this work is to prove the benefits of GPGPU in fluid dynamics, a comparison between a low–end CPU and a high–end GPU would not make sense. Thus, benchmarks were conducted on GPUs and CPUs of approximately the same time period and price cate-

[3]HyperThreading (HT) is a technology developed by Intel to optimize the use of CPU resources. If a CPU supports HT, the operating system will see one more logical core for each physical core in the CPU. This technology can lead to an increase of performance in some cases but it can't double the speed as with another physical core because only few hardware elements are really duplicated.

Model	Intel i5 760	Intel i5 2410m	AMD Phenom II X4 840
Architecture	Lynnfield	Sandy Bridge	Propus
Frequency	2.8 GHz	2.3 GHz	3.2 GHz
Cores	4	2	4
Threads	4	4 (HT)	4
Cache L2	4 x 256 KB	2 x 256 KB	4 x 512 KB
Cache L3	8 MB	3 MB	/
TDP	95 W	35 W	95 W
Performance	51 GFLOPS	46 GFLOPS	N. A.
Release date	JUL 2010	FEB 2011	JAN 2011
Price	205 $	N.A.	110 $

Table 6.1: CPU list

gory. Mid–range desktop hardware launched during 2010–2011 in the price range from 100 € to 200 € was available. For the notebook, the hardware is the typical one that can be found in a mid–range notebook ($\sim$ 700 €) sold during 2011. Table 6.1 and 6.2 report the hardware components available during the tests. All the CPUs tested were capable to take advantage of the autovectorization technology provided by the GCC compiler thanks to the support of SIMD set instructions like SSE and SSE2.

One of the most important advantages of the GPU architecture is its energy efficiency. For example the Thermal Design Power (TDP) of the i5 760 is 95 W while the TDP of the GTX 460 is almost double, 160 W, but in single precision the GPU has theoretically almost 20× the performance of the CPU. This means that if a simulation is well suited to run on a GPU, there are also important energetic, and consequently economical, savings. This is a critical aspect, for example, for clusters and supercomputers, where an important problem is the control of the temperature of the environment where the hardware is located. A higher GFLOPS/WATTS ratio means less heat and energy for the same computational power.

6.7.2 Solver setup

Many meshes with different number of elements were used in the following tests. This aspect is important because there is always some overhead when computations are carried out in parallel. For overhead it is intended any time, software and hardware resources needed for the execution of the application that are not strictly related to the computations performed to find the solution. For example in a shared memory system, when using a multithreaded application such as the multithreaded version of ExPReS, be-

Model	NVIDIA GT 540M	NVIDIA GTX 460
Architecture	Fermi	Fermi
Compute Units (SM)	2	7
Cores	96 (2x48)	336 (7x48)
Global Memory	1 GB DDR3	1 GB GDDR5
Local Memory	16–48 KB	16–48 KB
Cache L2	768 KB	768 KB
OpenCL version	1.1	1.1
CUDA capability	2.1	2.1
TDP	35 W	160 W
Performance (single precision)	258 GFLOPS	907 GFLOPS
Release date	JAN 2011	JUL 2010
Price	N. A.	229 $

Table 6.2: GPU list

fore any parallel computation, some time is needed for the creation of the required threads. Some time and hardware resources are also needed during computations to ensure cache coherence of the shared variables. Overhead arises also when computations are carried out by a GPU. As explained in 4.3.1, during the execution of a program on a GPU some time and resources are needed to allow communications between the host and the device. All of these aspect lead to the fact that if the problem to be solved is too small (i.e. the mesh is too coarse or few time–steps are computed), the overhead can be so deleterious that the serial version of the code can be faster than the parallel one. Thus, there exists a trade–off point before which the serial version of the code performs faster than its parallel version. What is expected is an increase of the benefits of the parallel version of ExPReS when the problem size increases.

An important aspect related to the solver setup and the GPU speedup respect to the CPU is the floating point precision chosen in the simulation. ExPReS is capable to perform simulations using both single or double precision floating–point numbers. However in order to guarantee a good accuracy of the results and to prevent numerical cancellation, double precision is mandatory. Usually on GPUs data sheets single precision floating point performance are shown in GFLOPS. A modern mid–range GPU is capable of 1–2 TFLOPS in single precision. High–end GPUs reach over 4 TFLOPS in single precision. However in double precision computations, performance are lower, even by an order of magnitude. This is clear in figure 4.14. Another aspect to keep in mind is that theoretical GFLOPS performances are only

Case	1	2	3	4
Nodes	352819	67895	18375	3781
Elements	703542	133694	34662	7116
Timesteps	500	3000	10000	50000

Table 6.3: Case list

indicative of the real GPUs speed: in the real case an important contribution is also given by drivers and other system bottlenecks.

Mesh list for benchmarks The set of meshes and the total number of time–steps used in benchmarks are shown in table 6.3. All meshes discretize the same geometry, i.e. the NACA 0012 airfoil.

The biggest mesh has more than 700 000 elements while the smallest is appriximately 100 times smaller. Benchmarks were conducted with different number of time–steps on each mesh in order to finish each simulation in approximately the same time. This helps to figure out how well ExPReS scales with the problem size.

6.7.3 Results

Multiple cases with different meshes and different hardware were investigated. The results of the obtained speedups are compared in tables 6.4, 6.5, 6.6, 6.7.

Comments One curious aspect outlined by these results is the performance improvement provided by HyperThreading. In these tests only the i5 2410m processor can take advantage of this technology. For example in case 3 using 2 threads a speedup of 1.84 was obtained over the serial version of ExPReS , while using 4 threads with HyperThreading leads to a speedup of 2.58. Of course this speedup is not the same as the one provided by two additional physical core as shown for example by the i5 760, but the advantages are however appreciable. A true quad–core as the i5 760, in fact, provides a bigger speedup when 4 threads are used. However even in this case the scaling is not perfectly linear. This behavior is mainly due to the overhead related to the parallel architecture. For example the i5 760 with 4 threads provides a speedup of 3.12 over the serial version while theoretically, without any overhead, a speedup of 4 would be expected. With the same CPU, the use of 2 threads leads to a speedup of 1.95 which is nearly the value of 2 expected with a theoretical linear scaling. With the Phenom II X4 840 CPU a linear scaling is more visible. For example in case 4 a speedup of 1.99 is reached

	Phen. (1T)	(2T)	(4T)	i5 760 (1T)	(2T)	(4T)	i5 2410m (1T)	(2T)	(4T) HT	GT 540M	GTX 460
Phen. (1T)	1.00	0.51	0.31	0.58	0.34	0.21	0.72	0.41	0.30	0.26	0.07
(2T)	1.97	1.00	0.60	1.14	0.66	0.42	1.42	0.80	0.58	0.51	0.13
(4T)	3.28	1.66	1.00	1.89	1.10	0.70	2.36	1.33	0.97	0.85	0.22
i5 760 (1T)	1.73	0.88	0.53	1.00	0.58	0.37	1.25	0.70	0.51	0.45	0.11
(2T)	2.98	1.51	0.91	1.72	1.00	0.64	2.14	1.21	0.88	0.77	0.20
(4T)	4.69	2.38	1.43	2.70	1.57	1.00	3.37	1.90	1.39	1.22	0.31
i5 2410m (1T)	1.39	0.71	0.42	0.80	0.47	0.30	1.00	0.56	0.41	0.36	0.09
(2T)	2.46	1.25	0.75	1.42	0.83	0.53	1.77	1.00	0.73	0.64	0.16
(4T) HT	3.38	1.71	1.03	1.95	1.13	0.72	2.43	1.37	1.00	0.88	0.22
GT 540M	3.86	1.96	1.18	2.22	1.29	0.82	2.78	1.57	1.14	1.00	0.25
GTX 460	15.13	7.69	4.62	8.73	5.08	3.23	10.89	6.14	4.48	3.92	1.00

TIME	645.50	327.79	196.90	372.20	216.63	137.69	464.54	261.96	191.17	167.33	42.65

Table 6.4: Case 1, speedups

	Phen. (1T)	(2T)	(4T)	i5 760 (1T)	(2T)	(4T)	i5 2410m (1T)	(2T)	(4T) HT	GT 540M	GTX 460
Phen. (1T)	1.00	0.52	0.35	0.57	0.30	0.20	0.65	0.38	0.30	0.24	0.06
(2T)	1.91	1.00	0.67	1.10	0.58	0.38	1.25	0.73	0.57	0.45	0.11
(4T)	2.85	1.49	1.00	1.64	0.86	0.57	1.87	1.09	0.85	0.67	0.17
i5 760 (1T)	1.74	0.91	0.61	1.00	0.53	0.35	1.14	0.67	0.52	0.41	0.10
(2T)	3.32	1.73	1.16	1.90	1.00	0.66	2.17	1.27	0.99	0.78	0.19
(4T)	5.04	2.63	1.77	2.89	1.52	1.00	3.30	1.93	1.50	1.19	0.30
i5 2410m (1T)	1.53	0.80	0.54	0.88	0.46	0.30	1.00	0.58	0.46	0.36	0.09
(2T)	2.61	1.36	0.92	1.50	0.79	0.52	1.71	1.00	0.78	0.61	0.15
(4T) HT	3.36	1.75	1.18	1.92	1.01	0.67	2.20	1.29	1.00	0.79	0.20
GT 540M	4.25	2.22	1.49	2.44	1.28	0.84	2.78	1.63	1.27	1.00	0.25
GTX 460	17.05	8.90	5.98	9.77	5.14	3.38	11.16	6.53	5.08	4.01	1.00

TIME	636.34	332.37	223.08	364.84	191.82	126.16	416.61	243.67	189.57	149.69	37.33

Table 6.5: Case 2, speedups

	Phen. (1T)	(2T)	(4T)	i5 760 (1T)	(2T)	(4T)	i5 2410m (1T)	(2T)	(4T) HT	GT 540M	GTX 460
Phen. (1T)	1.00	0.70	0.29	0.66	0.34	0.21	0.76	0.41	0.29	0.17	0.05
(2T)	1.43	1.00	0.42	0.94	0.48	0.30	1.09	0.59	0.42	0.24	0.07
(4T)	3.42	2.39	1.00	2.25	1.15	0.72	2.60	1.41	1.01	0.57	0.17
i5 760 (1T)	1.52	1.06	0.44	1.00	0.51	0.32	1.16	0.63	0.45	0.25	0.07
(2T)	2.97	2.08	0.87	1.95	1.00	0.63	2.26	1.23	0.88	0.50	0.14
(4T)	4.74	3.32	1.39	3.12	1.59	1.00	3.60	1.95	1.40	0.79	0.23
i5 2410m (1T)	1.32	0.92	0.38	0.87	0.44	0.28	1.00	0.54	0.39	0.22	0.06
(2T)	2.43	1.70	0.71	1.60	0.82	0.51	1.84	1.00	0.71	0.41	0.12
(4T) HT	3.39	2.38	0.99	2.23	1.14	0.72	2.58	1.40	1.00	0.57	0.17
GT 540M	5.99	4.19	1.75	3.94	2.01	1.26	4.55	2.47	1.76	1.00	0.29
GTX 460	20.54	14.37	6.01	13.50	6.91	4.33	15.61	8.46	6.05	3.43	1.00

TIME	459.94	231.85	134.52	302.37	154.68	97.01	349.48	189.52	135.48	76.77	22.39

Table 6.6: Case 3, speedups

	Phen. (1T)	(2T)	(4T)	i5 760 (1T)	(2T)	(4T)	i5 2410m (1T)	(2T)	(4T) HT	GT 540M	GTX 460
Phen. (1T)	1.00	0.50	0.28	0.69	0.35	0.22	0.80	0.43	0.32	0.21	0.07
(2T)	1.99	1.00	0.56	1.37	0.70	0.45	1.60	0.86	0.63	0.41	0.13
(4T)	3.58	1.80	1.00	2.46	1.26	0.80	2.88	1.55	1.14	0.74	0.24
i5 760 (1T)	1.45	0.73	0.41	1.00	0.51	0.33	1.17	0.63	0.46	0.30	0.10
(2T)	2.84	1.42	0.79	1.95	1.00	0.64	2.28	1.23	0.90	0.59	0.19
(4T)	4.46	2.24	1.24	3.07	1.57	1.00	3.58	1.93	1.41	0.92	0.30
i5 2410m (1T)	1.24	0.62	0.35	0.86	0.44	0.28	1.00	0.54	0.39	0.26	0.08
(2T)	2.31	1.16	0.64	1.59	0.81	0.52	1.85	1.00	0.73	0.48	0.15
(4T) HT	3.15	1.58	0.88	2.17	1.11	0.71	2.53	1.37	1.00	0.65	0.21
GT 540M	4.83	2.42	1.35	3.32	1.70	1.08	3.88	2.10	1.53	1.00	0.32
GTX 460	14.97	7.51	4.18	10.30	5.28	3.36	12.03	6.49	4.75	3.10	1.00

TIME	436.37	218.95	121.83	300.21	153.80	97.91	350.74	189.28	138.39	90.34	29.15

Table 6.7: Case 4, speedups

with 2 threads and a speedup of 3.58 is reached with 4 threads. However this CPU has proved to be in general slower than others. This is probably due to the lack of the L3 cache. In numerical simulations a big cache is very important because its function is to prevent the CPU access to the RAM when the same data is required very often, avoiding possibles bottlenecks. The technology behind the cache memory allows it to run faster than the RAM but a bigger cache means a higher cost of the CPU. This is why in table 6.1 it is possible to see that the Phenom CPU was sold at approximately half the price of the i5 760. In ExPReS a big cache is important because some vectors, like the ones that contain the mesh topology, are very often accessed by cores.

