A High-order Method for Computational Hypersonic Aerothermodynamics

Vincenzo Fico

Department of Mechanical Engineering

University of Strathclyde

A thesis submitted for the degree of

PhilosophiæDoctor

October 2011

Declaration

Abstract

In this thesis we study and develop a high-order high-resolution numerical method for hypersonic flow simulation in the framework of high performance computing. A compact-Total Variation Diminishing (TVD) method for the Euler equations is selected as a solution method to study, extend and improve. A method to solve the Navier-Stokes equations is proposed, combining this compact-TVD method with a Kinetic Splitting technique for the Navier-Stokes flux. The resulting method has improved shock capturing properties and a computational advantage over operator splitting conventional methods. The stability and accuracy of the method are demonstrated through a wide range of test cases, including typical hypersonic flows.

We consider how this method may be used in the context of high-performance computing, exploring the three main paradigms of modern supercomputing: parallelisation through message passing, parallelisation through memory sharing and acceleration through Graphic Processing Units. A novel algorithm to parallelise our compact-TVD method using message passing is presented. The algorithm is based on structured-block partitioning and is an improvement over previous algorithms that employ the same approach. Our multi-block algorithm is proven to be suitable for massively parallel computing: we study its parallel performance and find that its parallel efficiency is about 90% when running a simulation of 1 million cells using 1 thousand Central Processing Unit (CPU) cores. We then present a strategy to parallelise our compact-TVD method in the complex computing environment of a Graphic Processing Unit (GPU). Our strategy consists of two steps: 1) algorithm break-down into elementary tasks; 2) adoption of a task-dependent domain partitioning. Our task-dependent partitioning strategy is shown to be very effective: the speed-up is about $29\times$ with respect to one CPU core when running a simulation of 3 million nodes.

To Andrea, who showed me the light ...

Acknowledgements

I would like to thank my supervisors Prof. David Emerson and Prof. Jason Reese for the careful proof-reading of this thesis. I thank Prof. Dave Emerson for his constant support, and for being always available for discussion and help during these three years. Thanks to Prof. Jason Reese for trying to transmit me his great passion for research. Thanks to Dr. Charles Moulinec for being a good friend and sharing interesting perspectives on fluid mechanics, and life in general. I also thank Dr. Ming Jiang for introducing me to his philosophical view on the charming world of computer science. I finally thank the staff at STFC Daresbury Laboratory, for providing the most friendly work environment one could ask for.

Contents

List of Figures

List of Tables

1

Introduction

Vehicles accessing space, such as launchers, shuttles and capsules, fly at very high speeds, typically much higher than the speed of sound in the surrounding medium. This condition is effectively expressed using the flight Mach number, M_∞, defined as the ratio of the vehicle speed, V_∞, to the local speed of sound, a_∞. Space vehicles often fly at Mach numbers as high as 25, and hence are in the so-called *hypersonic* flow regime, conventionally defined by $M_\infty > 5$ (1). Hypersonic flight conditions are design conditions for these vehicles, because they are associated with very high thermal and mechanical loads (Fig. 1.1). Simulation of these flows using Computational Fluid Dynamics (CFD) is, therefore, a key point in the design and verification of space vehicles, not only because it represents a cost-effective and quicker alternative to experimental verification, but also because experimental realisation and measurement of hypersonic flow conditions is often not feasible.

Hypersonic flow simulations are extremely challenging. Some of the outstanding issues in hypersonic flow simulation are: modelling of thermodynamic non-equilibrium phenomena, incorporation of boundary layer transition location models, simulation of ablation and gas-surface interactions, accurate modelling of radiation, accurate simulation of time-dependent separated flows, turbulent transport in the presence of real gas effects, handling of complex geometries and control surfaces, accurate simulation of localised heating due to jet and shock impingement, grid adaptation to resolve critical flow features such as shocks and shear layers. In this thesis we focus on the kernel of a fluid dynamic simulation, the numerical method used to solve the governing equations. Today, most CFD codes for hypersonic flow simulation employ second order shock-

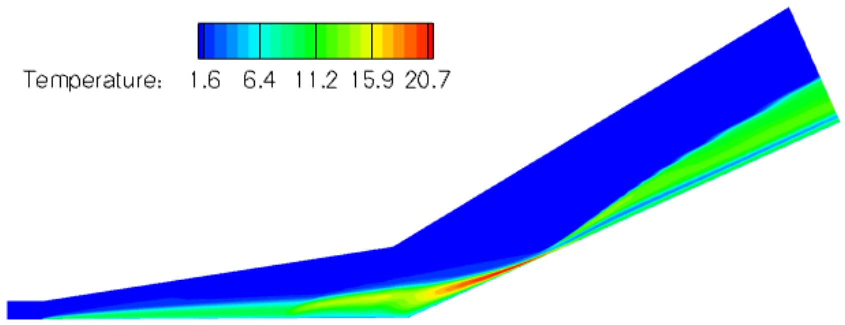


Figure 1.1: Hypersonic laminar shock-boundary layer interaction: high temperature and thin shock layer result in a very high wall heat flux on a flat plate-wedge geometry.

capturing schemes. Two common problems with these methods are: 1) high levels of dissipation, making them poor candidates for Large Eddy Simulation (LES); and 2) inadequate grid convergence properties. Sandham and Yee (2), for example, demonstrated that second-order *total variation diminishing* (TVD) schemes are unsuitable for Direct Numerical Simulation (DNS) of vortex pairing in a compressible shear layer (Fig. 1.2). It follows that there is considerable interest in designing robust and reliable high-order and high-resolution methods suitable for compressible, and in particular hypersonic, flow simulation.

Standard mesh-based numerical methods to solve partial differential equations can be divided into three main categories: finite difference (FD), finite volume (FV) and finite element (FE). In this thesis we consider finite difference methods, although the method studied can be applied in FV and FE frameworks as well, as long as a structured grid is considered. Two important characteristics of numerical methods are:

- Formal accuracy, that relates the discretisation error to the grid size. By definition, an n^{th} order accurate FD scheme computes a numerical derivative f_i' of any function $f(x)$ at a node x_i, such that the difference between the numerical derivative and the exact derivative $df(x_i)/dx$ is of magnitude Δx^n, where Δx is the grid size, i.e.:
$$f_i' = \frac{df(x_i)}{dx} + O(\Delta x^n).$$
- Resolution properties, that relate the range of length scales effectively resolved on a certain grid to the grid size. If the function to be discretised contains a wave of a certain spatial wavelength, the higher the resolution of the FD method, the

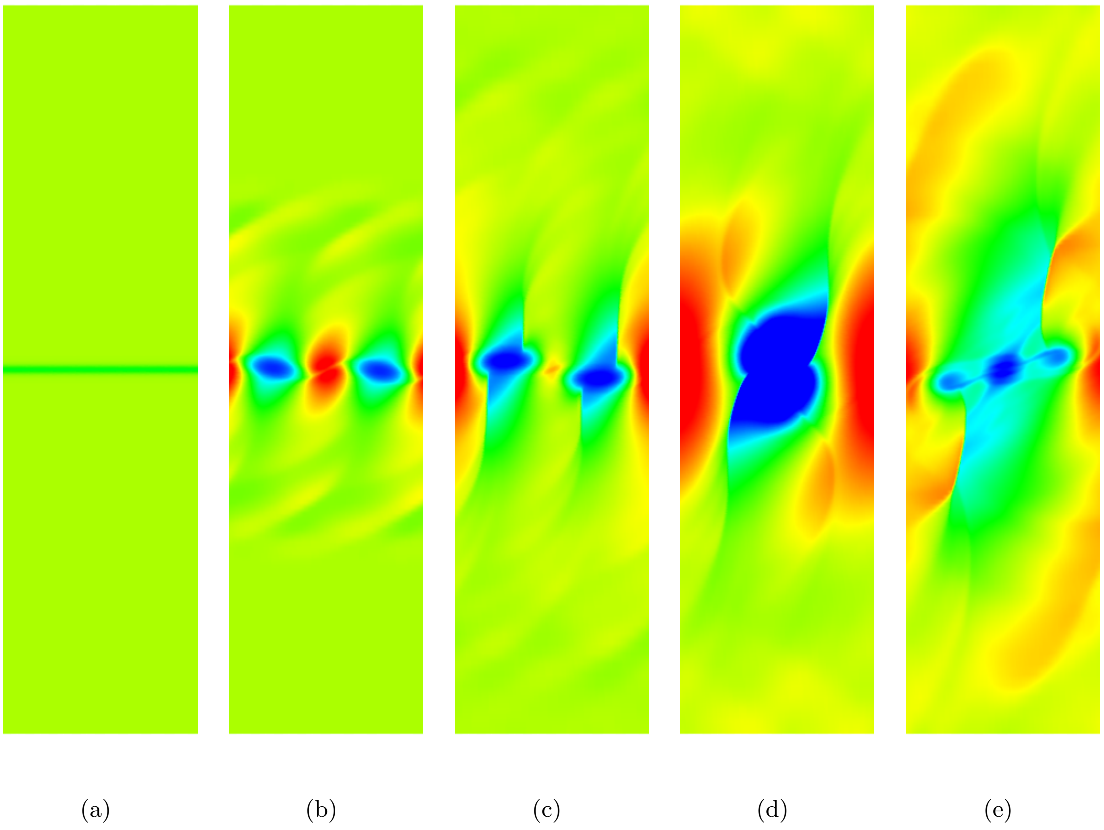


Figure 1.2: Density contours for vortex pairing in a compressible mixing layer: to correctly simulate this phenomenon requires higher-order numerical methods.

fewer grid points per wavelength are needed to capture the wave evolution, or, equivalently, on a particular grid higher resolution FD methods capture the time evolution of waves with smaller wavelengths. More about resolution properties is explained in Section 1.1.

It is clear that the formal accuracy determines the grid convergence properties of the numerical method, i.e. how quickly the computed solution converges to the exact solution as the grid is refined, and therefore it is important for any type of simulation. On the other hand, resolution properties are relevant only in applications, such as aeroacoustics, LES and DNS of turbulence, where the interaction between waves is the phenomenon to be simulated, and so it is crucial that the way the numerical method captures waves of different wavelength does not affect their simulated interaction.

CFD simulations can be very time consuming and so the time taken for the simulation to complete is a core issue. This is true especially for compressible flow simulations, where the non-linear character of the governing equations requires the use of computationally intensive numerical methods. Since the 1980s it was clear that in scientific and technical computing some of the tasks a program performs are not inter-dependent and so they could in principle be executed in parallel. Having many processing units performing these tasks in parallel can dramatically reduce the compute-time over a single processing unit. This very general idea is behind all parallel computers, from the vector processors developed in the 1980s to modern hybrid supercomputers hosting several shared-memory compute-nodes and accelerators. Modern High Performance Computing (HPC) is based on parallel processing, which is nowadays such a fundamental element of scientific computing in general, and of CFD in particular, that often certain numerical methods are preferred over others just because they are more suitable for parallel implementation. Certain types of CFD simulations are, in fact, only feasible if run on many Central Processing Units (CPUs). Also, CFD simulations are often part of a design or optimisation loop, and so are run many times. In these cases speeding up the simulation is a need rather than an option. While the improved performance of a single-core CPU has a direct positive impact on the performance of any code, using multiple CPUs, in a distributed or shared memory environment, requires careful thinking and often a re-design of the serial algorithm.

In this thesis we study and develop a high-order high-resolution numerical method for hypersonic flow simulation in the framework of high performance computing. In section 1.1 we build the background for the numerical method we have studied, introducing the basics of *compact* finite difference schemes. In section 1.2 we report a brief history of high performance computing, in order to understand the current scenario. In section 1.3 we review the HPC applications of CFD that are relevant to our research. Finally, in section 1.4, we highlight the main contribution of our work and outline the thesis.

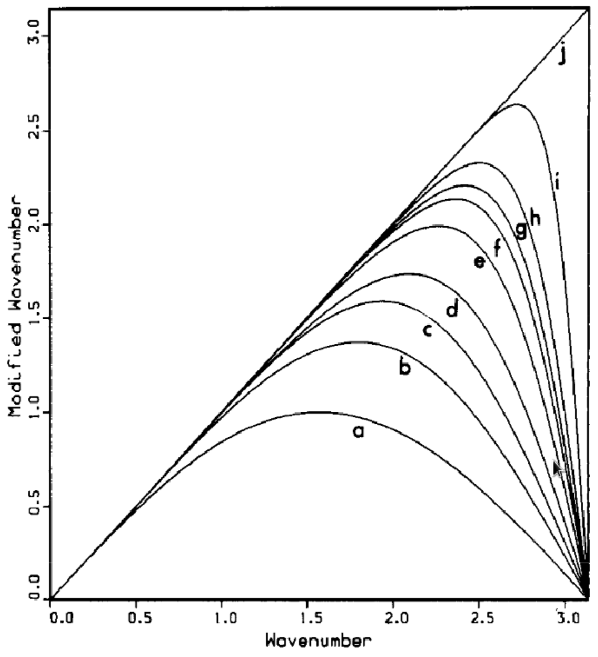


Figure 1.3: Modified wave-number vs. wave-number for first derivative approximations: (a) second order central differences; (b) fourth order central differences; (c) sixth order central differences; (d) standard Padé scheme ($\beta = c = 0$, $\alpha = 1/2$); (e) sixth order tri-diagonal scheme ($\beta = c = 0$, $\alpha = 1/3$); (f) eighth order tri-diagonal scheme ($\beta = 0$); (g) eighth order penta-diagonal scheme ($c = 0$); (h) tenth order penta-diagonal scheme; (i) spectral-like penta-diagonal scheme; (j) exact differentiation. Reproduced from Lele (3).