Tables 6.4, 6.5, 6.6, 6.7 can be compared to see the scaling capabilities of ExPReS between meshes of different sizes. From these comparisons it is possible to see the trend of the speedups given by the GPU over the CPU between different meshes. During the tests one of the machine was equipped with the i5 760 CPU and the GTX 460 GPU. These components belongs to the same time period and price category, so they are well suited for a comparison. The obtained speedups of the GPU over the CPU with 4 thread are respectively, from the coarsest mesh to the finest one: 3.36, 4.33, 3.38, 3.23. This means that the trend is not monotone. In fact it seems that there is a particular mesh dimension capable of maximizing the GPU/CPU speed ratio. This mesh is the one with approximately 35000 elements. One probable reason for this behavior could be the combination of the limited amount of the cache in the GPUs architecture with the overhead related for example to the host–device communications and kernel managing. This means that when the mesh is too coarse the cache doesn't saturate but some overheads can lead to bottlenecks. Then, when the mesh is too fine, overheads are negligible, but if the cache is full some data transfer between the slower global memory and the GPU are needed. An important difference between the CPU architecture and the GPU architecture is in fact the dimension of the cache memory, as shown in tables 6.2 and 6.1. The same thing is valid for the i5 2410m CPU and the GT 540M GPU that ships with the notebook used for benchmarks. The obtained speedups are: 1.53, 1.76, 1.27, 1.14. Even in this case it is possible to see that the maximum speedup is obtained with the mesh of the case 2, the one with approximately 35000 elements.

Another important aspect is the comparison between the GPUs used in these benchmarks. In fact as it can be seen in table 6.2, there is a considerable difference between the computational power of the two GPUs. This is evident because, from the hardware point of view, both GPUs are based on the same architecture (a revision of Fermi) but the GTX 460 has 7 Stream Multiprocessors, 5 more than the GT 540M (or 3.5 times more than the GT

540M). Moreover the GTX 460 uses the GDDR5 technology for the global memory while the GT 540M uses the cheaper and slower DDR3 technology. This is specially important when the cache is full and thus it is needed to access to the global memory location to recover the requested data. The declared performances in GFLOPS, provided by table 6.2, highlight that the GTX 460 is 3–4 times faster that the GT 540M. This is only an indicative index of performance and is related only to single precision, while during these benchmarks ExPReS was executed with double precision support. However tests reveals that the GTX 460 is in effect from 3.43 to 4.01 times faster than the GT 540M.

The multithreaded version of ExPReS seems to scale very well on a shared memory system provided by a multi–core CPU. Moreover the tested GPUs of the same price category and time period of the tested CPUs demonstrated the tangible benefits of the GPGPU approach.

Figure 6.20 shows the speedups given by the multithreaded and the GPU version of ExPReS over the serial one for each test case with the i5 760 CPU and the GTX 460 GPU. Figure 6.21 shows the speedups given by the multithreaded and the GPU version of ExPReS over the serial one for each case with the i5 2410m CPU and the GT 540M GPU. It is clearly visible the superiority of the GTX 460 in numerical computations and thus the advantages of the GPGPU. Of course this is a only a mid–range gaming graphics card: speedups become more relevant when using a high–end GPU specifically developed for general purpose computations.

6.7.4 Comparison with other software

In order to investigate the capability of ExPReS to take advantage of multi–core CPUs and GPUs architecture, it is possible to make a comparison between the ExPReS benchmarks and those advertised by some companies about their commercial software when the hardware acceleration provided by the GPU is active.

It must be noted that usually commercial software benchmarks are strictly related to an implementation on a cluster or a supercomputer and thus high–range hardware components are used for tests, such as Intel Xeon, AMD Opteron, NVIDIA Tesla. These components can easly exceed $ 1000. However a comparison with speedups obtained with mid–range hardware is still interesting.

Another important thing is that a direct comparison like this is not strictly correct because different software can solve different problems or even the same problem with different solution schemes. Thus the algorithm implemented in different softwares could have very different scalability capa-

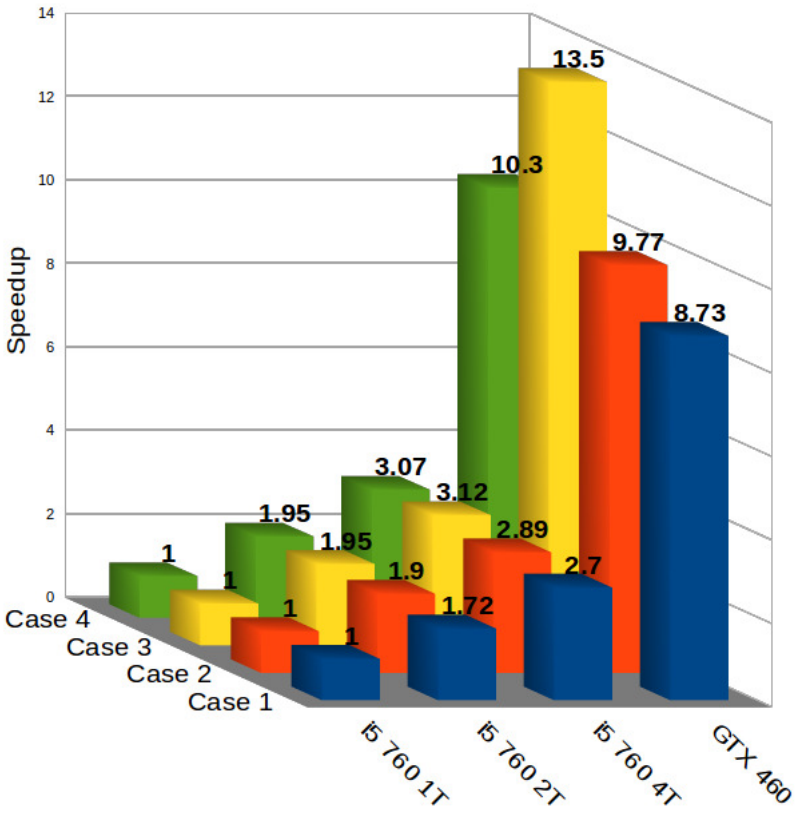


Figure 6.20: Results for the mid–range desktop configuration. Single thread, multiple threads and GPU executions.

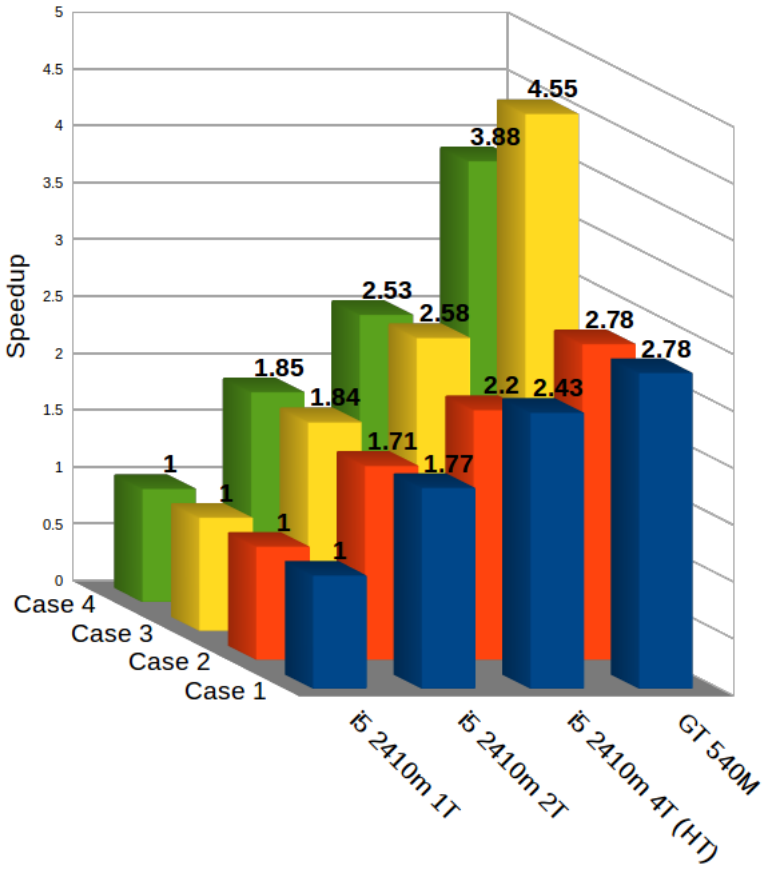


Figure 6.21: Results for the mid–range notebook configuration. Single thread, multiple threads and GPU executions.

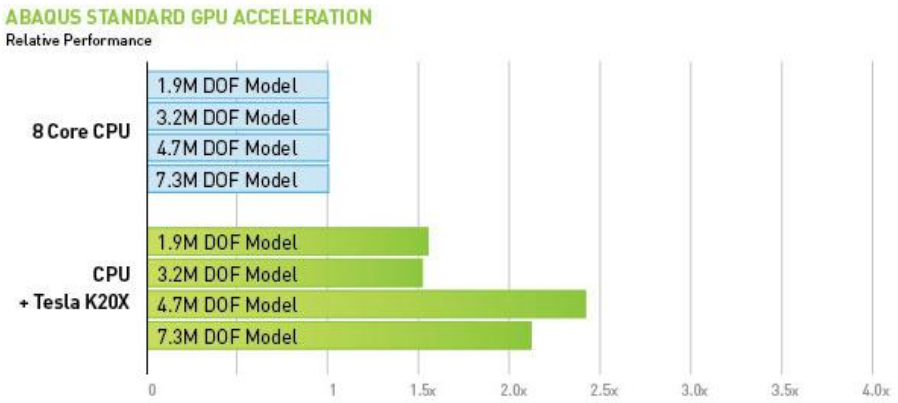


Figure 6.22: Speedups for SIMULIA Abacus [60].

bilities. For example a problem whose algorithm can take advantage of the local memory of the GPU is more likely to obtain a bigger speedup over the CPU. This means that a problem like a Montecarlo simulation can easily perform 100x faster with the GPU in respect to the CPU, while a structural and an aerodynamic solver generally shows lower speedups. Moreover when two software solve the same problem differences can arise, for example, between an implicit and an explicit scheme as stated in 5.4.

Here, some benchmark results provided by NVIDIA from its website [60] for numerical simulations software are reported for a comparison with ExPReS scalability capabilities.

Figure 6.22 shows the results advertised for SIMULIA Abacus, a structural solver. Tests were conduced on an Intel 8–core 2.6 GHz Sandy Bridge CPU and a NVIDIA Tesla K20X GPU. Viewing the results a maximum speedup of 2.5 is obtained when a single GPU is compared with a single multi–core CPU.

Figure 6.23 shows the speedups advertised for ANSYS Mechanical and ANSYS Fluent. Tests were conducted on various Intel Xeon (Westmere and Sandy Bridge) CPUs and NVIDIA Tesla GPUs (K20, K20X and C2075). The most important thing is that in all cases the speedups obtained varies from 1.9 to 2.4.

Figure 6.24 shows the speedups advertised for MSC Nastran. Tests were conducted on Intel Xeon (Sandy Bridge) CPUs and NVIDIA Tesla (K20 and K20X) GPUs. In the figure it is possible to see that the GPU solver is approximately 6x faster than the serial CPU solver and about 2x faster than the parallel CPU (8–core) version for the static structural solver (SOL 101). For the structural modal solver, SOL 103, the GPU is approximately

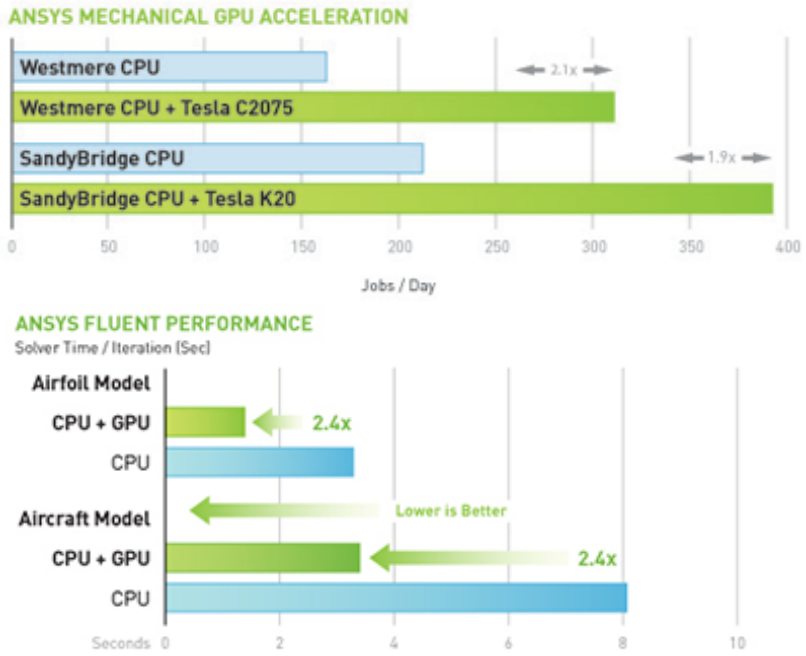


Figure 6.23: Speedups for ANSYS Mechanical and ANSYS Fluent [60].

3x faster than the CPU executing the serial solver and approximately 1.5x faster than the CPU parallel version.

In MathWorks MATLAB it is possible to analyze the benefits of the GPGPU launching an integrated benchmarks that compares the GPU and CPU performance for matrix computations both in single and double precision. The benchmark is based on how fast the hardware can solve a linear system (A\b) with different matrix sizes.

It must be noted that at the present time MATLAB only supports NVIDIA devices with double precision capabilities. The desktop configuration used for the benchmarks is the same machine used for ExPReS benchmarks: Intel i5 760 CPU and NVIDIA GTX 460 GPU. Figure 6.25 shows the results of the speedups of the GPU over the CPU with different matrix sizes. It is possible to see that the benefits of the GPGPU approach increase with the problem dimension, due to the relative reduction of overhead in respect to the effective computation. Figure 6.26 compares the performance of the CPU and the GPU with single precision matrix computations and with different matrix sizes. Figure 6.27 compares the performance of the CPU and the GPU with double precision matrix computations and different matrix sizes.

It is possible to see that a maximum speedup of about 5 with the single precision and of approximately 2 with the double precision can be achieved

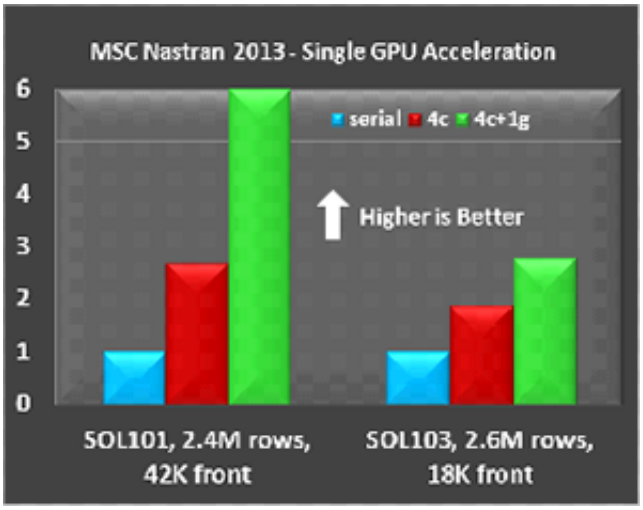


Figure 6.24: Speedups for MSC Nastran [60].

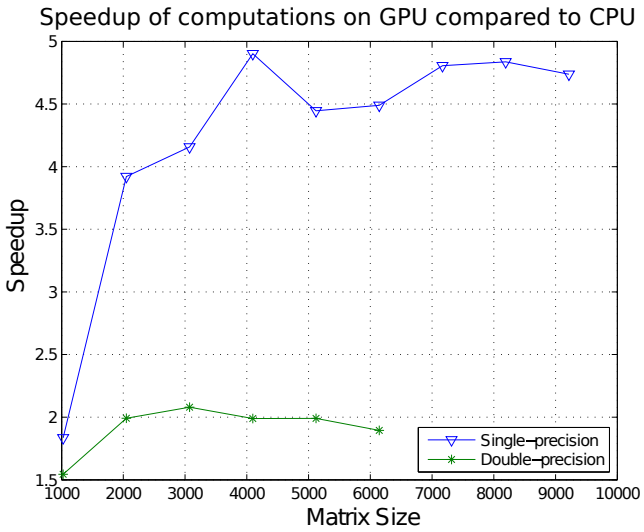


Figure 6.25: MathWorks MATLAB, speedups of the GPU over the CPU, single and double precision.

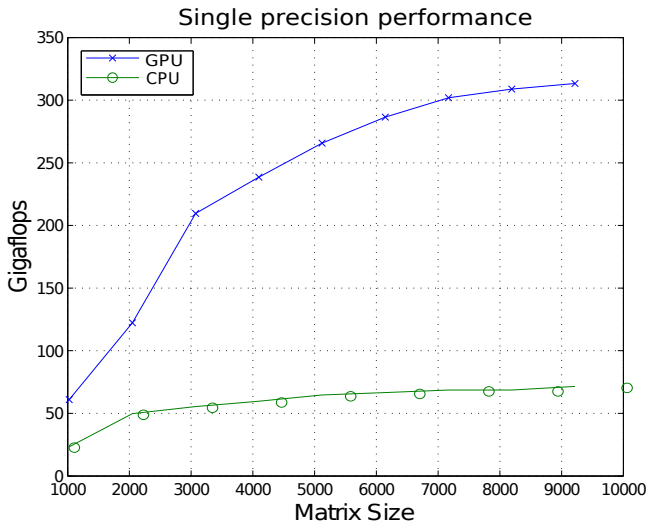


Figure 6.26: MathWorks MATLAB, single precision performance, CPU and GPU.

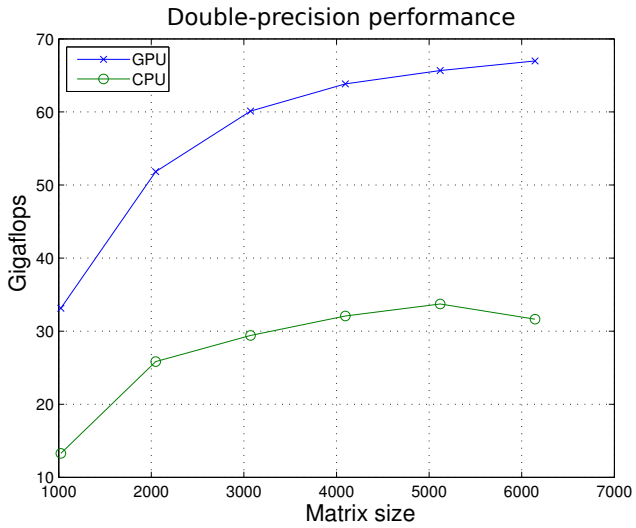


Figure 6.27: MathWorks MATLAB, double precision performance, CPU and GPU.

using GPGPU in MATLAB.

Comments Looking at the benchmarks results of commercial software related to structural and fluid dynamic problems it is possible to see that the gains achieved with the use of GPUs are such that a speed up of about an order of magnitude can be expected over the CPU. This is in perfect agreement with the scalability provided by ExPReS.

Chapter 7

Concluding remarks

In this work the 2–fields full potential formulation was successfully applied to real gas flows. The use of this particular formulation was justified by a deep comparison with the more standard 1–field approach. The 2–fields scheme is more versatile and can easily be used with a general enthalpy function. With this formulation it is therefore possible and quite simple to study Bethe–Zel'dovich–Thompson (BZT) fluids and their particular behaviors when their thermodynamic state is near the critical point. Two different gas models were introduced from general thermodynamics principles: the polytropic ideal gas, that is suitable for common aerodynamic flows, and the simplest BZT gas model, the polytropic van der Waals gas. The Bernoulli theorem was written for a general gas and the enthalpy variation through an isentropic transformation expressed as a function of the density only, thus closing the 2–fields problem.

This formulation was implemented from scratch in the solver ExPReS, that is based on a cell–centered finite volume discretization and an explicit time stepping scheme. The cell–centered approach was chosen since the solver was written to run on graphic processors. It in fact allows to avoid the typical problem of the branch divergence that causes a loss of computational efficiency.

Different numerical schemes to approximate the gradient and some upwinding techniques were tested. The biggest problem faced during this work was to find a scheme to reconstruct the gradient accurate and robust enough to keep the numerical solution procedure stable. In fact, the full potential problem strongly relies on the accuracy and robustness of the gradient reconstruction schemes and, even if some of them were successfully applied to the computing of viscous fluxes, the same schemes did not proved to be stable when applied to the full potential problem. This is probably the main reason for the big difficulties faced in the 3D implementation that made it very com-

plicated to compute a 3D subsonic solution and impossible (up to now) to find a stationary transonic solution. On the contrary, node–centered schemes proved to be more accurate and robust. Therefore, the implementation of a node–centered discretization could be a possible future development. It could be interesting to evaluate if the advantages provided by the more robust gradient reconstruction techniques could overcome the loss of efficiency due to the branch divergence.

Two versions of ExPReS were developed. The first one runs on CPU and is parallelized with a multithread approach using OpenMP. The second version was programmed with the OpenCL language to take advantage of the acceleration provided by GPU devices.

The program was firstly validated solving ideal gas flows and comparing results with solutions provided by another full potential solver or Euler equations solvers. Dense gas flows were then computed and compared to those available in literature and to those computed with the solver S^T that implements a time implicit 2–fields full potential formulation. Results proved the validity to use a potential flow solver to compute complicate flows with thermodynamic conditions near the critical point. Moreover, the cell–centered approach has proved successful to compute such solutions, even if it is more complicated and less robust than the node–centered discretization.

Typical BZT gas phenomena were observed with a good correspondence with solutions obtained solving Euler equations, especially in subsonic flows. Transonic dense gas flows with weak shocks showed to be quite accurate while, when strong shock waves forms in the flow field, the isentropic assumption generates big errors. Anyway, these errors could be eliminated introducing an entropy layer as done in [5]. This technique should however be tested in dense gas flows.

Two different strategies to simulate the motion of bodies in the field were implemented. The simplest is the adoption of transpiration boundary conditions that suits only small displacements. On the contrary, a technique that allows to take into account arbitrary large motions is the ALE formulation. The used form relies on the assumption that the mesh velocity is solenoidal (geometric conservation). Since only rigid mesh displacements were considered, this assumption is always verified. It could be however possible to adopt an ALE formulation that does not need this property, as explained in appendix A.

ExPReS was extended to compute 3D flows, but many stability problems were experienced. They are probably related to the gradient reconstruction schemes. This consideration relies on the fact that, if the Bernoulli theorem is deactivated and a transport field is assigned, it is possible to transport a wavefront in an empty domain without facing any numerical instability.

This analysis probably enforces the hypothesis that these instabilities are related to the gradient reconstruction inaccuracies, rather than the 2–fields formulation itself, since it has already been successfully applied to the computing of 3D flows using a node centered formulation that generally provides better accuracy in the approximation of gradients. It is however possible that all these convergence problems are related to a programming bug. The only computed solution concerns a subsonic lifting flow around the ONERA M6 wing. It was possible to compute this solution only by applying a filter that dissipates perturbations by averaging the solution on a cell with its neighboring cells that share with it a node.

Benchmarks were performed on different meshes to analyze the speedups obtained using general purpose GPU computing. The obtained speedups strongly vary depending on the used hardware. Comparisons were made between hardware of the same price range and production period. The best result was the speed up achieved with the graphic card GTX 460 compared to the processor i5-760. The simulation executed on this GPU was 13.5 times faster than the one–thread and 4.32 times than the four–threaded simulations that run on the CPU. To obtain these speedups using a multi-processor approach would require a much bigger investment than that necessary for a mid-range graphic card as the one used for these benkmarks. Moreover, the computational power of GPUs will hopefully increase rapidly in the next years since the major GPU producers of the world are strongly investing in this technology. It's therefore probable that GPGPU computing will offer a high computational power at lower prices in the future.

Appendix A

ALE formulation

The Arbitrary Lagrangian Eulerian (ALE) formulation is useful when a fluid dynamic solution must be computed around a moving and deforming body. This technique allows to approximate the Eulerian unknowns using a discretization that is not as usual on a fixed mesh, but on a grid that follows the body displacements. It is therefore a well suited approach to take into account fluid-structure interactions and to use modern CFD aerodynamic models to compute aeroelastic solutions.

In this appendix, after a brief introduction to the Eulerian and the Lagrangian points of view, the fundamental relation that allows to switch between them is introduced. This relation connects their time derivatives, allowing the expression of the conservation laws in both of them. In section §A.4, conservation laws are expressed on a moving volume, and the ALE form of a generic conservation law is presented. Finally, in section §A.5, the ALE formulation is specialized to the mass conservation equation and the Bernoulli theorem, obtaining the 2–fields ALE full potential problem.

A.1 Eulerian and Lagrangian points of view

These two approaches differ primarily for their typical application field. In fact, in fluid-dynamics the generally preferred approach is the Eulerian one because it is more indicated when particles are subjected to big displacements and the interest is generally focused on a fixed domain through which the fluid flows. On the contrary, the Lagrangian point of view is commonly adopted in structural simulations because in that case the focus is on the material particles displacements that are directly used to compute deformations and therefore stresses in the structure.

Consider the general physical quantity $\mathcal{F}$, that depends from space and time (for example the temperature in a room or in a solid). In the Eulerian

approach $\mathcal{F}$ is described by the mathematical function

$$f(\boldsymbol{r}, t), \tag{A.1}$$

whose independent variables are the space position $\boldsymbol{r}$ and the time t. The space position $\boldsymbol{r}$ has nothing to do with a material particle and if it is held fixed in $\boldsymbol{r} = \bar{\boldsymbol{r}}$, the function

$$\bar{f}(t) = f(\bar{\boldsymbol{r}}, t) = f(\boldsymbol{r}, t)|_{\boldsymbol{r}=\bar{\boldsymbol{r}}} \tag{A.2}$$

gives the value of the physical quantity $\mathcal{F}$ for all the particles that pass by the position $\bar{\boldsymbol{r}}$ at different time instants.

On the contrary, the Lagrangian representation of the quantity $\mathcal{F}$, is given by the mathematical function

$$F(\boldsymbol{X}_0, t), \tag{A.3}$$

whose independent variables are the initial particles positions $\boldsymbol{X}_0$ and the time t. If we consider the particular initial position $\bar{\boldsymbol{X}}_0$, the function

$$\bar{F}(t) = F\left(\bar{\boldsymbol{X}}_0, t\right) = F(\boldsymbol{X}_0, t)|_{\boldsymbol{X}_0=\bar{\boldsymbol{X}}_0} \tag{A.4}$$

gives the quantity $\mathcal{F}$ "felt" by the particle that at the initial time t_0 was in the position $\bar{\boldsymbol{X}}_0$.

To connect the Eulerian and the Lagrangian approaches, it is necessary to know the trajectory of all material particles, expressed as

$$\boldsymbol{R}(\boldsymbol{X}_0, t). \tag{A.5}$$

It should be clear that $\boldsymbol{R}(\boldsymbol{X}_0, t_0) = \boldsymbol{X}_0$. The function F can then be obtained evaluating the function f on these trajectories

$$F(\boldsymbol{X}_0, t) = f(\boldsymbol{R}(\boldsymbol{X}_0, t), t) = f(\boldsymbol{r}, t)|_{\boldsymbol{r}=\boldsymbol{R}(\boldsymbol{X}_0,t)}. \tag{A.6}$$

The inverse operation is always possible too since the function $\boldsymbol{R}$ must be invertible in order to avoid disappearance or annihilation of material particles:

$$f(\boldsymbol{r}, t) = F\left(\boldsymbol{R}^{-1}(\boldsymbol{r}, t), t\right) = F(\boldsymbol{X}_0, t)|_{\boldsymbol{X}_0=\boldsymbol{R}^{-1}(\boldsymbol{r},t)}, \tag{A.7}$$

where $\boldsymbol{R}^{-1}(\boldsymbol{r}, t)$ is the inverse of the function $\boldsymbol{r} = \boldsymbol{R}(\boldsymbol{X}_0, t)$ solved for the first independent variable. This particular function gives the initial position of the particle that at the time instant t resides at position $\boldsymbol{r}$.