1.1 High-accuracy methods with spectral-like resolution for gasdynamics

1.1.1 Basics of compact schemes

Many physical phenomena possess a range of space and time scales, turbulent flow being a common example. The simulation of these processes require all the relevant scales to be properly represented in the numerical model. These requirements have led to the development of spectral methods (4, 5). The use of spectral methods is, however, limited to flows with simple domains and simple boundary conditions. For this reason, compact methods have also been developed; these have spectral-like resolution properties, are cheaper to use than spectral and pseudo-spectral schemes, and are easier to handle, especially when complex geometries are involved.

The interest in compact schemes started with the work of Lele (3). Lele developed compact approximations to first and second order derivatives in a finite difference framework and used Fourier analysis to determine the error involved with such approximations. We take Lele's paper as a reference to explain how compact schemes are developed and why they may be preferred to explicit finite difference schemes.

Given the values of a function $f(x)$ on a set of grid nodes, explicit FD formulas compute the function derivative as a linear combination of the given function values. For simplicity, consider a uniformly spaced mesh and let h be the grid spacing. Given function values at nodes, $f_i = f(x_i)$, the finite difference approximation, f'_i, to the first derivative, $df(x_i)/dx$, depends on the function values at neighbouring nodes. For instance, for second and fourth-order central differences, the approximation f'_i depends on sets (f_{i-1}, f_{i+1}) and $(f_{i-2}, f_{i-1}, f_{i+1}, f_{i+2})$, respectively. In spectral methods, however, the value of f'_i depends on all of the nodal values. Compact finite difference schemes mimic this global dependence. They are a generalisation of Padé schemes, derived by writing the approximation in the form:

$$\begin{aligned}\beta f'_{i-2} + \alpha f'_{i-1} + f'_i + \alpha f'_{i+1} + \beta f'_{i+2} = \\ \frac{c}{6h}(f_{i+3} - f_{i-3}) + \frac{b}{4h}(f_{i+2} - f_{i-2}) + \frac{a}{2h}(f_{i+1} - f_{i-1}).\end{aligned} \tag{1.1}$$

Writing $f(x)$ and $f'(x)$ as Taylor polynomials with x_i as the starting point, the relations between the coefficients a, b, c and α, β are found by matching the Taylor coefficients of various orders. The first unmatched coefficient determines the formal accuracy of the formula Eq. 1.1. These constraints are:

$$\begin{aligned}
a+b+c &= 1+2\alpha+2\beta \quad &\text{(second order)}\\
a+2^2b+3^2c &= 2\tfrac{3!}{2!}(\alpha+2^2\beta) \quad &\text{(fourth order)}\\
a+2^4b+3^4c &= 2\tfrac{5!}{4!}(\alpha+2^4\beta) \quad &\text{(sixth order)}\\
a+2^6b+3^6c &= 2\tfrac{7!}{6!}(\alpha+2^6\beta) \quad &\text{(eighth order)}\\
a+2^8b+3^8c &= 2\tfrac{9!}{8!}(\alpha+2^8\beta) \quad &\text{(tenth order).}
\end{aligned}$$

If $f(x)$ is periodic, then the system of Eqs. 1.1 written for each node is a closed linear system in the unknown derivative values. This system is a cyclic penta-diagonal (tri-diagonal) system when β is nonzero (zero). The general non-periodic case requires additional relations appropriate for the near-boundary nodes. As the derivatives are computed through the solution of a linear system, in compact schemes, as well as in spectral schemes, the numerical derivative at a node depends on all of the function values along a grid line. This is a way to explain the improved resolution properties of compact FD formulas over the explicit ones, as is shown later in this section.

The system of Eqs. 1.1, along with a mathematical transformation between a non-uniform physical mesh and a uniform computational mesh, provides derivatives on a non-uniform mesh. The most interesting schemes are obtained for $\beta = 0$ and $c = 0$, as: (i) the resulting system is tri-diagonal, and thus efficient algorithms, such as the Thomas algorithm, can be employed for the inversion; (ii) the stencil is narrow, and so handling the boundaries is easier. In this sub-family, the most accurate scheme is sixth-order accurate, and has the following coefficients:

$$\alpha = 1/3, \qquad \beta = 0, \qquad a = 14/9, \qquad b = 1/9, \qquad c = 0.$$

Lele (3) performed a Fourier analysis of this scheme, among others, and the results are shown in Fig. 1.3. The way this analysis was performed is outlined in the following.

Consider a uniform mesh made of N nodes, such that $x_i = h(i-1)$, where $i = 1, 2, \dots, N$, and assume the function $f(x)$ is periodic over the computational domain, i.e. $f_1 = f_N$. The Fourier decomposition of $f(x)$ is:

$$f(x) = \sum_{k=-N/2}^{N/2} \hat{f}_k \exp\left(\frac{2\pi \mathrm{i} k x}{L}\right), \tag{1.2}$$

where $\mathrm{i} = \sqrt{-1}$ and $\hat{f}_k$ are the Fourier coefficients. Since $f(x)$ is real-valued, its Fourier coefficients satisfy the condition $\hat{f}_k = \hat{f}^*_{-k}$ for $1 \le k \le N/2$ and $\hat{f}_0 = \hat{f}^*_0$, where $*$ denotes the complex conjugate. Introducing the scaled wave-number, $\hat{k} = 2\pi kh/L = 2\pi k/N$, and the scaled coordinate, $\hat{x} = x/h$, the basis functions for the Fourier expansions can be written as: $\exp(\mathrm{i}\hat{k}\hat{x})$. The exact first derivative of Eq. 1.2 with respect to $\hat{x}$ is a function with Fourier coefficient $\hat{f}'_k = \mathrm{i}\hat{k}\hat{f}_k$. The differencing error of the first derivative may be assessed by comparing the Fourier coefficients of the derivative obtained from the differencing scheme $(\hat{f}'_k)_{fd}$ with the exact Fourier coefficients $\hat{f}'_k$. For central difference schemes $(\hat{f}'_k)_{fd} = i\hat{k}'\hat{f}_k$, where the modified wave number $\hat{k}'$ is real-valued. Each finite difference scheme corresponds to a particular function $\hat{k}'(\hat{k})$. Exact differentiation corresponds to the straight line $\hat{k}'(\hat{k}) = \hat{k}$, and spectral methods provide the exact answer. Plots of the modified wave number $\hat{k}'$ against wave-number $\hat{k}$ are shown in Fig. 1.3, taken from Lele (3), for a variety of schemes. Note that the domain in terms of the scaled wave number is $\hat{k} \in [0, \pi]$. With this procedure, resolution characteristics of different schemes can be compared. It is particularly interesting to compare the curves labelled "c" and "e" in Fig. 1.3, corresponding to explicit and compact sixth order schemes, respectively. Even though the two schemes have the same formal accuracy, the compact one stays close to the exact differentiation over a wider range of wave-numbers.

Compact schemes have been extensively studied over the past two decades. Tam *et al.* (6), for example, developed dispersion-relation-preserving (DRP) finite difference schemes. The basic idea of these schemes is to optimise the coefficients of the compact schemes to improve their resolution properties in order to resolve waves solved with a few number of points per wavelength. The schemes presented by Lele and Tam *et al.* are intrinsically non-dissipative, as central differences are used to approximate the derivatives. Centred algorithms cannot prevent odd-even decoupling, which gives rise to high-frequency oscillations even in smooth regions. Extra filtering procedures, which are equivalent to adding numerical dissipation in an *ad hoc* manner, are needed in order to stabilise the computation. For example, central difference schemes fourth-order accurate or higher are unstable when coupled with high-order boundary schemes using one-sided finite difference approximations (7, 8). Carpenter *et al.* (8) showed that for a sixth-order inner compact central scheme, only a third-order boundary scheme can be used without introducing instability. This results in a globally fourth-order accurate scheme even if the inner scheme is sixth-order accurate.

A small amount of numerical dissipation is often necessary in practical applications in order to damp oscillations arising from initial and boundary conditions. For this purpose, compact upwind schemes have been developed (9, 10, 11) which drop the restriction of symmetric coefficients, allowing the scheme to be upwind even with a centred stencil. The upwinding introduces an amount of dissipation which is limited to high wave-numbers and can be adjusted by a careful design of the scheme.

Finite volume compact schemes have also been attempted by several authors (9, 12, 13), who have demonstrated that a conservative formulation, in addition to having inherent advantages for the computation of compressible flows, also provides better resolution properties compared to a non-conservative approach.

1.1.2 Compact methods for gasdynamics

Even asymmetric compact schemes cause non-physical oscillations when applied directly to flows with discontinuities. The non-physical oscillations (Gibbs phenomena) do not decay in magnitude when the grid is refined. Several methods have been proposed to stabilise compact schemes when resolving flows with discontinuities. These methods belong to one of the following three categories.

The first class are those proposed by Cook and Cabot (14, 15), Fiorina and Lele (16) and Cook (17). Higher-order background dissipation terms are added to a central scheme. These artificial dissipation terms are related to very high order derivatives so that the error introduced is smaller than the truncation error of the scheme. This strategy is by far the most accurate, as one has full control of the amount of dissipation introduced, just sufficient to resolve discontinuities and damp unphysical oscillations. Drawbacks of this strategy are: (i) the fine tuning required for the artificial diffusivity coefficients, often case-dependent; (ii) the complexity; (iii) the loss of the compactness of the stencil, because of the higher-order derivatives in the artificial dissipation terms.

Into the second category fall methods which blend the compact scheme with an essentially non-oscillatory scheme, such as ENO/WENO (18, 19, 20). Compact and ENO/WENO schemes have complementary properties: compact schemes have excellent resolution properties but oscillatory behaviour near discontinuities; ENO/WENO schemes are non-oscillatory but dissipative even for intermediate wave-numbers, and unsatisfactory in smooth regions with moderately high field gradients. By switching from a compact to a ENO/WENO scheme near discontinuities one can achieve uniformly high accuracy solutions with high resolution in smooth regions and non-oscillatory behaviour in regions with steep gradients. However, a free threshold parameter, which controls the switch between the compact and the ENO/WENO scheme, needs to be tuned, and some of the hybrid schemes (9, 21) experience non-smooth transition near the interface where the schemes switch. Spurious waves will eventually propagate into the smooth regions, as reported by Adams and Shariff (9). Ren *et al.* (22) have developed a characteristic-wise hybrid scheme, which can be regarded as an improvement of the method proposed by Pirozzoli (21).

Methods belonging to the third category rely on a classical limiting strategy. Cockburn and Shu (23) have developed a non-linear limiter to avoid spurious oscillations while maintaining the formal accuracy of the scheme. However, in their numerical tests, spurious oscillations were still evident. Ravichandran (24) employed a TVD limiter combined with a Kinetic Flux Vector Splitting (KFVS) method to improve the stability of compact upwind schemes, proposing a third order scheme supposed to degenerate to first order accuracy at the extrema. Recently, Tu and Yuan (25) proposed a method where a compact upwind scheme is limited through a characteristic-based approach. The limiting approach is not as common as the hybridising approach, as degeneration of accuracy near extrema is an undesirable feature when solving, for example, shock-turbulence interaction problems. Furthermore, Cockburn and Shu (23) found that the introduction of minmod limiters in centred schemes affects the accuracy of the solution even in smooth regions, if there are spurious oscillations there to suppress. On the other hand, Ravichandran (24) showed that upwinding the compact scheme and employing kinetic splitting provides effective damping of spurious oscillations in smooth regions, and thus the accuracy degenerates just across discontinuities. This approach is attractive also for its relative simplicity and robustness, and lower computational cost.

Figure 1.4: CDC 6600, one of the first supercomputers, designed in the '60s.

1.2 A brief history of High Performance Computing

Many applications, especially in the area of scientific and technical computing, have always increasing computational requirements, and so drive the development of machines much more powerful than common computers, often referred to as *supercomputers*. The term supercomputer itself is rather fluid, and, because of the fast improvements in computing technologies, today's supercomputer tends to become tomorrow's ordinary computer. The term High Performance Computing (HPC) refers to scientific and engineering applications which require the use of supercomputers. In the past two decades supercomputers' outstanding computing power has been achieved through having several inter-connected Central Processing Units (CPUs) working in parallel to solve a problem, and so today the terms *parallel computing* and high-performance computing are closely related.