A.2 Time derivatives

To connect the time derivatives expressed in the two approaches it is necessary to introduce a preliminary concept. Consider a probe that moves through the domain. Its measure of the quantity $\mathcal{F}$ is given by the function

$$F_P(t) = f(\boldsymbol{X}_P(t), t) = f(\boldsymbol{r}, t)|_{\boldsymbol{r}=\boldsymbol{X}_P(t)}, \tag{A.8}$$

where $\boldsymbol{X}_P(t)$ denotes the probe trajectory and is completely not related to the materials particles motion field.

The time derivative of that measure is then computed applying the chain rule as it follows

$$\begin{aligned} \frac{d F_P}{d t} &= \boldsymbol{\nabla} f \Big|_{\boldsymbol{r}=\boldsymbol{X}_P(t)} \cdot \frac{d \boldsymbol{X}_P}{d t} + \frac{\partial f}{\partial t}\Big|_{\boldsymbol{r}=\boldsymbol{X}_P(t)} \\ &= \boldsymbol{\nabla} f \Big|_{\boldsymbol{r}=\boldsymbol{X}_P(t)} \cdot \boldsymbol{V}_P(t) + \frac{\partial f}{\partial t}\Big|_{\boldsymbol{r}=\boldsymbol{X}_P(t)}, \end{aligned} \tag{A.9}$$

where $\boldsymbol{V}_P(t)$ is the probe velocity.

To express the right hand side of equation (A.9), a more compact way is to introduce the operator

$$\frac{D_{\boldsymbol{V}_P} f}{D t} = \boldsymbol{\nabla} f \cdot \boldsymbol{V}_P(t) + \frac{\partial f}{\partial t}. \tag{A.10}$$

The derivative $\dfrac{D_{\boldsymbol{V}_P} f}{D t}$ is a function of $\boldsymbol{r}$ and t since $f = f(\boldsymbol{r}, t)$. Equation (A.9) can then be reformulated as

$$\frac{d F_P}{d t} = \frac{D_{\boldsymbol{V}_P} f}{D t}\Big|_{\boldsymbol{r}=\boldsymbol{X}_P(t)}. \tag{A.11}$$

To obtain the variation rapidity of $\mathcal{F}$ not for a single probe that moves with velocity $\boldsymbol{V}_P$, but on a grid of points (eventually infinite in number) that at the time instant t moves with the velocity field $\boldsymbol{v}_g$ expressed in Eulerian form as $\boldsymbol{v}_g = \boldsymbol{v}_g(\boldsymbol{r}, t)$, expression (A.10) is simply extended to

$$\frac{D_{\boldsymbol{v}_g} f}{D t} = \boldsymbol{\nabla} f \cdot \boldsymbol{v}_g(\boldsymbol{r}, t) + \frac{\partial f}{\partial t}. \tag{A.12}$$

Material Derivative The material derivative is the variation rapidity of $\mathcal{F}$ felt by material particles that at time instant t are in position $\boldsymbol{r}$ and move with the velocity field $\boldsymbol{u}(\boldsymbol{r}, t)$, that is the speed of the fluid. This is therefore a simple extension of equation (A.12), that is

$$\frac{D f}{D t} = \frac{D_{\boldsymbol{u}} f}{D t} = \boldsymbol{\nabla} f \cdot \boldsymbol{u}(\boldsymbol{r}, t) + \frac{\partial f}{\partial t}. \tag{A.13}$$

The operator $\frac{D}{Dt}$ takes as input the Eulerian field f and gives as output another Eulerian field and not a Lagrangian one. However this operator is used to connect the two points of view since, if it is evaluated for $\boldsymbol{r} = \boldsymbol{R}(\boldsymbol{X}_0, t)$, it gives the Lagrangian field $\dfrac{\partial F}{\partial t}$.

$$\frac{\partial F}{\partial t} = \left.\frac{D f}{D t}\right|_{\boldsymbol{r}=\boldsymbol{R}(\boldsymbol{X}_0, t)}. \tag{A.14}$$

To conclude the correspondences between the Eulerian and the Lagrangian points of view, the relation between the two velocity fields expressed in both approaches is

$$\boldsymbol{U}(\boldsymbol{X}_0, t) = \frac{\partial \boldsymbol{R}}{\partial t} = \boldsymbol{u}(\boldsymbol{r}, t)|_{\boldsymbol{r}=\boldsymbol{R}(\boldsymbol{X}_0, t)}. \tag{A.15}$$

A.3 Second order time derivatives

In section §2.4.5, the ALE form of the second order time derivative has been introduced. To infer it, it is necessary to compute the second order derivative of equation (A.8). The derivative of equation (A.9) is

$$\begin{aligned}\frac{d^2 F_I}{d t^2} = \frac{d}{dt}\left[\boldsymbol{\nabla} f\Big|_{\boldsymbol{r}=\boldsymbol{X}_I(t)}\right] \cdot \boldsymbol{V}_I(t) + \boldsymbol{\nabla} f\Big|_{\boldsymbol{r}=\boldsymbol{X}_I(t)} \cdot \frac{d \boldsymbol{V}_I}{d t} \\ + \frac{d}{dt}\left(\frac{\partial f}{\partial t}\Big|_{\boldsymbol{r}=\boldsymbol{X}_I(t)}\right).\end{aligned} \tag{A.16}$$

To compute the two ordinary time derivatives of $\boldsymbol{\nabla} f(\boldsymbol{r}, t)|_{\boldsymbol{r}=\boldsymbol{X}_I(t)}$ and of

$\left.\dfrac{\partial f}{\partial t}\right|_{\boldsymbol{r}=\boldsymbol{X}_I(t)}$ it is necessary to apply again the chain rule as follows

$$\begin{aligned}
\frac{d}{dt}\left[\boldsymbol{\nabla} f\Big|_{\boldsymbol{r}=\boldsymbol{X}_I(t)}\right] &= \boldsymbol{\nabla}\left(\boldsymbol{\nabla} f\right)|_{\boldsymbol{r}=\boldsymbol{X}_I(t)} \cdot \boldsymbol{V}_I(t) + \left.\frac{\partial \boldsymbol{\nabla} f}{\partial t}\right|_{\boldsymbol{r}=\boldsymbol{X}_I(t)} \\
&= \boldsymbol{\nabla}\left(\boldsymbol{\nabla} f\right)|_{\boldsymbol{r}=\boldsymbol{X}_I(t)} \cdot \boldsymbol{V}_I(t) + \left.\left(\boldsymbol{\nabla}\frac{\partial f}{\partial t}\right)\right|_{\boldsymbol{r}=\boldsymbol{X}_I(t)} \quad \text{(A,17a)} \\
\frac{d}{dt}\left[\left.\frac{\partial f}{\partial t}\right|_{\boldsymbol{r}=\boldsymbol{X}_I(t)}\right] &= \left.\left(\boldsymbol{\nabla}\frac{\partial f}{\partial t}\right)\right|_{\boldsymbol{r}=\boldsymbol{X}_I(t)} \cdot \boldsymbol{V}_I(t) + \left.\frac{\partial^2 f}{\partial t^2}\right|_{\boldsymbol{r}=\boldsymbol{X}_I(t)} . \quad \text{(A.17b)}
\end{aligned}$$

Substituting relations (A.17) in equation (A.16), the following expression is obtained

$$\begin{aligned}
\frac{d^2 F_P}{d t^2} &= \left[\boldsymbol{\nabla}\left(\boldsymbol{\nabla} f\right)|_{\boldsymbol{r}=\boldsymbol{X}_P(t)} \cdot \boldsymbol{V}_P(t)\right] \cdot \boldsymbol{V}_P(t) \\
&+ 2\left.\left(\boldsymbol{\nabla}\frac{\partial f}{\partial t}\right)\right|_{\boldsymbol{r}=\boldsymbol{X}_P(t)} \cdot \boldsymbol{V}_P(t) \\
&+ \left.\boldsymbol{\nabla} f\right|_{\boldsymbol{r}=\boldsymbol{X}_P(t)} \cdot \frac{d\boldsymbol{V}_P}{d t} + \left.\frac{\partial^2 f}{\partial t^2}\right|_{\boldsymbol{r}=\boldsymbol{X}_P(t)} . \qquad \text{(A.18)}
\end{aligned}$$

As done in section §A.2, the symbol $\dfrac{D^2_{\boldsymbol{V}_P}}{D\,t^2}$ is introduced to indicate the quantity

$$\begin{aligned}
\frac{D^2_{\boldsymbol{V}_P} f}{D\,t^2} &= \left[\boldsymbol{\nabla}\left(\boldsymbol{\nabla} f\right) \cdot \boldsymbol{V}_P(t)\right] \cdot \boldsymbol{V}_P(t) + 2\left(\boldsymbol{\nabla}\frac{\partial f}{\partial t}\right) \cdot \boldsymbol{V}_P(t) \\
&+ \boldsymbol{\nabla} f \cdot \frac{d\boldsymbol{V}_P}{d t} + \frac{\partial^2 f}{\partial t^2} \qquad \text{(A.19)}
\end{aligned}$$

that, introducing the probe acceleration $\boldsymbol{A}_P(t)$, can be rewritten as

$$\begin{aligned}
\frac{D^2_{\boldsymbol{V}_P} f}{D\,t^2} &= \left[\boldsymbol{\nabla}\left(\boldsymbol{\nabla} f\right) \cdot \boldsymbol{V}_P(t)\right] \cdot \boldsymbol{V}_P(t) + 2\left(\boldsymbol{\nabla}\frac{\partial f}{\partial t}\right) \cdot \boldsymbol{V}_P(t) \\
&+ \boldsymbol{\nabla} f \cdot \boldsymbol{A}_P(t) + \frac{\partial^2 f}{\partial t^2}. \qquad \text{(A.20)}
\end{aligned}$$

Therefore

$$\frac{d^2 F_P}{d t^2} = \left.\frac{D^2_{\boldsymbol{V}_P} f}{D\,t^2}\right|_{\boldsymbol{r}=\boldsymbol{X}_P(t)} . \qquad \text{(A.21)}$$

To obtain the variation rapidity of $\mathcal{F}$ not for a single probe that moves with velocity $\boldsymbol{V}_P$ and acceleration $\boldsymbol{A}_P$, but on a grid of points (eventually infinite in number) that at the time instant t moves with the velocity and acceleration fields$\boldsymbol{v}_g$ and $\boldsymbol{a}_g$, both expressed in Eulerian form as $\boldsymbol{v}_g = \boldsymbol{v}_g(\boldsymbol{r}, t)$ and $\boldsymbol{a}_g = \boldsymbol{a}_g(\boldsymbol{r}, t)$, expression (A.10) is extended to

$$\frac{D^2_{\boldsymbol{v}_g} f}{D t^2} = [\boldsymbol{\nabla}(\boldsymbol{\nabla} f) \cdot \boldsymbol{v}_g(\boldsymbol{r}, t)] \cdot \boldsymbol{v}_g(\boldsymbol{r}, t) + 2\left(\boldsymbol{\nabla}\frac{\partial f}{\partial t}\right) \cdot \boldsymbol{v}_g(\boldsymbol{r}, t) + \boldsymbol{\nabla} f \cdot \boldsymbol{a}_g(\boldsymbol{r}, t) + \frac{\partial^2 f}{\partial t^2} \tag{A.22}$$

that can be reformulated as

$$\frac{D^2_{\boldsymbol{v}_g} f}{D t^2} = \boldsymbol{\nabla}(\boldsymbol{\nabla} f) : (\boldsymbol{v}_g(\boldsymbol{r}, t) \otimes \boldsymbol{v}_g(\boldsymbol{r}, t)) + 2\left(\boldsymbol{\nabla}\frac{\partial f}{\partial t}\right) \cdot \boldsymbol{v}_g(\boldsymbol{r}, t) + \boldsymbol{\nabla} f \cdot \boldsymbol{a}_g(\boldsymbol{r}, t) + \frac{\partial^2 f}{\partial t^2}, \tag{A.23}$$

where with $\boldsymbol{a} \otimes \boldsymbol{b}$ is meant the tensor

$$(\boldsymbol{a} \otimes \boldsymbol{b})_{ij} = a_i b_j. \tag{A.24}$$

The quantity $\boldsymbol{a}_g(\boldsymbol{r}, t)$ is the grid acceleration expressed in Eulerian form and computed as

$$\boldsymbol{a}_g(\boldsymbol{r}, t) = \frac{D_{\boldsymbol{v}_g}\boldsymbol{v}_g}{D t} = \boldsymbol{\nabla}\boldsymbol{v}_g \cdot \boldsymbol{v}_g + \frac{\partial \boldsymbol{v}_g}{\partial t} \tag{A.25}$$

A.4 Conservation laws on moving volumes

The conservation of the general Eulerian field $f(\boldsymbol{r}, t)$ written on a moving control volumes has the form

$$\frac{d}{dt}\int_{V(t)} f = -\oint_{\partial V(t)} f\left(\boldsymbol{u} - \boldsymbol{v}^{\partial}\right) \cdot \hat{\boldsymbol{n}}, \tag{A.26}$$

where $\boldsymbol{u}$ is the fluid velocity and $\boldsymbol{v}^{\partial}$ is the speed of the boundary of $V(t)$. Equation (A.26) simply states that the variation of the quantity of $\mathcal{F}$ contained in $V(t)$ is equal to the inflow of $\mathcal{F}$ through the boundaries of the control volume.

As demonstrated in chapter 9 of [61], the theorem of time differentiation of an integral on a moving domain states that

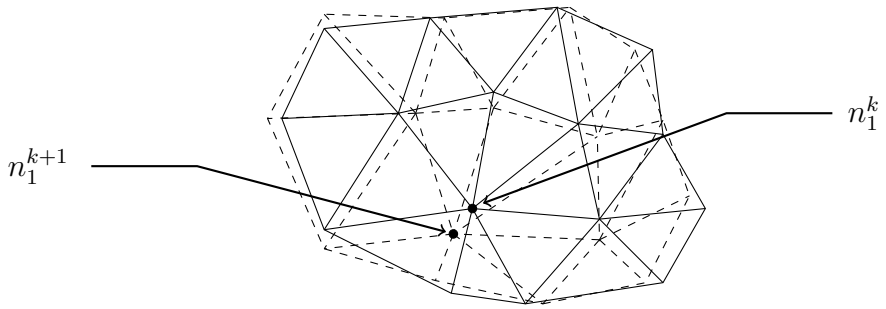


Figure A.1: Deforming mesh. The continuous triangulation is the mesh at time instant k while the dashed one is the mesh at time instant $k+1$

$$\frac{d}{dt}\int_{V(t)} f = \int_{V(t)} \frac{\partial f}{\partial t} + \oint_{\partial V(t)} f\, \boldsymbol{v}^{\partial} \cdot \hat{\boldsymbol{n}}, \tag{A.27}$$

that applied to equation (A.26) gives

$$\int_{V(t)} \frac{\partial f}{\partial t} + \oint_{\partial V(t)} f\, \boldsymbol{u} \cdot \hat{\boldsymbol{n}} = 0. \tag{A.28}$$

When this conservation law has to be numerically solved on a moving mesh, the time derivative that appears in equation (A.28) must be related to the time derivative computed with respect to the grid nodes. This derivative is given by

$$\frac{D_{\boldsymbol{v}_g} f}{D\, t} \tag{A.29}$$

that, approximated numerically on point 1 with the explicit Euler scheme, gives

$$\left.\frac{D_{\boldsymbol{v}_g} f}{D\, t}\right|_{\text{point 1}} \approx \frac{f_1^{k+1} - f_1^k}{\Delta t}, \tag{A.30}$$

where f_1^{k+1} is the solution in the position of the point 1 at time instant $k+1$ and f_1^k is the solution in the position of the point 1 at time instant k. These two positions are different as sketched in figure A.1.

Solving relation (A.12) for the partial time derivative gives

$$\frac{\partial f}{\partial t} = \frac{D_{\boldsymbol{v}_g} f}{D\, t} - \boldsymbol{\nabla} f \cdot \boldsymbol{v}_g \tag{A.31}$$

that, substituted in the conservation law (A.28), gives the first form of the ALE formulation of a generic conservation law

$$\int_{V(t)} \left(\frac{D_{\boldsymbol{v}_g} f}{D\, t} - \boldsymbol{\nabla} f \cdot \boldsymbol{v}_g \right) + \oint_{\partial V(t)} f\, \boldsymbol{u} \cdot \hat{\boldsymbol{n}} = 0. \tag{A.32}$$

Geometric conservation law When speaking about ALE formulation, a frequent and debated topic is the necessity of the geometric conservation property that requires that the mesh motion satisfies the divergence free constraint

$$\boldsymbol{\nabla} \cdot \boldsymbol{v}_g = 0. \tag{A.33}$$

The ALE form (A.32) does not need this hypothesis since in its deduction it was absolutely not required. On the contrary, hypothesis (A.33) is necessary to deduce the second ALE form of a conservation law. In fact, substituting equation (A.33) in the identity

$$\boldsymbol{\nabla} \cdot (f\, \boldsymbol{v}_g) = \boldsymbol{\nabla} f \cdot \boldsymbol{v}_g + f\, \boldsymbol{\nabla} \cdot \boldsymbol{v}_g, \tag{A.34}$$

gives

$$\boldsymbol{\nabla} f \cdot \boldsymbol{v}_g = \boldsymbol{\nabla} \cdot (f\, \boldsymbol{v}_g) \tag{A.35}$$

that, inserted in the first ALE form (A.32), with the use of the divergence theorem, gives the second ALE form

$$\int_{V(t)} \frac{D_{\boldsymbol{v}_g} f}{D\, t} + \oint_{\partial V(t)} f\, (\boldsymbol{u} - \boldsymbol{v}_g) \cdot \hat{\boldsymbol{n}} = 0 \tag{A.36}$$

A.5 2–fields full potential problem

As explained in (A.4), the second ALE form of the mass conservation law is

$$\int_{V(t)} \frac{D_{\boldsymbol{v}_g} \rho}{D\, t} + \oint_{\partial V(t)} \rho\, (\boldsymbol{\nabla}\phi - \boldsymbol{v}_g) \cdot \hat{\boldsymbol{n}} = 0, \tag{A.37}$$

where the fluid velocity has been expressed, as usual, as the gradient of the kinetic potential.

To obtain the ALE formulation of the Bernoulli theorem, equation (A.31) is substituted in equation (2.20)

$$\begin{aligned} \frac{\partial \phi}{\partial t} + \frac{1}{2} \boldsymbol{\nabla}\phi \cdot \boldsymbol{\nabla}\phi + h_{is} &= \frac{1}{2} V_\infty^2 + h_\infty, \\ \frac{D_{\boldsymbol{v}_g} \phi}{D\, t} - \boldsymbol{\nabla}\phi \cdot \boldsymbol{v}_g + \frac{1}{2} \boldsymbol{\nabla}\phi \cdot \boldsymbol{\nabla}\phi + h_{is} &= \frac{1}{2} V_\infty^2 + h_\infty. \end{aligned} \tag{A.38}$$

The final ALE form of the 2–fields full potential flow problem that requires the conservation property is therefore

$$\begin{cases} \displaystyle\int_{V(t)} \frac{D_{\boldsymbol{v}_g}\rho}{D\,t} + \oint_{\partial V(t)} \rho\,(\boldsymbol{\nabla}\phi - \boldsymbol{v}_g)\cdot\hat{\boldsymbol{n}} = 0 \\ \\ \dfrac{D_{\boldsymbol{v}_g}\phi}{D\,t} - \boldsymbol{\nabla}\phi\cdot\boldsymbol{v}_g + \dfrac{1}{2}\boldsymbol{\nabla}\phi\cdot\boldsymbol{\nabla}\phi + h_{is} = \dfrac{1}{2}V_\infty^2 + h_\infty \end{cases} \tag{A.39}$$

Numerical approximation In this work, system (A.39) has been numerically approximated with the same techniques used in chapter 5. The final discretized form is

$$\begin{cases} R_i^{n+1} = R_i^n - \dfrac{\Delta t}{V_i}\displaystyle\sum_{j=1}^{N_f} \bar{\rho}_{i,j}^{\,n}\ \left(\boldsymbol{G}_{\Phi^n}^{F_{i,j}} - \boldsymbol{V}_{i,j}^n\right)\cdot\hat{\boldsymbol{n}}_{i,j}^n\ \Delta L_{i,j}^n \\ \\ \Phi_i^{n+1} = \Phi_i^n + \Delta t\left(\dfrac{1}{2}V_\infty^2 - \dfrac{1}{2}\boldsymbol{G}_{\Phi^n}^{C_i}\cdot\boldsymbol{G}_{\Phi^n}^{C_i} + \boldsymbol{G}_{\Phi^n}^{C_i}\cdot\boldsymbol{V}_i^n + h_\infty - h_{is}\left(R_i^{n+1}\right)\right) \end{cases} \tag{A.40}$$

where now the quantities $\Delta L_{i,j}$ and $\hat{\boldsymbol{n}}_{i,j}$ depend from the time instant since they are related to a moving and deforming mesh. On the contrary, the volume V_i does not change its measure (but could change its shape) given that problem (A.39) requires the geometric conservation property (A.33). Finally, the symbol $\boldsymbol{V}_{i,j}^n$ and $\boldsymbol{V}_i^n$ denotes[1] respectively the numerically reconstructed mesh velocity on the interface i, j and on the cell i.

[1]The reader should not get confused by the symbol $\boldsymbol{V}_i$ that is a vector and expresses the velocity of the grid and the symbol V_i that is a scalar and denotes the finite volume measure

Appendix B

The fundamental derivative of gas dynamics

In this brief appendix the expression for the fundamental derivative of gas dynamics is deduced for the two cases of polytropic ideal gas and polytropic van der Waals gas.

B.1 Polytropic ideal gas

The expression $\Gamma(s,\,\rho)$ is now deduced starting from equation (3.71), rewritten here for completeness.

$$\Gamma(s,\,\rho) = 1 + \frac{\rho}{c(s,\,\rho)} \left(\frac{\partial c}{\partial \rho}\right)_s .$$

It is therefore necessary to compute the function $c(s,\,\rho)$, that is by definition

$$c(s,\,\rho) = \left(\frac{\partial P}{\partial \rho}\right)_s^{1/2} = \left\{\frac{\partial}{\partial \rho}\left[-\left(\frac{\partial e}{\partial v}\right)_s\Big|_{v=1/\rho}\right]\right\}^{1/2} \tag{B.1}$$

that, substituting equation (3.43) into B.1, and evaluating it for $v = 1/\rho$, leads to

$$c(s,\,\rho) = \left\{(\gamma-1)\,\gamma\,e_0\left(\frac{\rho}{\rho_0}\right)^{\gamma-1} \exp\left[\frac{\gamma-1}{R}\,(s-s_0)\right]\right\}^{1/2} . \tag{B.2}$$

Differentiating this expression gives

$$\left(\frac{\partial c}{\partial \rho}\right)_s = \sqrt{\gamma\,(\gamma-1)\,e_0}\,\frac{\gamma-1}{2\,\rho_0}\left(\frac{\rho}{\rho_0}\right)^{\frac{\gamma-3}{2}}\exp\left[\frac{\gamma-1}{2\,R}\,(s-s_0)\right] \tag{B.3}$$

that, substituted in equation (3.71), gives

$$\Gamma^{PIG}(s,\,\rho) = 1 + \frac{\gamma-1}{2\,c\,(s,\,\rho)}\underbrace{\sqrt{(\gamma-1)\,\gamma\,e_0}\left(\frac{\rho}{\rho_0}\right)^{\frac{\gamma-1}{2}}\exp\left[\frac{\gamma-1}{2\,R}\,(s-s_0)\right]}_{c(s,\rho)} \tag{B.4}$$

where equation (B.2) was used to identify the speed of sound. It can thus be simplified, obtaining the final value of $\Gamma\,(s,\,\rho)$ for the PIG model

$$\Gamma^{PIG} = \frac{\gamma+1}{2} \tag{B.5}$$

that results therefore to be constant.

B.2 Polytropic van der Waals gas

In the case of polytropic van der Waals gas, the function $c\,(s,\,\rho)$ can be obtained by substituting equation (3.60) in expression (B.1) (that is general and valid for every gas model), obtaining thus

$$c\,(s,\,\rho) = \left\{(e_0 + a\,\rho_0)\left(\frac{\rho}{\rho_0}\,\frac{1-\rho_0\,b}{1-\rho\,b}\right)^{\delta}\exp\left[\frac{\delta}{R}\,(s-s_0)\right]\frac{\delta\,(\delta+1)}{(1-\rho\,b)^2} - 2\,a\,\rho\right\}^{1/2} \tag{B.6}$$

The differentiation of equation (B.6) leads to

$$\left(\frac{\partial c}{\partial \rho}\right)_s = \frac{1}{2}\left\{(e_0 + a\,\rho_0)\left(\frac{\rho}{\rho_0}\,\frac{1-\rho_0\,b}{1-\rho\,b}\right)^{\delta}\exp\left[\frac{\delta}{R}\,(s-s_0)\right]\frac{\delta\,(\delta+1)}{(1-\rho\,b)^2} - 2\,a\,\rho\right\}^{-1/2}$$
$$\left\{\frac{\delta\,(\delta+1)\,(\delta+2\,\rho\,b)}{\rho\,(1-\rho\,b)^3}\,(e_0 + a\,\rho_0)\left(\frac{\rho}{\rho_0}\,\frac{1-\rho_0\,b}{1-\rho\,b}\right)^{\delta}\exp\left[\frac{\delta}{R}\,(s-s_0)\right] - 2\,a\right\} \tag{B.7}$$

that substituted in equation (3.71) results in the polytropic van der Waals fundamental derivative $\Gamma^{PvdW}\,(s,\,\rho)$.

Appendix C

Estratto in lingua italiana

C.1 Introduzione

Attualmente sono presenti solutori fluidodinamici specializzati nella risoluzione di diversi tipi di problemi di interesse aerodinamico. I casi simulati in ambito ingegneristico sono spesso caratterizzati da comportamenti fortemente non lineari che giocano un ruolo centrale nell'evoluzione della corrente. Tali non linearità riguardano ad esempio onde d'urto e separazioni dello strato limite. Per poter cogliere questi fenomeni si rende necessaria l'implementazione di modelli raffinati, quali equazioni di Eulero e di Navier-Stokes (con modelli di turbolenza). Questi modelli sono tuttavia particolarmente costosi da un punto di vista computazionale. Nonostante la continua crescita della potenza di calcolo offerte dei moderni supercomputer non è ancora possibile completare in tempi ragionevoli simulazioni DNS nel caso di applicazioni pratiche. Questo comporta la necessità di modificare la formulazione con l'introduzione di modelli di turbolenza (RANS, LES) e soprattutto l'inevitabilità di costose verifiche sperimentali. Grazie alla sempre più crescente necessità di ottimizzazione delle geometrie aerodinamiche e al conseguente elevato numero di simulazioni richieste nelle prime fasi di progettazione, una formulazione utilizzata in passato ma caratterizzata da un' elevata efficienza computazionale è stato recentemente rivalutata: la formulazione a potenziale. Lo schema a potenziale permette di completare simulazioni su correnti comprimibili inviscide in tempi che possono risultare anche di diversi ordini di grandezza inferiori alle simulazioni con Eulero e Navier-Stokes. Inoltre la soluzione a potenziale soddisfa il livello di accuratezza necessario nelle prime fasi di progetto. Difatti tale formulazione è stata recentemente adottata al fine di valutare la stabilità aeroelastica di velivoli deformabili da Parrinnello [5].