The first computers were actually supercomputers, special devices developed for military or research applications, often unique. This is the case, for example, of Colossus (1943) and Manchester Mark I (1945) in the UK, and of MIT Whirlwind (1950) in the US. The first parallel computer is probably UNIVAC LARC (26), developed in the US

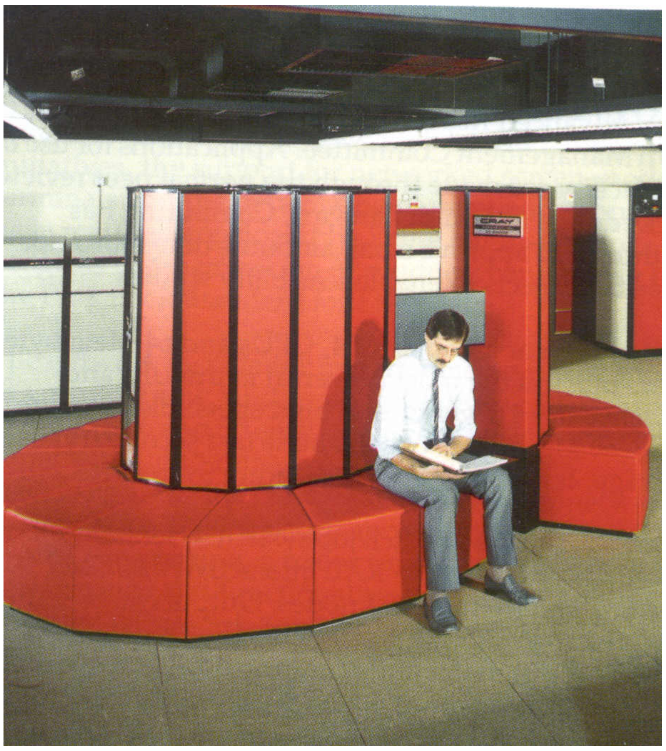


Figure 1.5: Cray X-MP, example of supercomputer design in the '80s, based on vector processors.

at Lawrence Livermore Laboratory in 1960: it used two CPUs for computing and a processor dedicated to Input/Output (I/O) operations. The success of this computer was very limited, because at the time CPUs ran slower than the main memory they were attached to. Therefore, there was a significant time where the main memory was idle. This idle time was exploited by the CDC 6600, shown in Fig. 1.4, often considered the first successful supercomputer design (27). The CDC 6600 computer, designed by Seymour Cray, had 10 Peripheral Processors (PPs) and a single CPU. The CPU was able to perform arithmetic and logic operations only and was controlled in turn by one of the PPs. While a PP was using the CPU to perform some computation, the others were performing other tasks, such as loading data into the main memory, and so all of the PPs were effectively working in parallel. Furthermore, because the CPU had to handle arithmetic and logic operations only, it had a simple design and was much faster than any other CPU at the time. Therefore, the CDC 6600 is also the first example of a *Reduced Instruction Set Computer*. In 1965 it was the fastest computer in the

Figure 1.6: Cray Jaguar, an example of a modern MPP system, rated the third fastest supercomputer in the world in June 2011.

world, with a peak speed of 36 Mega FLoating-point OPerations per Second (FLOPS). In 1972 Cray left CDC to found Cray Research Inc., later becoming Cray Inc., which today is one of the leading HPC companies.

The CDC 6600 was able to overlap computing and memory I/O, but from the computing point of view it was still a serial machine. The first machines able to perform several calculations in parallel appeared in the 1970s and were based on *vector processors*. At the time the main memory was big enough to store all the necessary data and the CPUs were much faster than the main memory, so the problem was how to effectively exchange data between the CPU and the main memory. Vector processors applied a single algorithm to a large data set, fed in the form of an array. This type of processing, referred to as Single Instruction Multiple Data (SIMD) in Flynn's taxonomy (28), relied on certain features of such processors; while instruction pipelining and different functional units for different instructions were already featured in previous designs, peculiar to vector processors were data pipelining and the availability of several functional units for the same instruction. Supercomputers powered by a vector processor were the CDC Star-100 (1974), Cray-1 (1975) and the CDC Cyber 205 (1981), the last one achieving a peak speed of 400 MFLOPS. Vector processors first exploited the parallel nature of some algorithms, i.e. the property of such algorithms to be split into tasks which are not inter-dependent, and so can be performed in parallel without altering the final result. This idea was taken a step further and machines like the Cray

Rank	Site	Computer
1	Riken AICS, Japan	Kei - Fujitsu 8-core SPARC64 VIIIfx
2	NSC Tianjin, China	Tianhe-1A - NUDT 6-core Intel X5670 2.9 GHz, NVIDIA M2050 GPU
3	DOE/SC/ORNL, US	Jaguar - Cray XT5 6-core AMD 2.6 GHz
4	NSC Shenzhen, China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVIDIA C2050 GPU
5	GSIC Center Tokyo, Japan	Tsubame - HP ProLiant SL390s G7 node, NVIDIA M2050 GPU
6	DOE/NNSA/LANL/SNL, US	Cielo - Cray XE6 8-core AMD 2.4 GHz
7	NASA Ames, US	Pleiades - SGI Altix 4-core Intel X5670 2.9 GHz
8	DOE/SC/LBNL/NERSC, US	Hopper - Cray XE6 12-core 2.1 GHz
9	CEA, France	Tera-100 - Bull bullx super-node S6010/S6030
10	DOE/NNSA/LANL, US	Road Runner - IBM BladeCenter PowerXCell i8 / AMD 1.8 GHz

Table 1.1: World top 10 supercomputers, extracted from the top500 published in June 2011(`http://www.top500.org/lists/2010/11`).

X-MP (1983), Cray-2 (1985), Cray Y-MP (1988) and ETA-10G (1989) were powered by 4 to 8 vector processors attached to a *shared memory*. For example, the Cray X-MP, shown in Fig. 1.5, had 4 vector processors and achieved a peak performance of 400 MFLOPS.

In the 1990s improvements in scalar processors, particularly microprocessors, resulted in the decline of traditional vector processors in supercomputers and the appearance of vector processing techniques in mass-market CPUs. Attention turned then to Massively Parallel Processing (MPP) systems, hosting hundreds of CPUs, each one attached to its own memory, and interconnected through a fast network. This type of architecture has dominated the HPC scene since. Indeed, MPP machines fully exploit the parallelism of an algorithm and allow the maximum flexibility, having hundreds of CPUs working independently, following a Multiple Instruction Multiple Data (MIMD) type of processing (28). Also, these systems can employ commodity hardware and benefit from the hardware development driven by the mass-market, and so they are very cost-effective. On the negative side, even the most parallel algorithms require some information to be shared between the CPUs. In a distributed memory system information sharing takes place through message passing within the network, and the communication cost can significantly impact the overall performance. Latest designs of processors, hosting several execution units or *cores*, each one acting as a CPU, have further complicated the problem: the cores share memory and network interconnects and so making effective use of the available resources is far from trivial. Some of the most famous MPP systems are Intel ASCI Red (1995), IBM ASCI White (2000), Earth Simulator (2002), IBM ASCI Blue Gene (2005), Cray Jaguar (2009). For example, the Cray Jaguar XT5 system (`http://www.nccs.gov/computing-resources/jaguar/`), shown in Fig. 1.6, has 18,688 compute nodes, each one containing two hex-core processors. With a peak speed of 1.75 Peta FLOPS (PFLOPS), it is currently (June 2011) rated the third fastest supercomputer in the world.

Recently, a new trend has emerged in HPC, where standard CPUs are used in combination with co-processors. These devices are used by the CPU to parallel process large arrays of data in a SIMD fashion. Devices of this type are, for example, Graphic Processing Units (GPUs) and Power Cell processors. Originally developed for the mass-market and used for video processing, they employ those vector processing techniques first exploited by vector processors in the 1980s, improved over two decades to achieve computing powers up to Tera FLOPS. The applications that could use such computing power were limited by the special-purpose nature of video processing and the difficulty to program these devices. While these limitations have hindered the popularity of Cell Processors, GPUs have evolved to become more useful as general-purpose vector processors, and a computer science sub-discipline has arisen to exploit this capability, namely General-Purpose computing on Graphics Processing Units (GPGPU). In the first 10 positions of the top500, the list of the world's most powerful computers, published in June 2011 (`http://www.top500.org/lists/2010/11`), the second, fourth and fifth positions are occupied by systems using GPUs as co-processors (Table 1.1). In tenth position lies the Road Runner: this was the first system able to break the PFLOPS barrier, and employs Cell Processors as accelerators. Another reason for the recent success of accelerators is their high energy efficiency: power consumption has become such an important issue in HPC that, for example, IBM has traded processor speed for a lower power consumption in its MPP system Blue Gene. The GPU accelerated

Tianhe-1A is capable of 2.57 PFLOPS consuming 4 MW, while the traditional system Jaguar is only capable of 1.75 PFLOPS but consumes 6.9 MW.

1.3 Computational Fluid Dynamics and High Performance Computing

1.3.1 Parallel application of compact schemes

Computational Fluid Dynamics applications are among those driving the development of high performance computing. Typical CFD applications that require the use of HPC are climate and ocean modelling, DNS and LES of turbulence, and, in general, applications where the smallest length scale to be simulated is much smaller than the size of the domain. The parallel implementation of CFD methods is a very extensive topic, and reviewing how parallel computing has been used to solve CFD problems is not the scope of this section. Here we focus only on the parallel computing applications of compact finite-difference schemes.

Compact schemes have a semi-global nature, as a linear system must be solved to compute spatial derivatives along grid lines. Efficient parallelisation of these schemes therefore represents a significant challenge. Nevertheless, parallel implementation of such methods is of great interest, because, as highlighted in section 1.1, they are exceptional candidates for DNS and LES of turbulence, that are, indeed, highly computationally intensive. Previously, Sun and Moitra (29) and Povitsky (30) devised parallel algorithms for the solution of the underlying linear system, but, as highlighted by Ladeinde *et al.* (31), these algorithms scale poorly at high processor counts. Improved scalability can be achieved if a structured-block domain decomposition technique is employed: Gaitonde (32) and Sengupta *et al.* (33) have employed compact schemes to advance independently the solution in overlapping sub-domains. Laizet *et al.* (34) have proposed a dual-domain decomposition, namely *slab* decomposition, which changes during a single time step depending on the spatial direction processed. Their incompressible DNS code showed good parallel performance, even though in some tests communication took up to 40% of the total simulation time because of the global data transposition implied by the dual decomposition.

The class of compact upwind methods developed by Pirozzoli (21) have opened new possibilities for the application of a classical and efficient parallel multi-block strategy to compact schemes. These compact schemes compute a numerical flux function, unlike other compact schemes which compute derivatives, and so a multi-block parallelisation approach can be implemented by simply enforcing the continuity of the numerical flux function across block-interfaces. Chao *et al.* (35) have exploited this possibility by developing a multi-block compact method. Their parallel algorithm is based on a compact-WENO scheme employing domain decomposition.

The parallelisation strategies discussed so far are task-based and rely on the Single Program Multiple Data (SPMD) programming model. The modern trend in High Performance Computing is to increase the number of cores sharing the memory on a compute-node. In a shared memory environment, thread-based parallelism is possible, based on the Symmetric Multi-Processing (SMP) programming model. For large-scale fluid dynamics calculations employing compact schemes, this is particularly interesting, as efficient parallelisation in a distributed memory environment is possible by weakly relaxing the global dependency of the scheme. As noted by Chao *et al.* (35), this will affect the global resolution properties of the scheme as the number of cores is increased. In a shared memory environment, parallelisation is in principle possible without modifying the numerical method (and so preserving the resolution properties), even though attention must be paid in order to achieve good parallel performance. Many investigations into dual-level parallelism have been published. For example, Huan and Tafti (36) combined multi-threading and multi-tasking to achieve a good load balance when the governing equations are solved using Adaptive Mesh Refinement. Gropp *et al.* (37) have shown that decreasing the number of tasks and increasing the number of threads can speed up explicit flux computation as the mesh size increases. Recently, Rabenseifner *et al.* (38) have shown that a dual-level parallelism mapped onto the machine topology can significantly speed up a parallel block-tri-diagonal CFD solver.

1.3.2 GPU-accelerated CFD

In the last decade the mass-market has driven the development of extremely powerful Graphic Processing Units, extensively used to run computer animations. The development of the hardware and also of high-level programming languages, such as Cg, BrookGPU, OpenCL and CUDA, has made GPUs attractive for scientific computing in general, and CFD in particular.

Fluid dynamic simulations have been one of the first testing grounds for GPU-accelerated physics-based animations. Flow solvers for computer graphics are based on Stam's method (39), the so-called *Stable Fluids* algorithm. This is a semi-Lagrangian method solving the Navier-Stokes equations, and is very quick and suitable for vector processing. It is also stable and so, even though it is explicit in time, it allows large time-steps to be used in unsteady simulations. The method is not accurate enough for engineering computation, but it captures the major characteristics of fluid motion, and so it is very popular for physics-based graphics applications where accuracy is not essential but speed is. Applications of Stam's method can be found, for example, in the work by Liu *et al.* (40) and Harris *et al.* (41). Liu *et al.* (40) performed several 3D flow simulations, e.g. flow over a city, and their goal was to have a real-time solver along with visualisation running on the GPU, while Harris *et al.* (41) used Stam's method for cloud visualisation.

Accurate CFD methods can also benefit from the computing power of GPUs. However, performance strongly depends on the hardware, as one may expect, but also on the method, mostly because of the very tight constraints on the memory access patterns on GPUs, and, surprisingly, on the programming language, because compiler optimisations play a fundamental role in GPUs even more than in CPUs. In the following we review some applications and corresponding performance figures. Speed-ups are referenced to a single-core CPU implementation of the same method, unless differently specified.

Fluid dynamic simulations of engineering interest have been performed using Lattice-Boltzmann methods. The Lattice-Boltzmann model (LBM) is attractive for GPU processors since it is simple to implement, is extremely parallel, and incurs a significant computational cost. Early work by Li *et al.* (42, 43) showed modest speed-ups around 6×, obtained using Cg on an NVIDIA GeForce FX 5900 Ultra. More recent work has shown the real potential of GPUs for LBM. Tölke and Krafczyk (44) achieved a speed-up of 100× with their CUDA implementation of the LBM on an NVIDIA 8800 Ultra, while Simek *et al.* (45) performed atmospheric dispersion simulations, achieving speed-ups as high as 77×.