L'elevato numero di simulazioni necessarie nelle prime fasi di progetto ha lo scopo di ridurre la regione dello spazio dei parametri all'interno della quale

ricercare la soluzione ottima, per permettere successivamente ai modelli più precisi di completare l'ottimizzazione. Diventa quindi essenziale la parallelizzazione del solutore a potenziale per supportare più efficientemente il carico di lavoro richiesto. Recentemente grazie al concetto di General Purpose Graphic Process Unit (GPGPU) è diventato possibile sfruttare la potenza di calcolo delle moderne schede video per accelerare la risoluzione di problemi numerici che manifestano una struttura ad elevato parallelismo dati. Se l'algoritmo in gioco si presta bene all'implementazione su GPU, è possibile ridurre i tempi di calcolo di decine o centinaia di volte rispetto alle comuni CPU. L'implementazione di un efficiente solutore aerodinamico su GPU richiede tuttavia l'uso di particolari strumenti di sviluppo e la conoscenza della struttura di base della sua architettura hardware. I primi approcci al calcolo su schede video (si veda [6]) si basavano su una mappatura del problema numerico in trasformazioni grafiche di pixel e vertici. Tale strategia risultava tuttavia particolarmente scomoda seppur computazionalmente efficiente. Successivamente, con il rilascio di CUDA nel 2007 e di OpenCL nel 2008, sono stati introdotti linguaggi appositamente sviluppati per facilitare la programmazione su GPU. Grazie a ciò sono stati sviluppati solutori per diversi tipi di problemi.

L'ottimizzazione delle geometrie aerodinamiche non può prescindere dall'accuratezza del modello di gas utilizzato nelle simulazioni. Il modello di gas ideale politropico fornisce risultati ottimi nelle comuni applicazioni aerodinamiche, come per le correnti attorno ad un ala. Tuttavia esistono casi in cui le condizioni termodinamiche sono tali da rendere necessario l'utilizzo di modelli di gas reali. Questo è particolarmente vero quando ci si trova ad operare in regime di gas denso, ovvero vicino al punto critico del gas. Si tratta delle condizioni in cui operano tipicamente i generatori di tipo Organic Rankine Cycle (ORC). In questo tipo di applicazioni, utilizzando gas di Bethe–Zel'dovich–Thompson (BZT) si rende possibile la presenza di inusuali fenomeni come urti di espansione e ventagli di compressione, che possono essere sfruttati per incrementare l'efficienza di tali macchine come ad esempio mostrato in [10]. Operando nella regione dei gas densi è infatti possibile sfruttare alcune peculiarità dei gas, come la riduzione del salto entropico attraverso gli urti, per ottenere elevate efficienze energetiche. La grandezza termodinamica protagonista di questa regione è la derivata fondamentale della gasdinamica Γ, il cui modulo e segno determinano le caratteristiche delle onde rarefattive e compressive. I già citati gas BZT sono infatti caratterizzati dall'esistenza di una regione di inversione in cui la Γ diventa minore di zero, rendendo quindi possibile l'esistenza di queste singolari onde.

Scopo di questo lavoro è lo sviluppo di un solutore esplicito a volumi finiti a potenziale, ExPReS, parallelizzato sia su CPU (usando OpenMP),

sia su GPU (usando OpenCL), che implementa sia un modello di gas ideale politropico, sia un modello di gas reale (van der Waals politropico). Lo scopo è quello di ottenere un software che sia un *proof-of-concept* per dimostrare i vantaggi dell'utilizzo delle GPU per la risoluzione di problemi aerodinamici. Inoltre un secondo scopo è verificare l'applicabilità di un'efficiente formulazione a potenziale anche in regime di gas denso. Particolare rilevanza viene data anche all'uso di software libero.

C.2 CFD a potenziale

La formulazione a potenziale utilizzata nel presente lavoro consente la simulazione di correnti aerodinamiche comprimibili, inviscide ed isoentropiche.

L'approccio a potenziale si basa sull'espressione del campo di velocità $\boldsymbol{u}$ come

$$\boldsymbol{u} = \boldsymbol{\nabla}\phi, \tag{C.1}$$

dove ϕ è il potenziale cinetico.

Ciò porta a due tipi di indeterminazione. Innanzitutto dato che nelle equazioni il potenziale compare sempre sotto segno di derivata. Dunque se $\tilde{\phi}$ è soluzione, anche $\tilde{\phi} + C$ lo, dove C è una costante arbitraria. Dato però che ciò che importa è la conoscenza del campo di velocità, questa indeterminazione non risulta problematica. Si ha inoltre un'indeterminazione sul valore di circolazione attorno ad ogni corpo portante presente in un dominio molteplicemente connesso. Questo secondo tipo di indeterminazione è risolta grazie all'introduzione della scia a valle dei corpi portanti e l'applicazione su di essa della condizione di Kutta.

L'ipotesi di isoentropicità

$$s\left(\boldsymbol{r},\, t\right) = s_\infty \tag{C.2}$$

è a rigore applicabile in assenza di urti. Tuttavia per un'ala o un profilo in regime transonico, quando gli urti sono deboli, è ancora possibile utilizzare tale ipotesi in maniera approssimata. Infatti, come dimostrato in [22], in presenza di deboli discontinuità di pressione attraverso un urto, il conseguente salto di entropia risulta trascurabile.

Le equazioni utilizzate nella formulazione a potenziale sono la conservazione della massa nella forma integrale

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_V \rho = -\oint_{\partial V} \rho\, \boldsymbol{\nabla}\phi \cdot \hat{\boldsymbol{n}}, \tag{C.3}$$

e il teorema di Bernoulli

$$\frac{\partial \phi}{\partial t} + \frac{1}{2}\boldsymbol{\nabla}\phi \cdot \boldsymbol{\nabla}\phi + h_{is}(\rho) = \frac{1}{2}V_\infty^2 + h_\infty, \tag{C.4}$$

ricavato dell'equazione di conservazione della quantità di moto espressa in forma differenziale.

Il modello termodinamico del gas è tenuto in considerazione attraverso la funzione entalpia per trasformazioni isoentropiche $h_{is}(\rho)$.

In questo lavoro due formulazioni sono state confrontate: la formulazione bicampo, dove le incognite sono rappresentate dalla densità ρ e dal potenziale ϕ, e la formulazione monocampo, dove l'unica incognita è rappresentata dal potenziale ϕ. La formulazione bicampo fa uso contemporaneamente delle equazioni di conservazione della massa e del teorema di Bernoulli, mentre nella formulazione monocampo il teorema di Bernoulli viene utilizzato per esprimere la densità in funzione del potenziale in modo da sostiturne l'espressione nella legge di conservazione della massa

Per risolvere l'indeterminazione sulla circolazione, si applica la condizione di Kutta imponendo la continuità del campo di densità attraverso la scia, inquanto questa non costituisce una superficie di discontinuità per le grandezze termodinamiche ma solo per il potenziale ϕ. La condizione di Kutta è inoltre equivalente all'assenza di salti di velocità normale alla scia.

Al fine di stabilizzare la simulazione, è necessario introdurre della dissipazione numerica. Ciò è fattibile ricorrendo alla tecnica dell'upwinding, necessaria inoltre per tenere in considerazione la causalità spaziale nella corrente, specialmente nelle regioni supersoniche.

Particolare attenzione è posta sul confronto tra la formulazione monocampo e bicampo. Il principale vantaggio della formulazione moncampo consiste nel dimezzamento del numero di incognite rispetto alla formulazione bicampo, caratteristica che lo pone in vantaggio nell'implementazione di un solutore implicito. Tuttavia la formulazione bicampo risulta migliore sotto moltri altri aspetti, come un minor numero di calcoli non lineari per l'assemblaggio di matrici e vettori nel caso implicito. Il bicampo consente inoltre una più agevole formulazione ALE ed infine la possibilità di implementare modelli di gas diversi dal gas ideale politropico.

In questo lavoro è stato implementato il solutore ExPReS. E' stato utilizzato uno schema a potenziale bicampo esplicito capace di simulare il movimento di corpi portanti tramite formulazione ALE o condizioni al contorno di traspirazione.

C.3 Termodinamica

In questo lavoro è stato utilizzato l'approccio assiomatico alla termodinamica proposto da Galgani e Scotti [25]. La particolarità di questo approccio consiste nel fatto che la grandezza principale, l'entropia, espressa in funzione dell'energia interna e volume specifico,

$$s = s(e,\, v) \tag{C.5}$$

deve soddisfare tre assiomi: monotonia rispetto all'energia, omogeneità e superadditività rispetto ai suoi argomenti.

Come accennato in C.2 nella formulazione a potenziale bicampo il modello termodinamico di gas è preso in considerazione attraverso l'espressione dell'entalpia per il caso di trasformazione isoentropica $h_{is}(\rho)$, espressa in funzione della densità. Nota questa funzione per un certo modello di gas, è possibile implementare tale modello nel solutore. In ExPReS sono attualmente presenti i modelli di gas ideale politropico e di van der Waals politropico. Mentre il primo dei due è particolarmente adatto allo studio dei comuni problemi di interesse aerodinamico, il secondo permette di cogliere i fenomeni tipici dei gas densi, quali urti di espansione, onde miste, ventagli compressivi e urti con un debole salto di entropia. La particolarità del modello di gas di van der Waal consiste nella sua capacità di modellare le forze di repulsione a breve distanza e le forze di attrazione a lunga distanza, grazie all'uso dei coefficienti a e b

$$\left(P + a\,\rho^2\right)(1 - b\,\rho) = \rho\, R\, T \tag{C.6}$$

Il modello di gas di van der Waals tuttavia sovrastima la dimensione della regione in cui si manifestano tali fenomeni.

La grandezza termodinamica protagonista del comportamento del gas nella regione dei gas densi è la derivata fondamentale della gasdinamica Γ. Sulla base del segno e del modulo assunto da tale grandezza si possono distinguere tre tipologie di gas:

- $\Gamma > 1$ ovunque: fluido LMC (Low Molecular Complexity)
- $\Gamma > 0$ ed esiste una regione termodinamica dove$0 < \Gamma < 1$: fluido HMC (High Molecular Complexity)
- $\Gamma < 0$ in alcune regioni termodinamiche: fluido BZT (Bethe–Zel'dovich–Thompson)

La derivata fondamentale della gasdinamica è definita come

$$\Gamma\left(s,\, v\right) \triangleq \frac{v^3}{2\, c^2\left(s,\, v\right)} \left(\frac{\partial^2 P}{\partial v^2}\right)_s \tag{C.7}$$

ciò significa che i fluidi BZT ammetttono una regione termodinamica dove la curvatura delle isoentropiche nel diagramma P-v cambia segno. Le conseguenze di questo comportamento diventano evidenti grazie alla formula che lega la variazione di entropia alla variazione di volume specifico [22]

$$\Delta s = -\left(\frac{\partial^2 P}{\partial v^2}\right)_s \frac{\left[\Delta v\right]^3}{12\, T} + \mathcal{O}\left(\left[\Delta v\right]^4\right) \tag{C.8}$$

Infatti, durante un'espansione si ha $\Delta v > 0$ e quindi per $\Gamma < 0$ si ottiene $\Delta s > 0$. Sono quindi ammessi urti di espansione e ventagli di compressione. Un'altra conseguenza importante è data dal fatto che quando $\Gamma \approx 0$, è possibile dimostrare [29] che $\Gamma = \mathcal{O}\left(\left[\Delta v\right]^4\right)$. Ciò significa che in questa regione termodinamica gli urti sono associati a un debole salto di entropia e si può quindi sfruttare tale aspetto per incrementare l'efficienza energetica dei dispositivi che operano in queste condizioni. Dato che il modello di gas ideale politropico fornisce un valore di Γ costante, questo non è utilizzabile per lo studio di correnti in condizioni di gas denso. Il modello di van der Waal politropico fornisce invece una Γ variabile in funzione dello stato termodinamico ed è quindi in grado di modellare la regione termodinamica tipica dei gas BZT. Parte di questa tesi è dedicata proprio all'indagine della regione dei gas densi.

C.4 Calcolo parallelo

Uno degli obiettivi principali di questo lavoro consiste nel valutare lo speedup offerto dalle moderne GPU per accelerare le simulazioni aerodinamiche. Il solutore ExPReS sviluppato in questo lavoro è stato scritto in C/C++ e parallelizzato sia su CPU che su GPU. L'architettura delle moderne GPU prevede sostanzialmente un processore dotato di centinaia o migliaia di cores in grado sostanzialmente di effettuare la stessa operazione contemporaneamente su dati diversi. Grazie a questa peculiarità si possono ottenere grandi vantaggi quando il problema in gioco può essere risolto attraverso un algoritmo che esibisce un elevato parallelismo dati. Grazie alla sua architettura altamente parallela, la GPU è in grado di accelerare, a seconda del problema in gioco, l'esecuzione delle simulazioni anche nell'ordine del centinaia di volte rispetto ai tempi tipici per una CPU. All'interno di una GPU è presente una gerarchia di memoria, dove particolare importanza assume la memoria globale, che deve essere utilizzata per le comunicazioni tra CPU e GPU. CPU e GPU

sono collegate attraverso il bus PCI-Express. Essendo tale bus molto più lento della comunicazione tra il processore della GPU e la sua memoria, la strategia utilizzata quando un problema numerico deve essere risolto su GPU consiste nel caricare tutti gli input nella memoria globale della GPU, effettuare i calcoli e solo alla fine della simulazione trasferire i risultati dalla GPU alla CPU. Tale strategia è ovviamente stata seguita anche nello sviluppo della versione per GPU di ExPReS.