More standard CFD methods are less GPU-friendly and rarely achieve such high performance. Incompressible flow solvers have been implemented on a single GPU by Scheidegger *et al.* (46) and Cohen and Molemaker (47). Scheidegger *et al.* (46)

implemented the 2D Simplified Marker And Cell (SMAC) method (48), a technique used for free-surface flow simulations, and obtained speed-ups on NVIDIA GeForce FX 5950 Ultra and 6800 Ultra varying from 7× to 21×. Cohen and Molemaker (47) implemented an incompressible Navier-Stokes multi-grid solver with Boussinesq approximation. Their implementation on an NVIDIA Tesla C1060 was 8× faster than a multi-threaded code running on a quad-core Intel Xeon E5420 at 2.5GHz. Their code supports double precision, although this made it 46% to 66% slower than in single precision.

Extremely relevant to our present research is the work on compressible Euler equations by Brandvik and Pullan (49, 50) and Elsen *et al.* (51). Brandvik and Pullan (49, 50) implemented a FV structured solver for the Euler equations in 2D and 3D and considered both BrookGPU and CUDA tool-kits. They achieved speed-ups of 29× in 2D and 16× in 3D when using CUDA, while using BrookGPU they could only achieve a speed-up of 3× in 3D. Elsen *et al.* (51), instead, successfully used BrookGPU to port a subset of a Navier-Stokes solver. The GPU-based code solves the 3D compressible Euler equations on a single GPU. They simulated the flow over a hypersonic vehicle using an NVIDIA GTX8800 and measured speed-ups from 15× to 40× with respect to a single core of an Intel Core 2 Duo E6600 2.4GHz.

The numerical methods used by Brandvik and Pullan (49, 50) and Elsen *et al.* (51) are second-order accurate at most. Work on higher-order shock-capturing methods on GPUs is extremely limited. Antoniou *et al.* (52) performed LES on the Rayleigh-Taylor instability using a GPU implementation of a WENO scheme. They used up to 4 devices, available in an NVIDIA Tesla S1070. Speed-ups from 17× to 27× were reported when using a single device vs. two cores of a 3.0 GHz quad-core Intel Xeon X5450. Griebel and Zaspel (53) ported to GPU part of a 3D two-phase incompressible Navier-Stokes code. The code is based on the *level set* technique, and employs WENO formulas for the level set re-initialisation stage. This stage was ported to GPU using CUDA and double precision arithmetic was used. For this task, speed-ups from 9× to 13× with respect to a single core of a 2.66 GHz quad-core Intel Core i7 were measured.

To our knowledge, the only GPU application of a compact scheme can be found in the work by Hagen *et al.* (54). They implemented a FV structured solver for the Euler equations using Cg. Several numerical schemes for the cell-face fluxes were considered, including a compact-WENO scheme (55). They considered two compute-nodes: the first had an Intel Xeon 2.8 GHz as host and an NVIDIA GeForce 6800 Ultra as accelerator; the second had an AMD Athlon X2 4400+ as host and an NVIDIA GeForce 7800 GTX as accelerator. For each node, speed-ups were measured comparing run-times on the host to run-times on the attached device. They simulated a 2D shock-jet interaction and measured speed-ups from $8\times$ to $9\times$ for the first node, and from $14\times$ to $19\times$ for the second node. The higher speed-ups found for the second node were attributed to an under-performing compiler: while in the first node the Intel compiler could fully exploit the potential of the Intel CPU, in the second node the GNU compiler could not generate an optimal code for the AMD CPU.

1.4 Objectives and thesis outline

The focus of this thesis is the application of compact schemes for the simulation of hypersonic flows. Compact schemes are of interest for applications which require higher-order accuracy and spectral-like resolution, such as LES and DNS of turbulence. As these applications are in general extremely computationally demanding, it is essential to study such methods in the context of high-performance computing.

In section 1.1.2 three classes of compact-based methods for gasdynamics have been discussed: methods of the first type use higher order dissipation terms, those of the second type blend compact and ENO/WENO schemes and those of the third type apply TVD filters. We have selected a compact-TVD scheme for the Euler equations (25), as a solution method to study, extend and improve. This method has been chosen because it is believed to be a good compromise between accuracy, robustness and computational cost. The description of the method, along with our modifications and improvement, is the subject of Chapter 2.

We also propose a method to solve the Navier-Stokes equations, combining this compact-TVD method with a Kinetic Splitting technique for the Navier-Stokes flux (56). The resulting method has improved shock capturing properties and a computational advantage over conventional methods based on operator splitting. The stability and accuracy of the method are demonstrated through a wide range of test cases. A detailed description of the numerical method is the subject of Chapter 3.

We then consider how this method may be used in the context of high-performance computing, exploring the three main paradigms of modern supercomputing:

- Parallelisation through message passing. We explore a new parallelisation technique, assessing its performance and impact on computed results.
- Parallelisation through memory sharing. We highlight performance issues and identify the causes.
- Acceleration through GPU. We develop a compressible Navier-Stokes code based on a compact scheme running entirely on a GPU. We identify a strategy to overcome the memory constraints associated with GPU computing, and we quantify obtainable speed-ups.

The different parallelisation strategies are discussed Chapter 4.

We assess the capability of our method over an extensive set of test cases, involving one-, two- and three-dimensional flows, supersonic and hypersonic flows, and compressible DNS-oriented test cases. These tests are all presented along with reference experimental results, analytical or independent numerical solutions in Chapters 5 and 6. Finally, in Chapter 7 we draw our conclusions.

2

A compact-TVD method for the solution of the Euler equations

In this chapter we describe the compact-TVD method proposed by Tu and Yuan (25) and the techniques it employs. The chapter is structured as follows: in section 2.1 the theory of conservative compact-upwind approximations for derivatives is outlined. The compact formula is given, along with its boundary closures, and its properties discussed. In section 2.2 upwind-TVD schemes based on flux splitting are introduced and two limiter functions are given. Section 2.3 describes how the techniques introduced in sections 2.1 and 2.2 are combined to obtain the inviscid flow solver proposed by Tu and Yuan (25). In section 2.4 the time discretisation is given. Finally, section 2.5 provides details about the implementation of the method and the enforcement of boundary conditions.

2.1 Conservative compact-upwind finite difference schemes

In this section we present the compact scheme used to discretise the inviscid flux in the Euler solver proposed by Tu and Yuan (25). This method belongs to a class of methods derived by Pirozzoli (21), which possess features highly desirable for the simulation of compressible flows:

- they are upwind and therefore have built-in numerical dissipation for improved stability;

- they are conservative and correctly locate discontinuities, according to the Lax-Wendroff theorem (57).

In section 2.1.1 we detail the derivation of this class of methods. In section 2.1.2 we present a resolution study, showing why compact formulas may be preferred to explicit ones. Finally, in section 2.1.3 we present the formulas employed at the domain boundaries in non-periodic problems.

2.1.1 Derivation of conservative compact-upwind formulas

As customary, $f(x)$ is a function whose values are known on a set of nodes, $x_1, x_2, \ldots, x_N$, and values of its derivative, $f'(x)$, over the same set of nodes must be determined. The nodes are equally spaced over the domain and h is the grid spacing. Unlike more conventional compact schemes, such as those in section 1.1, the derivative is not directly reconstructed, but computed from the numerical flux function. The numerical flux function $\hat{f}(x)$ is, by definition, a function such that the difference between its values at the intermediate nodes $i+1/2$ and $i-1/2$ gives a r^{th} order approximation of the derivative $f'(x_i)$:

$$\frac{1}{h}(\hat{f}_{i+1/2} - \hat{f}_{i-1/2}) = f'(x_i) + O(h^r), \qquad i = 1, 2, \ldots, N. \tag{2.1}$$

Computing the derivative from the numerical flux function automatically generates a conservative discretisation. Reconstructing the numerical flux is equivalent to an interpolation problem (18), where values of a function $g(x)$ at the intermediate nodes, $x_{i+1/2}$ for $i = 1, 2, \ldots, N$, must be evaluated from its cell-average values,

$$\overline{g}_i = \frac{1}{h}\int_{x_{i-1/2}}^{x_{i+1/2}} g(x) \quad dx, \qquad i = 1, 2, \ldots, N.$$

Functions $f(x)$ and $g(x)$ are related by:

$$\hat{f}_{i+1/2} = \hat{g}_{i+1/2}, \qquad f_i = \overline{g}_i, \tag{2.2}$$

where $\hat{g}_{i+1/2}$ is the estimated value of $g(x)$ at intermediate node $x_{i+1/2}$. Given the identities in Eq. 2.2, Eq. 2.1 is equivalent to:

$$\hat{g}_{i+1/2} = g(x_{i+1/2}) + O(h^r). \tag{2.3}$$

A compact reconstruction of $g(x)$ around the intermediate node $x_{i+1/2}$ is:

$$\sum_{l=-L_1}^{L_2} \alpha_l \hat{g}_{i+1/2+l} = \sum_{m=-M_1}^{M_2} a_m \overline{g}_{i+m}, \tag{2.4}$$

that is equivalent to:

$$\sum_{l=-L_1}^{L_2} \alpha_l \hat{f}_{i+1/2+l} = \sum_{m=-M_1}^{M_2} a_m f_{i+m}, \tag{2.5}$$

where L_1, L_2, M_1, M_2 are integer constants that determine the stencil for the reconstruction. Assuming that the function $g(x)$ can be expanded in a Taylor series up to order R around $x_{i+1/2}$,

$$g(x) = \sum_{n=0}^{R-1} g^{(n)}_{i+1/2} \frac{(x - x_{i+1/2})^n}{n!} + O(h^R), \tag{2.6}$$

then, as the cell-average value of $g(x)$ can be written as

$$\overline{g}_{i+m} = \frac{G_{i+m+1/2} - G_{i+m-1/2}}{h},$$

where $G(x)$ is the primitive function of $g(x)$, i.e.

$$G(x) = \sum_{n=0}^{R-1} g^{(n)}_{i+1/2} \frac{(x - x_{i+1/2})^{n+1}}{(n+1)!} + O(h^{R+1}),$$

the following expression for the cell-average values of $g(x)$ holds:

$$\overline{g}_{i+m} = \sum_{n=0}^{K-1} g^{(n)}_{i+1/2} \frac{1}{(n+1)!} [m^{n+1} - (m-1)^{n+1}] h^n + O(h^{R+1}). \tag{2.7}$$

Inserting Eqs. 2.6 and 2.7 into Eq. 2.4, recalling Eq. 2.3 and matching the coefficients for the n^{th} power of h, the following formula is found:

$$(n+1) \sum_{l=-L_1}^{L_2} \alpha_l l^n = \sum_{m=-M_1}^{M_2} a_m [m^{n+1} - (m-1)^{n+1}]. \tag{2.8}$$

For a r^{th} order interpolation, where $r \leq R$, Eq. 2.8 shows that a system of r equations, for $n = 0, \ldots, r-1$, must be satisfied. The resulting linear system must be solved to determine the coefficients α_l ($L_1 + L_2 + 1$ unknowns) and a_m ($M_1 + M_2 + 1$ unknowns).

As the linear system is homogeneous, at least one coefficient must be set to a non-zero quantity in order to exclude the trivial solution (all coefficients equal to zero), for example $a_{-M_1} = 1$, and so the unknown coefficients are $L_1 + L_2 + M_1 + M_2 + 1$.

Choosing $L_1 = L_2 = M_1 = M_2 = 1$ yields compact-upwind interpolations with a narrow stencil, which only require the inversion of a tri-diagonal matrix. The most accurate scheme on this stencil is fifth-order accurate and has coefficients:

$$\alpha_{-1} = 9, \alpha_0 = 18, \alpha_1 = 3, a_{-1} = 1, a_0 = 19, a_1 = 10.$$

Recalling Eq. 2.2, a fifth-order compact-upwind reconstruction formula for the numerical flux function is obtained:

$$9\hat{f}_{i-1/2} + 18\hat{f}_{i+1/2} + 3\hat{f}_{i+3/2} = f_{i-1} + 19f_i + 10f_{i+1}. \tag{2.9}$$

It is important to point out that the system of Eqs. 2.9 is diagonally dominant, and so its numerical inversion through a tri-diagonal direct solver, such as the Thomas algorithm, is stable.