La versione CPU del solutore è stata parallelizzata utilizzando OpenMP in modo da poter sfruttare l'architettura SMP (Symmetric Multi Processing) attraverso la tecnica del multithreading, distribuendo il carico di lavoro tra i diversi core della CPU. La versione GPU del solutore è stata invece sviluppata utilizzando OpenCL. Attualmente gli standard più diffusi per il calcolo su GPU sono rappresentati da CUDA e OpenCL. La scelta di OpenCL è stata guidata dalla sua compatibilità con un maggiorn numero di dispositivi. OpenCL mette a disposizione un API (Application Programming Interface) e un linguaggio di programmazione, OpenCL C. L'API fornisce al programmatore le funzioni e i tipi di dati necessari per la comunicazione tra CPU e GPU mentre il linguaggio, un derivato del C99, permette la scrittura delle funzioni, chiamate kernel, da eseguire sulla GPU. La programmazione del solutore aerodinamico su GPU richiede la scrittura di due tipi di codici sorgente: quelli che verranno compilati ed eseguiti su CPU e quelli che verranno compilati in runtime ed eseguiti su GPU. Infatti il codice sorgente che contiene i kernel viene compilato durante l'esecuzione del programma su CPU ed eseguito successivamente su GPU.

Come mostrato in questo lavoro, la programmazione su GPU porta a notevoli vantaggi in termini di tempi di calcolo, alle spese di una maggiore difficoltà durante la fase iniziale di scrittura del codice sorgente. Infatti l'architettura stessa delle GPU comporta alcune limitazioni e criticità, come quelle della memory consistency e della branch divergence. Inoltre la programmazione su scheda video non può prescindere da una conoscenza di base dell'architettura hardware tipica di un processore grafico.

C.5 Discretizzazione numerica

Il solutore ExPReS sviluppato in questo lavoro implementa la formulazione a potenziale comprimibile bicampo grazie ad una discretizzazione spaziale a volumi finiti a celle centrate. L'avanzamento in tempo viene effettuato grazie ad uno schema esplicito sfalsato. La scelta di questi metodi di discretizzazione è strettamente legata alla necessità di schemi che possano essere eseguiti in maniera efficiente su hardware parallelo, in particolar modo su GPU. Innanz-

itutto la scelta di un metodo esplicito di integrazione permette l'avanzamento in tempo senza la necessità di risolvere sistemi lineari: ogni cella del dominio può avanzare indipendentemente dalle altre grazie alla sola conoscenza della soluzione nel dominio all'istante precedente. Ciò permette di distribuire con facilità ed efficienza l'aggiornamento delle celle del dominio tra i cores di una GPU (o di una CPU). Per quanto riguarda la scelta delle celle centrate, questa è strettamente legata ad alcune limitazioni dell'architettura della GPU che devono essere rispettate per ottenere un'esecuzione efficiente e corretta del programma. Sostanzialmente dato che la massima efficienza computazionale su GPU si ottiene quando tutti i cores svolgono lo stessa serie di istruzioni, evitando strutture condizionali (rami di if), durante l'accumulo dei flussi su una cella, la formulazione a nodi centrati si troverebbe in svantaggio rispetto a quella a celle centrate a causa del numero variabile di interfacce relative ai diversi nodi. I problemi che si incontrerebbero sono principalmente legati alla memory consistency e alla branch divergence.

La formulazione a potenziale bicampo discretizzata si presenta nella forma

$$\begin{cases} \dfrac{R_i^{n+1} - R_i^n}{\Delta t} = -\dfrac{1}{V_i} \displaystyle\sum_{j=1}^{N_f} \bar{\rho}_{i,j}^{\,n} \; \boldsymbol{G}_{\Phi^n}^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j} \; \Delta L_{i,j} \\ \\ \dfrac{\Phi_i^{n+1} - \Phi_i^n}{\Delta t} = \dfrac{1}{2} V_\infty^2 - \dfrac{1}{2} \boldsymbol{G}_{\Phi^n}^{C_i} \cdot \boldsymbol{G}_{\Phi^n}^{C_i} + h_\infty - h_{is}\left(R_i^{n+1}\right) \end{cases} \tag{C.9}$$

dove R_i^n è l'approssimazione della densità sulla cella i all'istante n, V_i il volume della cella i–esima, $\bar{\rho}_{i,j}^{\,n}$ è la densità usata per il calcolo del flusso (quindi upwindata) sull'interfaccia $i-j$, $\hat{\boldsymbol{n}}_{i,j}$ è la normale all'interfaccia $i-j$ e $\Delta L_{i,j}$ è la dimensione dell'interfaccia. Il pedice ∞ indica le condizioni asintotiche. I simboli $\boldsymbol{G}_{\Phi^n}^{C_i}$ e $\boldsymbol{G}_{\Phi^n}^{F_{i,j}}$ indicano la ricostruzione numerica dei gradienti rispettivamente sulla cella e sull'interfaccia. Per quanto riguarda i gradienti sulle celle sono stati implementati due metodi. Il primo si basa sul teorema di Green-Gauss

$$\boldsymbol{\nabla} u \,|_{V_i} \approx \boldsymbol{G}_u^{C_i} = \frac{1}{|V_i|} \sum_{j=1}^{N_f} \bar{U}_{i,j} \, \hat{\boldsymbol{n}}_{i,j} \, \Delta L_{i,j} \tag{C.10}$$

mentre il secondo usa una interpolazione ai minimi quadrati, dove il gradiente sulla cella viene calcolato facendo uso di uno stencil costituito da alcune celle adiacenti, come rappresentato in figure C.1. Per quanto riguarda il gradiente sulle interfacce tra le celle, questo può essere calcolato noti i gradienti di centro cella delle due celle che competono all'interfaccia.

simple stencil augmented stencil

Figura C.1: Least Square C-Gradient stencils

Diverse strategie possono essere adottate per calcolare il valore di densità con cui calcolare il flusso di massa. Per ragioni di stabilità numerica e causalità spaziale in ExPReS sono state implementati schemi di upwinding, dove il valore di interfaccia viene modificato risalendo nella direzione della velocità locale. Una tecnica di upwind consiste ad esempio nell'assegnare all'interfaccia il valore della densità della cella upwind, oppure utilizzare una funzione che tenga conto in piccola parte anche del valore della cella downwind. Un'altra strategia consiste nel calcolare il gradiente di interfaccia come media pesata tra il valore delle sue celle e poi sottrarre a questo una quantità in modo da risalire nella direzione del vento locale

$$\bar{\rho} = \rho + \boldsymbol{\nabla}\rho \cdot \boldsymbol{l} \tag{C.11}$$

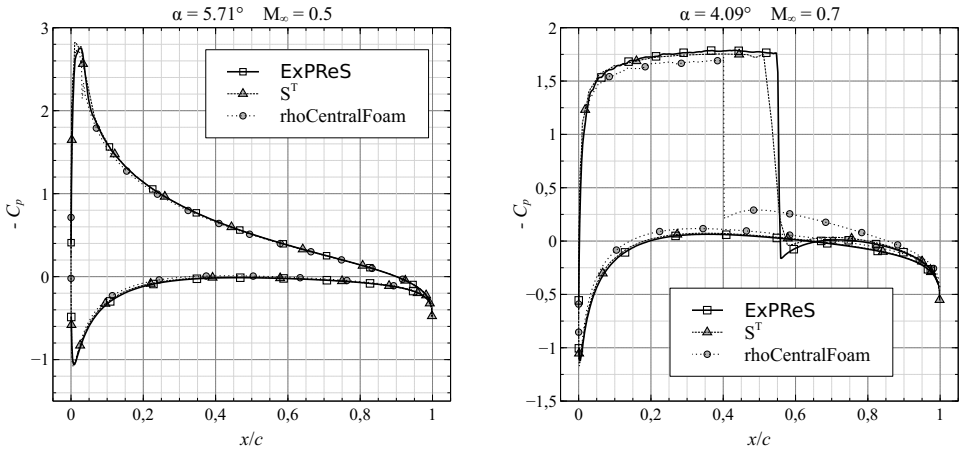
Con lo scopo di aumentare la stabilità numerica del metodo, è stato implementato uno schema di avanzamento temporale sfalsato. Ciò significa che l'equazione di conservazione della massa e il teorema di Bernoulli sono risolti con un certo ordine, sfruttando una soluzione intermedia. Infatti viene prima risolta l'equazione di conservazione della massa per calcolare il nuovo valore di densità. Con questo nuovo valore di densità viene poi risolta l'equazione di Bernoulli in modo da aggiornare il valore del potenziale

$$R_i^{n+1} = R_i^n - \frac{\Delta t}{V_i} \sum_{j=1}^{N_f} \bar{\rho}_{i,j}^{\,n} \, \boldsymbol{G}_{\Phi^n}^{F_{i,j}} \cdot \hat{\boldsymbol{n}}_{i,j} \, \Delta L_{i,j}$$

$$\Phi_i^{n+1} = \Phi_i^n + \Delta t \left(\frac{1}{2} V_\infty^2 - \frac{1}{2} \boldsymbol{G}_{\Phi^n}^{C_i} \cdot \boldsymbol{G}_{\Phi^n}^{C_i} + h_\infty - h_{is}\left(R_i^{n+1}\right) \right)$$

Questa strategia permette di aumentare la stabilità del metodo rispetto al semplice schema di Eulero esplicito.

Nel solutore ExPReS sono stati inoltre implementate strategie di accelerazione della convergenza. La prima tecnica è quella del local time stepping,



(a) NACA0012 subsonico (b) NACA0012 transonico

Figura C.2: Soluzioni in regime di gas denso

grazie al quale ogni cella avanza con il massimo valore di Δt locale permesso dal vincolo di stabilità (CFL). La seconda tecnica è quella del rilassamento, dove la soluzione all'istante corrente viene calcolata come media pesata tra la soluzione all'istante precedente e quella aggiornata con l'integrazione temporale sfalsata. Anche nel caso dello schema con rilassamento è stata analizzata la stabilità numerica.

Per quanto riguarda le condizioni al contorno, sul corpo immerso nella corrente è imposta la condizione di non penetrabilità, mentre agli estremi del dominio, a grande distanza dal corpo, sono imposte le condizioni asintotiche, eventualmente upwindate.

In ExPReS inoltre è possibile simulare il movimento dei corpi, in modo da effettuare simulazioni instazionarie. Questo può essere effettuato grazie alle condizioni al contorno di traspirazione sul bordo del corpo oppure grazie alla formulazione ALE, dove la griglia viene deformata sulla base di una legge temporale con velocità di griglia a divergenza nulla.

C.6 Risultati e conclusioni

Il solutore è stato validato utilizzando risultati reperibili in letteratura per correnti subsoniche e transoniche attorno al profilo NACA 0012 [59] oppure utilizzando solutori a potenziale [5] e di equazioni di Eulero [57].

Le prime soluzioni calcolate riguardano regimi di gas ideale in condizioni subsoniche e transoniche. Dai risultati si nota un buon accordo tra la soluzione fornita da ExPReS e quella fornita dagli altri solutori nel caso di correnti

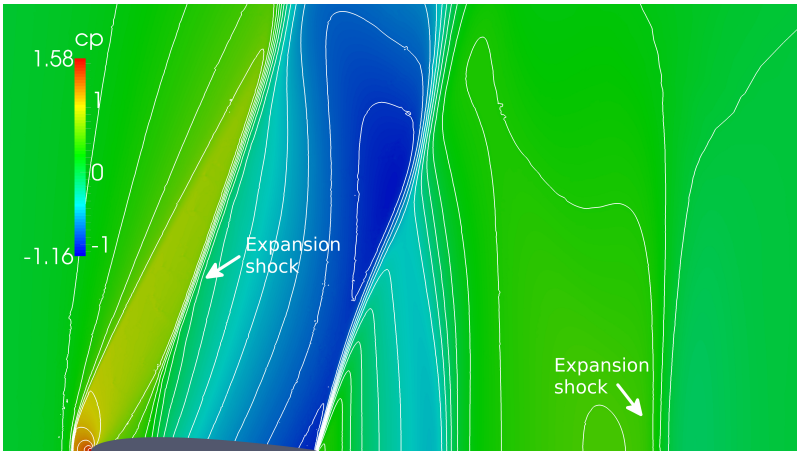


Figura C.3: NACA0012 in condizioni di gas denso

subsoniche e correnti transoniche senza incidenza. Nel caso di correnti transoniche con incidenza il solutore ExPReS è risultato in buon accordo con l'altro solutore a potenziale S^T mentre, come previsto, la posizione dell'urto identificata dai solutori di Eulero è risultata differente.

Successivamente il solutore è stato testato con il modello di gas reale di van der Waals politropico utilizzando correnti in cui il gas si trovava nelle tipiche condizioni termodinamiche dei gas densi. In particolare sono stati ricostruiti alcuni casi presentati in [59] dove sono state messe in evidenza fenomeni tipici dei gas densi. È stato possibile verificare come lo stesso numero di Mach asintotico che in condizioni di gas ideal porta ad una corrente transonica, in condizioni di gas denso porti invece ad una corrente senza urti. Successivamente lo studio si è spostato sulla ricerca degli urti di espansione, oggetto di attuale interesse in ambito di ricerca. Il solutore ExPReS si è dimostrato in grado di ricostruire l'urto di espansione previsto sul profilo a incidenza nulla, come mostrato in figure C.3.

Tuttavia alcune discrepanze tra i risultati sono state rilevate dopo il bordo d'uscita del profilo, probabilmente a causa dell'ipotesi si isentropicità che non viene più soddisfatta attraverso un forte urto.

Successivamente sono stati condotti test per correnti portanti in condizioni di gas denso in cui è stato riscontrato un discreto accordo con i risultati calcolati risolvendo le equazioni di Eulero.

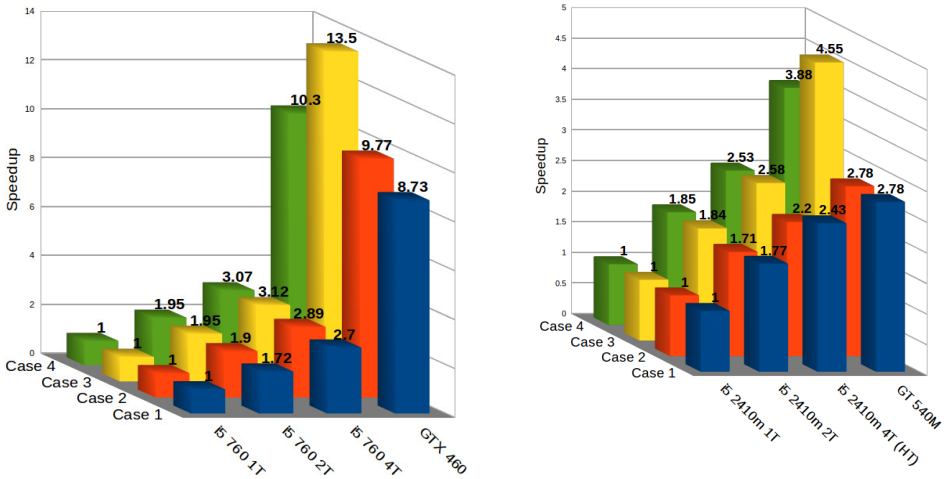
Un'altra funzionalità di ExPReS implementata in questo lavoro è la possibilità di simulare correnti instazionarie attorno a profili oscillanti, grazie alle condizioni al contorno di traspirazione e alla formulazione ALE. I risultati per il caso di gas ideale politropico sono stati confrontati con quelli forniti dal software S^T.

Le prestazioni di alcuni metodi di accelerazione della convergenza alla soluzione stazionaria sono state analizzate mediante uno studio di convergenza del residuo. Il metodo di local time stepping non è risultato benefico nelle prove che sono state effettuate. Il sovrarilassamento invece ha consentito una diminuzione del numero di iterazioni per raggiungere lo stato stazionario, sebbene sia soggetto ad un limite di stabilità (numero di CFL) più stringente della tecnica standard di avanzamento in tempo.

Il solutore ExPReS è stato modificato per includere la possibilità di calcolare correnti tridimensionali. Sono stati riscontrati tuttavia numerosi problemi di instabilità numerica, probabilmente legati agli schemi di ricostruzione del gradiente. Questa considerazione nasce dal fatto che, se si assegna un campo di trasporto (disattivando quindi il teorema di Bernoulli), è possibile simulare senza problemi il trasporto di un fronte d'onda in un dominio vuoto. Ciò probabilmente rafforza la possibilità che l'instabilità sia legata alla ricostruzione dei gradienti piuttosto che alla formulazione bicampo, dato che quest'ultima è già stata ampiamente utilizzata per il calcolo di correnti tridimensionali utilizzando uno schema a nodi centrati che generalmente garantisce una maggiore precisione nel calcolo dei gradienti.

I benchmark condotti per determinare lo speedup tra CPU e GPU sono stati effettuati su hardware dello stessa fascia di prezzo (intorno ai 100-200 euro) e dello stesso periodo di produzione (anno 2011). Tale scelta è stata presa in modo tale da evitare confronti tra CPU di fascia bassa e GPU di fascia alta o confronti tra architetture troppo diverse, condizioni che fornirebbero risultati poco onesti. I benchmark sono stati condotti su alcune griglie di dimensioni differenti, in modo da valutare l'andamento dello speedup al crescere della dimensione del problema. È stato analizzato lo speedup su quattro griglie differenti e con delle configurazioni desktop e notebook di fascia media. Uno dei risultati è riportato in figure C.4.

Si nota come le prestazioni del solutore scalino all'aumentare del numero di cores della CPU in maniera quasi lineare grazie alla parallelizzazione con OpenMP. Soprattutto è evidente il divario prestazionale tra GPU e CPU nella configurazione desktop. Il fatto che lo speedup vari in funzione della dimensione della griglia è molto probabilmente associato all'effetto combinato tra i colli di bottiglia nella comunicazione tra CPU e GPU, particolarmente importanti quando la mesh ha pochi elementi, e la dimensione della cache della GPU, che potrebbe venir facilmente saturata con griglie con elevato numero di celle.



(a) Desktop (b) Notebook

Figura C.4: GPU speedup. 2011 mid range hardware

Bibliography

[1] J. D. Cole and E. M. Murman. Calculation of plane steady transonic flows. *AIAA Journal*, 9(1):114–121, 1971. 9

[2] A. Jameson. Transonic potential flow calculations using conservation form. In *Proc. of AIAA 2nd Computational Fluid Dynamic Conference, Hartford, Conn*, pages 148–155, 1975. 9

[3] A. Jameson and D. A. Caughley. Transonic potential flow calculations using conservative form. In *In Proceedings AIAA Third Computational Fluid Dynamics Conference. Albuquerque*, 1977. 9

[4] R. E. Neel. *Advances in Computational Fluid Dynamics: Turbulent Separated Flows and Transonic Potential Flows*. PhD thesis, Virginia Polytechnic Institute and State University, August 1997. 9

[5] A. Parrinello. *Independent Fields Full Potential Formulation for Aeroelastic Analyses*. PhD thesis, Politecnico di Milano, March 2012. 9, 92, 109, 113, 115, 148, C-1, C-10

[6] T. R. Hagen, J. M. Hjelmervik, K.-A. Lie, J. R. Natvig, and M. O. Henriksen. Visual simulation of shallow-water waves. *Simulation Modelling Practice and Theory*, 13(8):716–726, 2005. 10, 64, C-2

[7] E. Elsen, P. LeGresley, and E. Darve. Large calculation of the flow over a hypersonic vehicle using a gpu. *Journal of Computational Physics*, 227(24):10148–10161, 2008. 11

[8] T. Brandvik and G. Pullan. Acceleration of a two-dimensional euler flow solver using commodity graphics hardware. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 221(12):1745–1748, 2007. 11

[9] A. Corrigan, F. F. Camelli, R. Löhner, and J. Wallin. Running unstructured grid-based cfd solvers on modern graphics hardware. *International Journal for Numerical Methods in Fluids*, 66(2):221–229, 2011. 11

[10] A. Guardone, A. Spinelli, and V. Dossena. Influence of molecular complexity on nozzle design for an organic vapor wind tunnel. *Journal of engineering for gas turbines and power*, 135(4), 2013. 11, C-2

[11] P. Colonna, J. Harinck, S. Rebay, and A. Guardone. Real-gas effects in organic rankine cycle turbine nozzles. *Journal of Propulsion and Power*, 24(2):282–294, 2008. 11

[12] M. S. Cramer and G. M. Tarkenton. Transonic flows of bethe-zel'dovich-thompson fluids. *Journal of Fluid Mechanics*, 240:197–228, 1992. 11

[13] *GCC User's Guide*, 2013. http://gcc.gnu.org/onlinedocs/. 13

[14] OpenMP Architecure Review Board. *OpenMP Application Program Interface*, 2013. http://openmp.org/wp/. 13, 61

[15] The Khronos Group Inc. *The OpenCL Specification*, 2012. 13, 72

[16] *GDB User's Guide*, 2013. https://www.gnu.org/software/gdb/documentation/. 13

[17] *gprof User's Guide*, 2013. http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html. 13

[18] *gDEBugger User's Guide*, 2013. http://www.gremedy.com/webhelp/. 13

[19] *ParaView User's Guide*, 2013. http://www.paraview.org/paraview/help/documentation.html. 13, 112

[20] *Veusz User's Guide*, 2013. http://home.gna.org/veusz/. 13

[21] *gmsh User's Guide*, 2013. http://geuz.org/gmsh/. 13

[22] L. D. Landau and E. M. Lifshitz. *Fluid Mechanics, Course of Theoretical Physics*, volume 6. Pergamon Press, 3 revised english edition, 1959. 16, 47, C-3, C-6

[23] B. P. Brown and B. M. Argrow. Application of Bethe-Zel'dovich-Thompson Fluids in Organic Rankine Cycle engines. *Journal of Propulsion and Power*, 16(6):1118–1124, 2000. 33

[24] J. Harinck, A. Guardone, and P. Colonna. The influence of molecular complexity on expanding flows of ideal and dense gases. *Physics of Fluids*, 21, 2009. 33

[25] L. Galgani and A. Scotti. On subadditivity and convexity properties of thermodynamic functions. *Pure and Applied Chemistry*, 22:229–236, 1970. 33, C-5

[26] H. B. Callen. *Thermodynamics and an introduction to thermostatistics*. John Wiley & sons, 2nd edition, 1985. 33, 40

[27] L. P. Quartapelle and F. Auteri. *Fluidodinamica Incomprimibile*. Casa Editrice Ambrosiana, 1st edition, 2013. 33, 36, 37

[28] P. A. Thompson. A fundamental derivative in gasdynamics. *The Physics of Fluids*, 14(9):1843–1849, September 1971. 45

[29] M. S. Cramer and A. Kluwick. On the propagation of waves exhibiting both positive and negative nonlinearity. *Journal of Fluid Mechanics*, 142(1):9–37, 1984. 47, C-6

[30] P. A. Thompson and K. C. Lambrakis. Negative shock waves. *J. Fluid Mech*, 60(1):187–208, 1973. 48

[31] http://computing.llnl.gov/. 51, 52, 53, 54, 58, 59, 61, 62, 64

[32] M. J. Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, 100(9):948–960, 1972. 52

[33] NVIDIA Corporation. *OpenCL Programming Guide for the CUDA Architecture*, 2012. 54, 72, 76

[34] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967. 55

[35] http://neilkemp.us/src/sse_tutorial/sse_tutorial.html. 57

[36] http://sci.tuomastonteri.fi/programming/sse. 63

[37] NVIDIA Corporation. *NVIDIA OpenCL JumpStart Guide*, 2011. 65

[38] A. Munshi, B. R. Gaster, T. G. Mattson, J. Fung, and D. Ginsburg. *OpenCL Programming Guide*. Addison-Wesley Professional, 2011. 72

[39] R. Tsuchiyama, T. Nakamura, T. Iizuka, A. Asahara, and S. Miki. *The OpenCL Programming Book*. Fixstars Corporation, 2010. 72

[40] AMD, Inc. *AMD Accelerated Parallel Processing OpenCL*, 2012. 72

[41] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa. *Heterogeneous computing with OpenCL*. Morgan Kaufmann, 2011. 72

[42] M. Scarpino. *OpenCL in action*. Manning Publications, 2011. 72

[43] NVIDIA Corporation. *Cuda C Programming Guide*, 2012. 76

[44] NVIDIA Corporation. *NVIDIA's Next Generation CUDA Compute Architecture: Fermi*. 76

[45] NVIDIA Corporation. *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110*. 76

[46] V. Selmin. The node-centred finite volume approach: bridge between finite differences and finite elements. *Computer Methods in Applied Mechanics and Engineering*, 102(1):107–138, 19932. 80

[47] J. Blazek. *Computational Fluid Dynamics*. Elsevier, 2001. 80, 97

[48] I. C. Kampolis, X. S. Trompoukis, V. G. Asouti, and K. C. Giannakoglou. Cfd-based analysis and two-level aerodynamic optimization on graphics processing units. *Computer Methods in Applied Mechanics and Engineering*, 199(9):712–722, 2010. 84

[49] B. Diskin, J. L. Thomas, E. J. Nielsen, H. Nishikawa, and J. A. White. Comparison of node-centered and cell-centered unstructured finite-volume discretizations. part i: viscous fluxes. *AIAA paper*, 2009. 86

[50] A. Hasselbacher. A WENO Reconstruction Algorithm for Unstructured Grids Based on Explicit Stencil Construction. In *43th AIAA Aerospace Sciences Meeting and Exhibit*, January 2005. 88

[51] D. G. Holmes and S. D. Connell. *Solution of the 2D Navier-Stokes equations on unstructured adaptive grids*. American Institute of Aeronautics and Astronautics, 1989. 88

[52] N. Frink. Assessment of an unstructured-grid method for predicting 3-d turbulent viscous flows. *AIAA paper*, 1996. 88

[53] A. Haselbacher. On Constrained Reconstruction Operators. In *44th AIAA Aerospace Sciences Meeting and Exhibit*, January 2006. 88

[54] P. Masarati, M. Lanz, and P. Mantegazza. Multistep integration of ordinary, stiff and differential-agebraic problemsn for multibody dynamics applications. In *XVI Congresso Nazionale AIDAA*, September 2001. 94

[55] A. Jameson, W. Schmidt, and E. Turkel. Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes. *AIAA paper*, 1981. 97

[56] A. Quarteroni. *Modellistica Numerica per Problemi Differenziali*. Springer, 4th edition, 2008. 98

[57] http://www.openfoam.org/. 109, 113, C-10

[58] *gnuplot User's Guide*, 2013. http://www.gnuplot.info/documentation.html. 112

[59] P. Cinnella and P. M. Congedo. Aerodynamics Performance of Transonic Bethe-Zel'dovich-Thompson Flows past an Airfoil. *AIAA Journal*, 43(2):370–378, February 2005. 114, 115, 116, 122, 123, C-10, C-11

[60] http://www.nvidia.it/object/gpu-computing-applications-it.html. 142, 143, 144

[61] L. P. Quartapelle and F. Auteri. *Fluidodinamica Comprimibile*. Casa Editrice Ambrosiana, 1st edition, 2013. A-6