2.1.2 Resolution properties

Equation 2.5 is very general, in fact it describes any conservative linear finite-difference (FD) formula. Therefore, starting from Eq. 2.5, the resolution properties of any conservative linear FD scheme can be investigated using the linear advection equation:

$$v_t(x,t) + cv_x(x,t) = 0, \qquad c > 0, \tag{2.10}$$

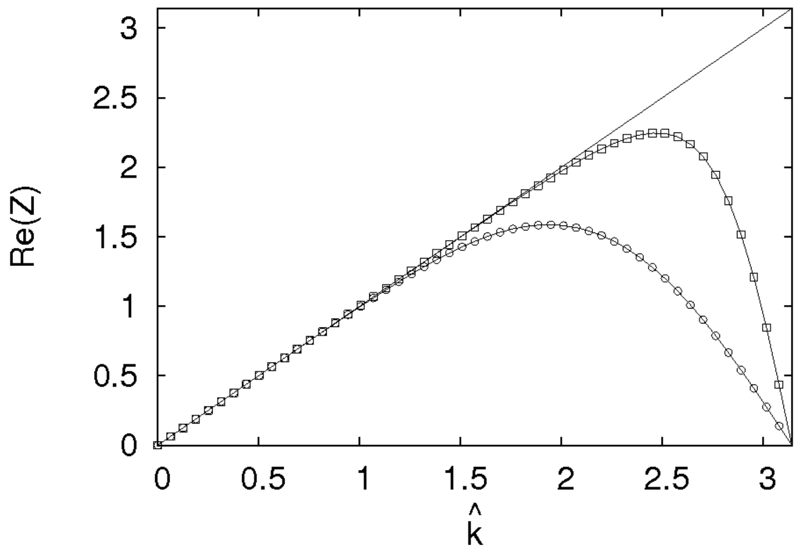
as a model equation. A mono-chromatic wave with wavenumber k will be investigated, with $v(x,0) = e^{\mathrm{j}kx}$ set as the initial condition, where $\mathrm{j} = \sqrt{-1}$. The exact solution to this problem is:

$$v(x,t) = e^{\mathrm{j}k(x-ct)}. \tag{2.11}$$

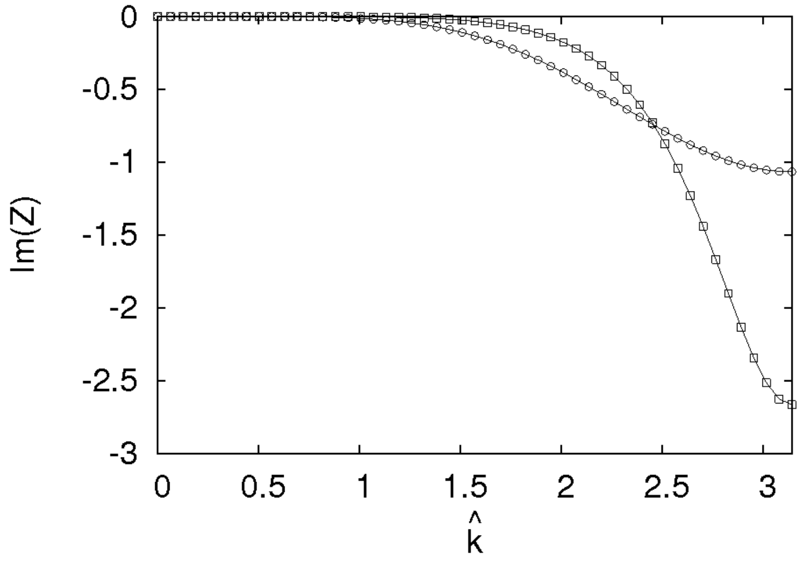
The semi-discrete form of Eq. 2.10, when using a uniform grid, $x_i = ih$ for $i = 1, 2, \dots, N$, is:

$$\frac{dv_i(t)}{dt} + c\frac{\hat{v}_{i+1/2}(t) - \hat{v}_{i-1/2}(t)}{h} = 0. \tag{2.12}$$

The initial condition is $v_i(0) = e^{\mathrm{j}\hat{k}i}$, where $\hat{k} = kh$ is the *scaled wavenumber*. It is related to the number of grid-points per wave-length, n_p, by: $\hat{k} = 2\pi/n_p$. As the minimum



(a) Dispersion curve



(b) Dissipation curve

Figure 2.1: Dispersion and dissipation properties for different approximation schemes. –□–: fifth-order compact-upwind; –◯–: fifth-order explicit-upwind; ——: exact solution.

number of points to describe a wave is $n_p = 2$, the range of scaled wave-numbers to consider is $\hat{k} \in [0, \pi]$. The solution to the semi-discrete problem is assumed to have the form:

$$v_i(t) = e^{\mathrm{j}\hat{k}\left(i - \frac{c}{h}\frac{Z(\hat{k})}{\hat{k}}t\right)}, \tag{2.13}$$

where $Z(\hat{k})$ is a complex-valued function, called a *transfer function.* Since at any time the numerical flux function, $\hat{v}_{i+1/2}(t)$, is a linear combination of exponential functions like $v_i(t)$, the following expression holds:

$$\hat{v}_{i+1/2+l}(t) = \hat{v}_{i+1/2}(t)e^{\mathrm{j}\hat{k}l}. \tag{2.14}$$

Substituting Eqs. 2.13 and 2.14 into Eq. 2.5, gives:

$$\hat{v}_i \sum_{l=-L_1}^{l=L_2} \alpha_l e^{\mathrm{j}\hat{k}(1/2+l)} = v_i \sum_{m=-M_1}^{m=M_2} a_m e^{\mathrm{j}\hat{k}m}. \tag{2.15}$$

Substituting Eqs. 2.13 and 2.14 into Eq. 2.12, gives:

$$\mathrm{j}Z(\hat{k})v_i = \hat{v}_i(e^{\mathrm{j}\hat{k}/2} - e^{\mathrm{j}\hat{k}/2}),$$

which, recalling that $\sin x = (e^{\mathrm{j}x} - e^{-\mathrm{j}x})/2\mathrm{j}$, is equivalent to:

$$Z(\hat{k})v_i = 2\hat{v}_i \sin \hat{k}/2. \tag{2.16}$$

Combining Eqs. 2.15 and 2.16, we finally obtain the following expression for the transfer function:

$$Z(\hat{k}) = 2\sin \hat{k}/2 \frac{\sum_{m=-M_1}^{M2} a_m e^{\mathrm{j}\hat{k}m}}{\sum_{l=-L_1}^{L_2} \alpha_l e^{\mathrm{j}\hat{k}(l+1/2)}}. \tag{2.17}$$

The transfer function, $Z(\hat{k})$, completely defines the capability of a numerical scheme to propagate a wave with a certain wavenumber. Separating real and imaginary parts of the transfer function, the numerical solution reads:

$$v_i(t) = e^{\frac{c}{h}\mathrm{Im}[Z(\hat{k})]t} e^{\mathrm{j}\hat{k}\left(i - \frac{c}{h}\frac{\mathrm{Re}[Z(\hat{k})]}{\hat{k}}t\right)}. \tag{2.18}$$

Comparing the numerical solution, Eq. 2.18, with the exact solution, Eq. 2.11, it is evident that:

- The exact solution is a wave with constant amplitude, while the numerical solution's amplitude changes proportionally to $e^{\frac{c}{h}\mathrm{Im}[Z(\hat{k})]t}$. For $c > 0$, a numerical method is stable if $\mathrm{Im}[Z(\hat{k})] \leq 0$. Also, for a stable scheme, $\mathrm{Im}[Z(\hat{k})]$ represents the *dissipation error*, as it determines how the wave amplitude is damped over time.

- The exact solution translates in time at speed c, while the numerical solution translates at speed $c\mathrm{Re}[Z(\hat{k})]/\hat{k}$. Therefore, $\mathrm{Re}[Z(\hat{k})]$ represents the *dispersion error.*

In Fig. 2.1 the dissipation and dispersion properties of two schemes are compared. We consider the fifth-order compact-upwind formula, Eq. 2.9, and the fifth-order explicit-upwind formula:

$$\hat{f}_{i+1/2} = \frac{1}{30}f_{i-2} - \frac{13}{60}f_{i-1} + \frac{47}{60}f_i + \frac{9}{20}f_{i+1} - \frac{1}{20}f_{i+2}, \tag{2.19}$$

which can be retrieved from the general formula, Eq. 2.5, by choosing $L_1 = L_2 = 0$, $M_1 = M_2 = 2$ and requiring fifth-order accuracy. The transfer function of an exact differentiation, $Z(\hat{k}) = \hat{k}$, is also shown as a reference. The comparison is interesting for several reasons: 1) it shows that, for a fixed formal accuracy, there is a gain in using a more computationally demanding compact formula over an explicit one; 2) any fifth-order conservative ENO/WENO scheme degenerates to Eq. 2.19 in smooth regions, and so Fig. 2.1 is also a comparison between a compact scheme and the best an ENO/WENO scheme with the same accuracy can achieve; 3) the two formulas Eq. 2.9 and Eq. 2.19 are combined in the multi-block scheme discussed in section 4.1.1, whose properties are intermediate between the compact and the explicit scheme.

Examining Fig. 2.1(a), we see that the explicit formula fails to propagate waves captured with 4 grid points per wavelength ($\hat{k} = 1.5708$) at the right speed. On the other hand, the compact formula is able to propagate waves captured with as few grid points per wavelength as 3 ($\hat{k} = 2.0944$) at the right speed.

Examining Fig. 2.1(b), we see that the compact formula is less dissipative than the explicit one for wavenumbers up to $\hat{k} \approx 2.4$ ($2 < n_p < 3$), while it is more dissipative at higher wave-numbers. The higher dissipation of waves with $\hat{k} > 2.4$ is, in fact, useful for two reasons: first, such waves are propagated at the wrong speed, as emerges from the analysis of Fig. 2.1(a); second, they can represent high-frequency spurious oscillations, and so dissipating them improves accuracy and stability.

This last consideration has a fundamental consequence for compressible fluid dynamics applications. In order to capture discontinuities, these high-order formulas are filtered using a TVD limiter, as is explained in section 2.2; spurious oscillations create local extrema, and are damped by the TVD filter by reducing the local accuracy; it follows that, if the high-order formula is able to suppress spurious high-frequency oscillations, the limiter does not detect local extrema, it is not active and so the accuracy is preserved.

2.1.3 Boundary closures

The development of appropriate boundary closures is crucial in order for a compact scheme to work properly (3), mostly because of the global nature of the reconstruction step, which involves all the nodes simultaneously. Indeed, boundary closures critically impact the stability characteristics of the scheme, and a stability study based on decomposition into Fourier harmonics may lead to wrong conclusions. A stability analysis of the full discretisation therefore becomes necessary. Most problems are not periodic, and so appropriate boundary closures for the formula Eq. 2.9 must be used. Gustafsson (58) showed that using boundary closures $(n-1)^{th}$-order accurate, along with an n^{th}-order accurate scheme for the interior, is sufficient to maintain n^{th}-order accuracy over the whole domain. Pirozzoli (21) proposed the following fourth-order explicit boundary closures:

$$\hat{f}_{1/2} = \frac{1}{4}f_0 + \frac{13}{12}f_1 - \frac{5}{12}f_2 + \frac{1}{12}f_3, \tag{2.20}$$

$$\hat{f}_{N+1/2} = \frac{25}{12}f_N - \frac{23}{12}f_{N-1} + \frac{13}{12}f_{N-2} - \frac{1}{4}f_{N-3}, \tag{2.21}$$

which can be derived from Eq. 2.5, by imposing $\alpha_0 = 1$, fourth-order accuracy, $L_1 = L_2 = 0$, $M_1 = 0$ and $M_2 = 3$ for the intermediate node $i = 1/2$, and $M_1 = 3$ and $M_2 = 0$ for the intermediate node $i = N + 1/2$. In Eq. 2.20, the quantity f_0 is evaluated at node x_0, which is either a boundary point, for a vertex-based discretisation, or a ghost point, for a cell-centred discretisation. In either case, f_0 is used to enforce a physical boundary condition. These boundary closures are therefore consistent with the physical nature of the problem, i.e. the propagation of right-running disturbances: while Eq. 2.20

carries information about the boundary condition at the left boundary, Eq. 2.21 simply extrapolates a value from the interior.

Pirozzoli (21) conducted a linear stability analysis of the discretisation employing Eq. 2.9 for the interior and Eqs. 2.20 and 2.21 near the boundaries. He concluded that the discretisation is linearly stable regardless of the number of grid-points, and also that the boundary closures have a stabilising effect with respect to the periodic case.

2.2 Split-flux upwind-TVD schemes

The *total variation* of a function $u(x,t)$, over a domain $x \in \Omega$, is defined as:

$$TV[u](t) = \int_{\Omega} \left| \frac{\partial u(x,t)}{\partial x} \right| dx, \tag{2.22}$$

or, if the function is given on a set of nodes x_i, for $i = 1, 2, \ldots, N$:

$$TV[u](t) = \sum_{i=1}^{N-1} |\Delta u_{i+1/2}(t)| = \sum_{i=1}^{N-1} |u_{i+1}(t) - u_i(t)|. \tag{2.23}$$

A desirable feature of numerical methods for hyperbolic conservation laws is that the total variation of the computed solution is a non-increasing function of time, i.e.

$$t_2 > t_1 \Rightarrow TV[u](t_2) \leq TV[u](t_1). \tag{2.24}$$

The solution of hyperbolic conservation laws can be discontinuous in space, and, when computed numerically, may contain unphysical oscillations, due to interpolation across discontinuities. *Total Variation Diminishing* (TVD) methods produce solutions free from spurious oscillations even when the solution is discontinuous. Unfortunately, linear TVD methods are only first order accurate (59). High-order TVD methods are non-linear, and are usually constructed using a high-order linear method along with a non-linear flux or slope limiter function. Since Harten (59) devised the first high-order TVD methods, many others have been proposed, the most common of which can be found in any standard CFD book. In this section we introduce two flux limiter functions devised by Tu *et al.* (60). Tu and Yuan (25) showed that combining either of these limiters with the compact linear formula Eq. 2.9 yields a stable and accurate scheme. Therefore, we employ these limiters for our simulations.

Consider the scalar conservation law:

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0, \tag{2.25}$$

whose semi-discrete conservative form, over a grid with uniform spacing Δx, is:

$$\frac{\partial u}{\partial t} + \frac{h_{i+1/2} - h_{i-1/2}}{\Delta x} = 0, \tag{2.26}$$

where $h_{i\pm1/2}$ are values of a TVD numerical flux function.

Unlike the wave equation, Eq. 2.10, where disturbances are known to travel in a certain direction depending on the sign of the constant c, in the more general case of Eq. 2.25 one must distinguish between right-running and left-running disturbances. Two approaches to this problem are Flux Difference Splitting, as in Roe's scheme, and Flux Vector Splitting (FVS), such as the Steger-Warming scheme. We are concerned with FVS techniques, which split the flux into two contributions:

$$f(u) = f^+(u) + f^-(u).$$

For a scalar conservation law positive and negative fluxes can be defined as follows:

$$f^+(u) = \begin{cases} f(u) & \text{if} \quad f'(u) > 0 \\ 0 & \text{if} \quad f'(u) \leq 0 \end{cases}, \tag{2.27}$$

$$f^-(u) = \begin{cases} 0 & \text{if} \quad f'(u) \geq 0 \\ f(u) & \text{if} \quad f'(u) < 0 \end{cases}. \tag{2.28}$$

Given a flux splitting technique to calculate $f^\pm(u)$, the TVD numerical flux function is split accordingly:

$$h_{i+1/2} = h^+_{i+1/2} + h^-_{i+1/2}, \tag{2.29}$$

and computed as a sum of a low-order flux and a high-order correction:

$$h^+_{i+1/2} = f^+_i + \phi^+_{i+1/2}, \qquad h^-_{i+1/2} = f^-_{i+1} - \phi^-_{i+1/2}. \tag{2.30}$$

The high-order correction is computed limiting a high-order numerical flux function. Tu *et al.* (60) proposed two flux limiters:

$$\begin{aligned}
\phi_\mathrm{A}(a,b,c) &= \begin{cases} \operatorname{sgn}(a)\min(|a|,|b|), & \text{if } a,b,c \text{ have the same sign,} \\ 0, & \text{otherwise.} \end{cases} \\
\phi_\mathrm{B}(a,b,c) &= \begin{cases} \operatorname{sgn}(a)\min\left(|a|,|b|,\frac{2bc}{|a|+|c|+\epsilon}\right), & \text{if } a,b,c \text{ have the same sign,} \\ 0, & \text{otherwise,} \end{cases}
\end{aligned}$$

where ϵ is an arbitrary small constant to avoid division by zero. The type A limiter is a version of the well known minmod limiter, while type B is a combination of the latter limiter and a version of the van Leer limiter. The high-order correction is computed from the flux limiter as follows:

$$\begin{aligned} \phi^+_{i+1/2} &= \phi(\Delta\hat{f}^+_{i+1/2}, \Delta f^+_{i+1/2}, \Delta f^+_{i-1/2}), \\ \phi^-_{i+1/2} &= \phi(\Delta\hat{f}^-_{i+1/2}, \Delta f^-_{i+1/2}, \Delta f^-_{i+3/2}). \end{aligned} \tag{2.31}$$

In Eqs. 2.31, ϕ is either ϕ_{A} or ϕ_{B}, and

$$\Delta\hat{f}^+_{i+1/2} = \hat{f}^+_{i+1/2} - f^+_i, \tag{2.32}$$

$$\Delta\hat{f}^-_{i+1/2} = f^-_{i+1} - \hat{f}^-_{i+1/2}, \tag{2.33}$$

$$\Delta f^\pm_{i+1/2} = f^\pm_{i+1} - f^\pm_i, \tag{2.34}$$

where $\hat{f}^\pm_{i+1/2}$ are the positive and negative part of any high-order numerical flux function at the intermediate node $x_{i+1/2}$.

The principle such limiter functions are based on is rather simple. Consider, for example, the positive flux (analogous considerations can be made for the negative flux). First, it is assumed to be a monotonic function of u (Eq. 2.27), and so the behaviour of $f^+(u)$ reflects the behaviour of u. If u does not have a local extremum ($\Delta f^+_{i+1/2}\Delta f^+_{i-1/2} \geq 0$) and the high-order reconstruction does not introduce one ($\Delta\hat{f}^+_{i+1/2}\Delta f^+_{i+1/2} \geq 0$ and $|\Delta\hat{f}^+_{i+1/2}| \leq |\Delta f^+_{i+1/2}|$), then the high-order reconstruction verifies the TVD property, and so it is used ($h^+_{i+1/2} = \hat{f}_{i+1/2}$). Otherwise, a first-order reconstruction that verifies the TVD property (18) is used ($h^+_{i+1/2} = f^+_i$). A rigorous proof that the scheme described in this section is TVD, is given by Tu *et al.* (60) and is not reproduced here.

2.2.1 Solution of one-dimensional scalar conservation laws

Combining the fifth-order compact-upwind scheme described in section 2.1, and the TVD limiters described in section 2.2, to solve the scalar conservation law is straightforward. The positive part of the high-order numerical flux function $\hat{f}^+_{i+1/2}$ is calculated by solving the tri-diagonal system Eq. 2.9 with boundary closures Eqs. 2.20 and 2.21, i.e.

$$\begin{aligned}
\hat{f}^{+}_{1/2} &= \frac{1}{4}f^{+}_{0} + \frac{13}{12}f^{+}_{1} - \frac{5}{12}f^{+}_{2} + \frac{1}{12}f^{+}_{3}, && (2.35)\\
9\hat{f}^{+}_{i-1/2} + 18\hat{f}^{+}_{i+1/2} + 3\hat{f}^{+}_{i+3/2} &= f^{+}_{i-1} + 19f^{+}_{i} + 10f^{+}_{i+1}, \quad i = 1, \cdots, N, && (2.36)\\
\hat{f}^{+}_{N+1/2} &= \frac{25}{12}f^{+}_{N} - \frac{23}{12}f^{+}_{N-1} + \frac{13}{12}f^{+}_{N-2} - \frac{1}{4}f^{+}_{N-3}. && (2.37)
\end{aligned}$$

The scheme to compute the negative part of the high order numerical flux function $\hat{f}^{-}_{i+1/2}$ can be obtained from the scheme above by symmetry considerations, and reads:

$$\begin{aligned}
\hat{f}^{-}_{1/2} &= \frac{25}{12}f^{-}_{1} - \frac{23}{12}f^{-}_{2} + \frac{13}{12}f^{-}_{3} - \frac{1}{4}f^{-}_{4}, && (2.38)\\
3\hat{f}^{-}_{i-1/2} + 18\hat{f}^{-}_{i+1/2} + 9\hat{f}^{-}_{i+3/2} &= 10f^{-}_{i} + 19f^{-}_{i+1} + f^{-}_{i+2}, \quad i = 1, \cdots, N, && (2.39)\\
\hat{f}^{-}_{N+1/2} &= \frac{1}{4}f^{-}_{N+1} + \frac{13}{12}f^{-}_{N} - \frac{5}{12}f^{-}_{N-1} + \frac{1}{12}f^{-}_{N-2}. && (2.40)
\end{aligned}$$

The correction to the low-order numerical flux $\phi^{\pm}_{i+1/2}$ is then calculated according to Eq. 2.31. The high order limited numerical flux function $h_{i+1/2}$ is finally computed by Eqs. 2.29 and 2.30.

2.3 Solution of the Euler equations

In this section we describe the solution of the Euler equations of gas dynamics using the compact-TVD method developed by Tu and Yuan (25). In a Cartesian reference frame (x, y, z), the Euler equations read:

$$\frac{\partial Q}{\partial t} + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial y} + \frac{\partial G}{\partial z} = 0, \qquad (2.41)$$

where

$$\begin{aligned}
Q &= \left\{ \begin{array}{c} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{array} \right\}, & E &= \left\{ \begin{array}{c} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho uH \end{array} \right\},\\
F &= \left\{ \begin{array}{c} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ \rho vH \end{array} \right\}, & G &= \left\{ \begin{array}{c} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ \rho wH \end{array} \right\}.
\end{aligned} \qquad (2.42)$$

Here, ρ is the mass density, (u, v, w) the components of the velocity vector V along (x, y, z), respectively, p the pressure, H the total enthalpy, $H = a^2/(\gamma - 1) + V^2/2$,

$a = \sqrt{\gamma p/\rho}$ is the speed of sound, and γ the adiabatic index. In order to solve Eq. 2.41 on domains of arbitrary shape using FD methods, it is expressed in general coordinates, (ξ, η, ζ),

$$\frac{\partial \tilde{Q}}{\partial t} + \frac{\partial \tilde{E}}{\partial \xi} + \frac{\partial \tilde{F}}{\partial \eta} + \frac{\partial \tilde{G}}{\partial \zeta} = 0, \tag{2.43}$$

where:

$$\tilde{Q} = \frac{Q}{J}, \qquad \tilde{E} = \frac{\xi_x E + \xi_y F + \xi_z G}{J}, \tag{2.44}$$

$$\tilde{F} = \frac{\eta_x E + \eta_y F + \eta_z G}{J}, \qquad \tilde{G} = \frac{\zeta_x E + \zeta_y F + \zeta_z G}{J}. \tag{2.45}$$

The metrics ξ_x, ξ_y, ξ_z, η_x, η_y, η_z, ζ_x, ζ_y, ζ_z define a transformation from an arbitrarily-shaped physical domain to a cubic domain, and J is the Jacobian of the transformation. In order to use FD formulas on equally spaced grids, this transformation is generally chosen such that the computational grid in physical space maps onto a uniform grid in computational space.

Using the method of lines to solve Eq. 2.43, the problem is split into three one-dimensional problems. We describe how the problem is solved for direction ξ, and the same algorithm is applied along η and ζ. The 1D problem along ξ is:

$$\frac{\partial \tilde{Q}}{\partial t} + \frac{\partial \tilde{E}}{\partial \xi} = 0, \tag{2.46}$$

whose semi-discrete conservative form is:

$$\frac{\partial \tilde{Q}_i}{\partial t} + \frac{\mathcal{E}_{i+1/2} - \mathcal{E}_{i-1/2}}{\Delta \xi} = 0. \tag{2.47}$$

As in the scalar one-dimensional example, the flux is split into its positive and negative parts, computed as the sum of a first-order term and a high-order correction:

$$\mathcal{E}^+_{i+1/2} = \tilde{E}^+_i + \phi^+_{i+1/2}, \qquad \mathcal{E}^-_{i+1/2} = \tilde{E}^-_{i+1} - \phi^-_{i+1/2}.$$

The core of the method is the estimation of the high-order correction, $\phi^{\pm}_{i+1/2}$, which must yield the most accurate TVD numerical flux reconstruction. The computation consists of two steps:

- a linear procedure for the estimation of the high-order numerical flux function;
- a non-linear procedure to filter the high-order numerical flux and make it TVD.

The linear procedure is essentially the same as for the scalar case: positive and negative parts of the high-order numerical flux function, $\hat{E}^{\pm}_{i+1/2}$, are reconstructed component-by-component by solving the linear systems given by Eqs. 2.35–2.37 and Eqs. 2.38–2.40, respectively. The non-linear procedure, instead, needs to be modified. The limiter introduced in section 2.2, like most limiters, is based on the assumption that the behaviour of the computed solution only depends on the reconstructed behaviour of the corresponding flux. In a system of conservation laws, each component of the vector of the conserved variables $\tilde{Q}$ is coupled to the others via the corresponding part of the flux $\tilde{E}(\tilde{Q})$, and so application component-by-component of the limiter may not be enough to enforce the TVD property of the computed solution. For this reason, a characteristic decomposition of the Euler equations is used for the non-linear part of the method, as detailed in the following.

2.3.1 Characteristic decomposition

The inviscid flux is a linearly homogeneous function of the conserved variables, and so the Euler equations can be written in the following non-conservative form:

$$\frac{\partial \tilde{Q}}{\partial t} + \tilde{A}\frac{\partial \tilde{Q}}{\partial \xi} = 0. \tag{2.48}$$

In Eq. 2.48, the flux Jacobian matrix is introduced:

$$\tilde{A} = \frac{d\tilde{E}}{d\tilde{Q}} = |\nabla\xi|\frac{d\check{E}}{dQ} = |\nabla\xi|A, \tag{2.49}$$

where $|\nabla\xi| = \sqrt{\xi_x^2 + \xi_y^2 + \xi_z^2}$ is the L^2-norm of the ξ-metrics vector, and

$$\check{E} = \frac{J}{|\nabla\xi|}\tilde{E} \tag{2.50}$$

is the inviscid flux per unit-area through a surface normal to the local ξ-direction. The flux Jacobian matrix A has five real eigenvalues, not all distinct. Defining the unit-vector parallel to the local ξ-direction, $(n_x, n_y, n_z) = \nabla\xi/|\nabla\xi|$, and the velocity component along the same direction, $V_n = un_x + vn_y + wn_z$, the eigenvalues are: $\lambda_1 = V_n - a$, $\lambda_2 = \lambda_4 = \lambda_5 = V_n$ and $\lambda_3 = V_n + a$. Also, A has a full set of linearly independent eigenvectors, and so it can be decomposed as follows:

$$A = R\Lambda L,$$

where Λ is a diagonal matrix with the eigenvalues of A on the diagonal, R and L are the right and left eigenvector matrices, respectively. If the differential form $Ld\tilde{Q}$ has a potential W, such that $dW = Ld\tilde{Q}$, then an alternative form of Eq. 2.48 is:

$$\frac{\partial W}{\partial t} + |\nabla\xi|\Lambda\frac{\partial W}{\partial \xi} = 0, \tag{2.51}$$

and W is the vector of the characteristic variables.

Characteristic variables evolve independently, as Λ is a diagonal matrix, and so a component-wise application of the method may be suitable for Eq. 2.51. For the Euler equations, characteristic variables exist only if the entropy is conserved along a stream line, i.e. the entropy conservation law can be considered in lieu of the total energy conservation law. Therefore, characteristic decomposition is allowed but it is incorrect across surfaces where entropy is produced, such as shock waves. The method under study employs characteristic decomposition in its non-linear part, as detailed in the following.

The flux increments are projected along characteristic directions:

$$\begin{array}{rclrcl} \Delta\hat{W}^{+}_{i+1/2} & = & L^{l}_{i+1/2}(\hat{E}^{+}_{i+1/2} - \tilde{E}^{+}_{i}), & \Delta W^{+}_{i+1/2} & = & L^{l}_{i+1/2}(\tilde{E}^{+}_{i+1} - \tilde{E}^{+}_{i}), \\ \Delta\hat{W}^{-}_{i+1/2} & = & L^{r}_{i+1/2}(\tilde{E}^{-}_{i+1} - \hat{E}^{-}_{i+1/2}), & \Delta W^{-}_{i+1/2} & = & L^{r}_{i+1/2}(\tilde{E}^{-}_{i+1} - \tilde{E}^{-}_{i}). \end{array} \tag{2.52}$$

Different superscripts are used for the left eigenvector matrices, $L^{l}_{i+1/2}$ and $L^{r}_{i+1/2}$: characteristic directions are not uniform in space, so one must choose where eigenvectors should be evaluated. Different possibilities are available, as explained in section 2.3.2. Quantities $\Delta\hat{W}^{\pm}_{i+1/2}$ and $\Delta W^{\pm}_{i+1/2}$ are, respectively, high and low-order upwind evaluations of the characteristic flux increments. Since they are expected to be independent, they can be limited according to Eq. 2.31:

$$\delta W^{+}_{i+1/2} = \phi(\Delta\hat{W}^{+}_{i+1/2}, \Delta W^{+}_{i+1/2}, \Delta W^{+}_{i-1/2}), \tag{2.53}$$

$$\delta W^{-}_{i+1/2} = \phi(\Delta\hat{W}^{-}_{i+1/2}, \Delta W^{-}_{i+1/2}, \Delta W^{-}_{i+3/2}). \tag{2.54}$$

Finally, the correction $\phi^{\pm}_{i+1/2}$ is calculated projecting the limited increment of the characteristic flux back into the conserved variable space:

$$\phi^{+}_{i+1/2} = R^{l}_{i+1/2}\delta W^{+}_{i+1/2}, \qquad \phi^{-}_{i+1/2} = R^{r}_{i+1/2}\delta W^{-}_{i+1/2},$$

where $R^{l}_{i+1/2}$ and $R^{r}_{i+1/2}$ are right eigenvector matrices, whose evaluation is consistent with the left eigenvector matrices.

2.3.2 Variants of the method

Several variants of the method described in section 2.3.1 can be generated by changing some of the basic techniques it relies on. These techniques are:

- **TVD filter**. As seen in section 2.2, two limiters, namely type A and B, may be used. Type A is expected to be more accurate while type B is expected to be more stable.

- **Characteristic decomposition**. In general, the characteristic decomposition is not uniform in space, so one must choose a state to evaluate the eigenvectors. Referring to Eqs. 2.52, two popular choices are:
  - Upwind evaluation, i.e. $L^l_{i+1/2} = L(Q_i)$ and $L^r_{i+1/2} = L(Q_{i+1})$.
  - Evaluation at an average state $Q_{i+1/2}$ between Q_i and Q_{i+1}, i.e. $L^l_{i+1/2} = L^r_{i+1/2} = L(Q_{i+1/2})$. A popular choice is the Roe average state, defined as:

$$\begin{aligned} \rho_{i+1/2} &= \sqrt{\rho_i \rho_{i+1}}, \\ \Phi_{i+1/2} &= \frac{\sqrt{\rho_i}\Phi_i + \sqrt{\rho_{i+1}}\Phi_{i+1}}{\sqrt{\rho_i}+\sqrt{\rho_{i+1}}}, \quad \Phi = u, v, w, H. \end{aligned} \tag{2.55}$$

  The upwind evaluation is expected to be more stable, while the evaluation based on the Roe average state is expected to be more accurate.

- **Flux Vector Splitting (FVS)**. Several FVS techniques are available in the literature. We have considered classic splittings, namely Steger-Warming and van Leer, and a splitting based on the kinetic theory of gases, namely the Kinetic Splitting.

  The Steger-Warming split flux can be expressed as:

$$\check{E}^{\pm} = R\Lambda^{\pm}LQ, \qquad \Lambda^{\pm} = \frac{\Lambda \pm \Lambda}{2}. \tag{2.56}$$

  This separates left and right-running waves based on the eigenvalues, and it is not differentiable at sonic points.

The van Leer split flux can be written as:

$$\begin{array}{rcll}
\check{E}^+ & = & \check{E} & \text{if} \quad M_n > 1 \\
\check{E}^+ & = & \check{E}^+_{mass} \left\{ \begin{array}{c} 1 \\ u + \frac{a}{\gamma}(2 - M_n)n_x \\ v + \frac{a}{\gamma}(2 - M_n)n_y \\ w + \frac{a}{\gamma}(2 - M_n)n_z \\ e_k + 2a^2 \frac{1+(\gamma-1)M_n - \frac{\gamma-1}{2}M_n^2}{\gamma^2-1} \end{array} \right\} & \text{if} \quad 0 < M_n < 1, \\
\check{E}^+ & = & \check{E} - \check{E}^- & \text{if} \quad M_n < 0,
\end{array} \tag{2.57}$$

and

$$\begin{array}{rcll}
\check{E}^- & = & \check{E} - \check{E}^+ & \text{if} \quad M_n > 0, \\
\check{E}^- & = & \check{E}^-_{mass} \left\{ \begin{array}{c} 1 \\ u + \frac{a}{\gamma}(-2 - M_n)n_x \\ v + \frac{a}{\gamma}(-2 - M_n)n_y \\ w + \frac{a}{\gamma}(-2 - M_n)n_z \\ e_k + 2a^2 \frac{1-(\gamma-1)M_n - \frac{\gamma-1}{2}M_n^2}{\gamma^2-1} \end{array} \right\} & \text{if} \quad 0 < M_n < 1, \\
\check{E}^- & = & \check{E} & \text{if} \quad M_n < -1,
\end{array} \tag{2.58}$$

where:

$$M_n = V_n/a, \quad \check{E}^{\pm}_{mass} = \pm\frac{1}{4}\rho a(M_n \pm 1)^2, \quad e_k = \frac{u^2 + v^2 + w^2}{2}.$$

The van Leer FVS is known to be smoother than the Steger-Warming FVS, being differentiable over the whole range of Mach numbers.

The Kinetic FVS is a less common technique, where left and right-running disturbances are computed from microscopic considerations. Some details about how the fluxes are derived are given in section 3.2. Here we give the expression for the inviscid kinetic split flux. Parameters for the splitting are the *peculiar velocity* $c = \sqrt{2p/\rho}$ (61), the normal speed ratio $S_n = V_n/c$, the coefficients $p^{\pm} = 1/2[1 \pm \text{erf}(S_n)]$ and $\epsilon = \exp\left(-S_n^2\right)/2\sqrt{\pi}$. Having defined these parameters, the kinetic split flux is:

$$\check{E}^{\pm} = \left\{ \begin{array}{c} \check{E}^{\pm}_{mass} \\ \check{E}^{\pm}_{mass}u + n_x p p^{\pm} \\ \check{E}^{\pm}_{mass}v + n_y p p^{\pm} \\ \check{E}^{\pm}_{mass}w + n_z p p^{\pm} \\ \check{E}^{\pm}_{mass}H \mp pc\epsilon/2 \end{array} \right\}, \tag{2.59}$$

where $\check{E}^{\pm}_{mass} = \rho(V_n p^{\pm} \pm c\epsilon)$ is the kinetic split mass flux. It is the most accurate but also the most computationally expensive, because it requires the evaluation of error and exponential functions.

2.3.3 A modification to the characteristic decomposition

In section 2.3.2 we presented several variants of the original method proposed by Tu and Yuan. We anticipate here results discussed in section 5.1.1, i.e. that the characteristic decomposition is essential to suppress spurious oscillations, but changing the limiter, the evaluation of the eigenvector matrices or the flux splitting has a rather marginal impact on the computed solution. On the other hand, in several simulations we found essential a modification we introduce in the limiting step, and this new feature is discussed in this section.

In the original method, the limited characteristic flux is computed by Eqs. 2.53 and 2.54. For example, the limiting equation for the positive flux is:

$$\delta W^+_{i+1/2} = \phi(\Delta \hat{W}^+_{i+1/2}, \Delta W^+_{i+1/2}, \Delta W^+_{i-1/2}),$$

where:

$$\begin{aligned}
\Delta \hat{W}^+_{i+1/2} &= L^l_{i+1/2}(\hat{E}^+_{i+1/2} - \tilde{E}^+_i),\\
\Delta W^+_{i+1/2} &= L^l_{i+1/2}(\tilde{E}^+_{i+1} - \tilde{E}^+_i),\\
\Delta W^+_{i-1/2} &= L^l_{i-1/2}(\tilde{E}^+_i - \tilde{E}^+_{i-1}).
\end{aligned}$$

We note that the quantity $\Delta W^+_{i-1/2}$ is based on a characteristic decomposition that is, in general, different from the one $\Delta \hat{W}^+_{i+1/2}$ and $\Delta W^+_{i+1/2}$ are based on. The characteristic decomposition is meaningful only at a grid node, because, for example, the entropy may not be conserved across grid nodes. Therefore, we propose to use the same characteristic decomposition for all of the characteristic flux increments involved in the limiting, and re-define $\Delta W^+_{i-1/2}$ as:

$$\Delta W^+_{i-1/2} = L^l_{i+1/2}(\tilde{E}^+_i - \tilde{E}^+_{i-1}).$$

Analogously, for the negative flux we compute:

$$\Delta W^-_{i+3/2} = L^r_{i+1/2}(\tilde{E}^-_{i+2} - \tilde{E}^-_{i+1}),$$

unlike the original method, where:

$$\Delta W^-_{i+3/2} = L^r_{i+3/2}(\tilde{E}^-_{i+2} - \tilde{E}^-_{i+1}).$$

This modification may seem marginal but, in fact, it has proven essential when the domain has boundaries with regions of very high curvature, such as corners. Without our modification, compression corners generate high numerical noise and, at hypersonic Mach numbers, negative pressure values upstream from the corner. Expansion corners are even more problematic, as they generate negative pressure values at moderatly high (supersonic) Mach numbers. Our modification, on the other hand, allows oscillation-free and stable simulation of corner flows. An example of how this modification improves stability and accuracy of the method is presented in section 6.1.2.

2.4 Time discretisation

The method described in section 2.3 provides a discretisation for the spatial operator $\mathscr{L}$ in a system of conservation laws:

$$\frac{\partial Q}{\partial t} = \mathscr{L}(Q). \tag{2.60}$$

The numerical model is fully defined once an approximation for the time derivative is specified. For time-accurate solutions a common choice is an explicit Runge-Kutta scheme.

The state vector Q at time t^n, $Q^n = Q(t^n)$, is known, and its value at time $t^{n+1} = t^n + \Delta t$, $Q^{n+1} = Q(t^{n+1})$, needs to be computed. In an m-stage explicit Runge-Kutta scheme, this is accomplished by calculating m intermediate states $Q^{(i)}$ between Q^n and Q^{n+1}, according to

$$Q^{(i)} = Q^{(0)} + \Delta t \sum_{k=0}^{i-1} c_{i,k} \mathscr{L}(Q^{(k)}), \qquad i = 1, 2, \ldots, m, \tag{2.61}$$

where $Q^{(0)} = Q^n$ and $Q^{(m)} = Q^{n+1}$ and $c_{i,k}$ are the Runge-Kutta coefficients. Runge-Kutta schemes can achieve high formal accuracy in time. However, there are two specific drawbacks:

- the number of operations per time step increases as well as the storage requirement, because m intermediate states need to be computed;
- if $\mathscr{L}$ is a TVD spatial operator and the numerical solution is required to be TVD, then the maximum allowed time step for stability decreases as the time accuracy of the scheme increases.

Shu and Osher (19) devised a class of Runge-Kutta schemes, which, if $\mathscr{L}$ is a TVD spatial operator, compute TVD solutions and, for given time-accuracy, have the least restrictive stability condition. Within this family of schemes, a popular choice is the third-order one:

$$\begin{aligned}
&Q^{(0)} = Q^n,\\
&Q^{(1)} = Q^{(0)} + \Delta t \mathscr{L}(Q^0),\\
&Q^{(2)} = Q^{(0)} + \Delta t \tfrac{1}{4}[\mathscr{L}(Q^{(0)}) + \mathscr{L}(Q^{(1)})],\\
&Q^{(3)} = Q^{(0)} + \Delta t \tfrac{1}{6}[\mathscr{L}(Q^{(0)}) + \mathscr{L}(Q^{(1)})] + \tfrac{2}{3}\mathscr{L}(Q^{(2)}),\\
&Q^{n+1} = Q^{(3)}.
\end{aligned} \tag{2.62}$$

This scheme has the same stability requirement as the first-order scheme, therefore no reduction of the maximum allowed time step is introduced by increasing the time accuracy; furthermore, intermediate states (0) and (1) share the same Runge-Kutta coefficients ($c_{2,0} = c_{2,1} = 1/4$ and $c_{3,0} = c_{3,1} = 1/6$), so one can set a single array to store the quantity $[\mathscr{L}(Q^{(0)}) + \mathscr{L}(Q^{(1)})]$.

2.5 Implementation and boundary conditions

The method described in this chapter has been implemented in two new codes. The first code, which we call CU5-TVD-NSK, is written in C++ language. It employs a FD cell-centred discretisation, i.e. the computational domain is divided into cells and the nodes for the FD discretisation are placed at the cell-centres. Indeed, this type of discretisation is less common than the vertex-based one in FD codes, and we have chosen it because it has an inherent advantage for the solution method studied when multi-block computational grids are considered, as explained in section 4.1.1. We have developed a parallel version of CU5-TVD-NSK, which we call CU5-TVD-NSK-MB, employing dual-level parallelism. This topic is discussed in section 4.1

The second code, which we call CUDA-CU5-TVD-NS, is written in the C language. It employs a FD vertex-based discretisation, i.e. the computational domain is divided into cells and the nodes for the FD discretisation are the vertices of the cells. The reason for having two implementations of the same method is that the code developed first, CU5-TVD-NSK, has a structure unsuitable for GPU computing, while the second, CUDA-CU5-TVD-NS, was developed specifically for GPUs. This topic is discussed in section 4.2.

For the Euler solver, the only difference between the two codes is the placement of the FD nodes (cell-centres versus cell-corners), but the different placement of the FD nodes impacts the way the boundary conditions are enforced. For the Navier-Stokes solver, the two codes also differ in the way viscous effects are accounted for, as explained in section 3.3.2.

2.5.1 Boundary conditions based on extrapolation

In CU5-TVD-NSK, the boundary conditions are enforced providing the computational domain with ghost cells, where fluid dynamic variables are either imposed by the boundary conditions or extrapolated from the interior. Consider, for example, a ξ-constant surface, where n denotes the ξ-parallel unit vector pointing into the computational domain, the ghost node is identified by the index $i = 0$ and the first interior node by the index $i = 1$. For the solution of the Euler equations, the following boundary conditions have been implemented:

- **Supersonic inflow**: inflow primitive variables, $(\rho_{in}, u_{in}, v_{in}, w_{in}, p_{in})$, provided as an input, are imposed at the ghost node.
- **Subsonic inflow**: inflow density and velocity are imposed at the ghost node, while the pressure is linearly extrapolated from the interior.
- **Supersonic outflow**: primitive variables at the ghost node are linearly extrapolated from the interior.
- **Subsonic outflow**: density and velocity at the ghost node are linearly extrapolated from the interior, while the pressure is set to its outflow value p_{out}, provided as an input.
- **Slip-wall**: there is no mass flux through the boundary surface, and derivatives of pressure, density and tangential velocity in the direction normal to the surface are zero. This is easily imposed by assigning the following values to the primitive variables at the ghost node:

$$\begin{aligned} \rho_0 &= \rho_1, \\ u_0 &= u_1 - 2V_{n,1}n_x, \\ v_0 &= v_1 - 2V_{n,1}n_y, \\ w_0 &= w_1 - 2V_{n,1}n_z, \\ p_0 &= p_1. \end{aligned} \tag{2.63}$$

2.5.2 Characteristic boundary conditions

We have provided our code CUDA-CU5-TVD-NS with Characteristic Boundary Conditions (CBC), because these are more accurate than those based on extrapolation, and more suitable for future potential DNS applications. For the theory of CBC we refer to the fundamental works by Thompson (62) and Poinsot and Lele (63), and to the recent work by Landmann *et al.* (64).

In the CBC approach, the governing equations are solved at the boundaries. Consider, for example, the boundary $\xi = 0$. The governing equations read:

$$\frac{\partial Q}{\partial t} + |\nabla\xi| \frac{\partial \check{E}}{\partial \xi} = S, \tag{2.64}$$

where

$$S = J \left[\check{E} \frac{\partial \left(|\nabla\xi|/J \right)}{\partial \xi} + \frac{\partial \tilde{F}}{\partial \eta} + \frac{\partial \tilde{G}}{\partial \zeta} \right]. \tag{2.65}$$

The variables appearing in Eqs. 2.64 and 2.65 have the same meaning as in section 2.3. Projecting Eq. 2.64 into characteristic directions, it becomes

$$\frac{\partial W}{\partial t} + |\nabla\xi| \Lambda \frac{\partial W}{\partial \xi} = S_W, \tag{2.66}$$

where

$$\frac{\partial W}{\partial t} = L \frac{\partial Q}{\partial t}, \qquad \Lambda \frac{\partial W}{\partial \xi} = L \frac{\partial \check{E}}{\partial \xi}, \qquad S_W = LS$$

and

$$\Lambda = \begin{bmatrix} V_n - a & 0 & 0 & 0 & 0 \\ 0 & V_n & 0 & 0 & 0 \\ 0 & 0 & V_n + a & 0 & 0 \\ 0 & 0 & 0 & V_n & 0 \\ 0 & 0 & 0 & 0 & V_n \end{bmatrix}.$$

In Eq. 2.66, $\partial W/\partial t$ is the unknown, S_W is computed in the same fashion as for internal nodes, and the wave amplitude, $\Lambda \partial W/\partial \xi$, is computed differently depending on the sign of the eigenvalues. More precisely, incoming waves, identified by $\lambda > 0$, are used to enforce physical boundary conditions, outgoing waves, identified by $\lambda < 0$, are computed from the interior, and steady waves, identified by $\lambda = 0$, have obviously zero amplitude.

The amplitude of outgoing waves is computed as follows. A fourth-order accurate amplitude is computed (65):

$$\left(\Lambda \partial_\xi \hat{W} \right)_0 = \frac{1}{\Delta\xi} L \left(-\frac{25}{12} \check{E}_0 + 4\check{E}_1 - 3\check{E}_2 + \frac{4}{3} \check{E}_3 - \frac{1}{4} \check{E}_4 \right),$$

as well as the first-order accurate amplitudes:

$$(\Lambda \partial_\xi W)_i = \frac{1}{\Delta \xi} L(\check{E}_{i+1} - \check{E}_i), \qquad i = 0, 1.$$

The TVD property is then enforced by applying the minmod limiter:

$$\Lambda \frac{\partial W}{\partial \xi} = \text{minmod}\left(\left(\Lambda \partial_\xi \hat{W} \right)_0, (\Lambda \partial_\xi W)_0, (\Lambda \partial_\xi W)_1 \right),$$

where:

$$\text{minmod}(a, b, c) = \begin{cases} a & \text{if } ab > 0,\ bc > 0 \text{ and } |a| < |b|, \\ b & \text{otherwise.} \end{cases}$$

Two types of boundaries have been implemented:

- **Farfield**. The physical boundary condition to be enforced is zero amplitude of incoming waves at any time. The characteristic formulation Eq. 2.66 is particularly amenable for this purpose: the wave amplitude is either set to zero or computed from the interior depending on the sign of the corresponding eigenvalue.

- **Slip-wall**. The physical boundary condition to be enforced is zero mass flow across the boundary at any time. Initialising the flow at the boundary so that:

  $$Q_2 n_x + Q_3 n_y + Q_4 n_z = 0,$$

  the condition:

  $$\frac{\partial}{\partial t}(Q_2 n_x + Q_3 n_y + Q_4 n_z) = 0 \tag{2.67}$$

  must be verified at any time. This equation is written in terms of the characteristic variables using the relation $\partial_t Q = R \partial_t W$, and reads:

  $$\sum_{i=1}^{5} (n_x R_{2,i} + n_y R_{3,i} + n_z R_{4,i}) \frac{\partial W_i}{\partial t} = 0. \tag{2.68}$$

  The amplitude of the first (outgoing) wave is computed from the interior. Second, fourth and fifth waves have zero amplitude, because $\lambda_2 = \lambda_4 = \lambda_5 = V_n = 0$. Finally, the amplitude of the third (incoming) wave is computed from Eq. 2.68.

Once the wave amplitudes have been computed according to the boundary type, the time derivative of the characteristic variables vector is computed:

$$\frac{\partial W}{\partial t} = S_W - \Lambda \frac{\partial W}{\partial \xi},$$

and, finally, the time derivative of the conserved variable vector:

$$\frac{\partial Q}{\partial t} = R \frac{\partial W}{\partial t}.$$

2.6 Summary

In this chapter we have presented the compact-TVD method for the Euler equations that we use as a starting point for our study. We have considered several variants to the original method proposed by Tu and Yuan (25) and proposed to use the high accuracy Kinetic FVS instead of the classic Steger-Warming. Moreover, we have introduced a modification to the characteristic treatment employed in the TVD filtering step. The impact of this modification will be shown in section 6.1.2.

3

Solution of the Navier-Stokes equations

In this chapter we present two methods for solving the Navier-Stokes equations based on the compact-TVD method presented in Chapter 2. One method employs the classical operator splitting, and is implemented in the code CUDA-CU5-TVD-NS. The other, implemented in the code CU5-TVD-NSK, employs the kinetic flux vector splitting for the Navier-Stokes flux, proposed by Chou and Baganoff (56). This method is interesting because it is more stable and has lower computational cost than a method differentiating the inviscid flux with the same compact-TVD method, while using a central operator to differentiate the viscous flux. Even more important, a method employing this technique is ideal for future extension to non-equilibrium flows, either interfacing the CFD solver with a particle method, or including non-equilibrium effects into the flux. In the first case, the CFD solver provides an exact flux boundary condition for the particle method, including the viscous component. In the second case, because any non-equilibrium split flux degenerates to the kinetic split flux where equilibrium is reached, consistency between equilibrium and non-equilibrium regions of the flow field is automatically guaranteed.

In section 3.1 we use a simplified model to explain how the methods differ, outlining their advantages and drawbacks. In section 3.2 we present in detail the Kinetic Splitting as derived by Chou and Baganoff (56), identifying its merits and approximations. Finally, in section 3.3 we describe in detail our Navier-Stokes solvers.

3.1 Inclusion of viscous effects

When solving the compressible Navier-Stokes equations using a compact finite-difference method, a shock-capturing scheme can be used to discretise the inviscid flux, i.e. the Euler flux, while the viscous flux can be discretised separately by means of a central difference operator. In order to explain as simply and clearly as possible the different ways to include viscous effects, we shall consider the following one-dimensional scalar conservation law as a model for the Navier-Stokes equations:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(f(u) + k(u)\frac{\partial u}{\partial x}\right) = 0. \tag{3.1}$$

A stable semi-discretisation of Eq. 3.1 is:

$$\frac{\partial u}{\partial t} + D_{up}^{\mathrm{TVD}} f - DkDu - kD_2 u = 0, \tag{3.2}$$

where D_{up}^{TVD} is the operator representing the compact-upwind TVD scheme described in section 2.2, and D and D_2 are compact-central operators for the first and second derivatives, respectively. In Eq. 3.2 the derivative of the viscous flux has been expanded via the chain rule, as suggested by Lele (3), to improve numerical stability.

Expanding the derivative of the viscous flux leads to very complicated expressions when the Navier-Stokes equations in general coordinates are considered. For this reason many authors, see for example Zhong (66), have discretised the viscous flux by applying twice a central operator for the first derivative, along with an upwind operator for the inviscid flux. This leads to the following semi-discrete form of Eq. 3.1:

$$\frac{\partial u}{\partial t} + D_{up}^{\mathrm{TVD}} f - D(kDu) = 0. \tag{3.3}$$

This is the approach used in our code CUDA-CU5-TVD-NS. A third option for discretising Eq. 3.1 is

$$\frac{\partial u}{\partial t} + D_{up}^{\mathrm{TVD}}(f - kDu) = 0. \tag{3.4}$$

This is the approach used in our code CU5-TVD-NSK. Such a discretisation is possible as long as a splitting technique for the total flux $f_{\mathrm{tot}} = f - ku_x$ is available. To the best of our knowledge, this has never before been investigated. The advantages of the latter discretisation over conventional ones are:

- It is computationally less expensive, since three linear systems are solved to compute derivatives, while five and four linear systems must be inverted when solving Eqs. 3.2 and 3.3, respectively. This computational advantage is even more remarkable when multi-dimensional problems and the Navier-Stokes equations are considered.
- It is fully conservative.
- The high-order compact reconstruction is applied to the total flux, which is very likely to be smoother than the inviscid flux, and so fewer spurious oscillations are expected.
- Since the total flux incorporates the physical viscosity and is better reconstructed, limiting such a flux is less prone to over-damping.

The major drawback of a method based on a discretisation like that of Eq. 3.4 is due to the upwinding of the viscous flux, which does not mimic the physical non-directional nature of this flux. This may possibly lead to instability for some diffusion-dominated flows and so requires careful testing.

As mentioned above, Eq. 3.4 relies on the availability of a splitting technique for the total flux. The Kinetic Flux Vector Splitting proposed by Chou and Baganoff (56), and discussed in section 3.2, is suitable for flux splitting the Navier-Stokes equations. The splitting was used by Chou and Baganoff in the framework of a finite-volume approach, and was made second-order via the *Monotone Upstream-centred Schemes for Conservation Laws* (MUSCL) approximation.

An attempt to combine the robustness and physical accuracy of kinetic splitting with the high formal accuracy and resolution properties of compact schemes can be found in the work by Ravichandran (24). The author employed a kinetic splitting for the inviscid flux, like the one in section 2.3.2, along with a compact-TVD scheme to solve the Euler equations, and the test cases clearly showed the higher accuracy with respect to a second-order MUSCL scheme employing the same splitting.

A conceptual problem arises when the Navier-Stokes equations are considered. The shock-capturing method proposed by Tu and Yuan (25) is characteristic-based, as the Euler flux is projected into characteristic space, and then limited. This is an important step for the scheme to be TVD, as it is demonstrated in section 5.1.1, and should not be removed. On the other hand, projecting the Navier-Stokes flux into characteristic space seems to be incorrect, because the characteristic curves can only be defined for the inviscid flux. However, we argue that: (i) in diffusion-dominated regions of the flow field, where the viscous flux plays an important role and the Euler flow approximation is no longer valid, no spurious oscillations occur, and so neither the limiter nor the projection into characteristic space affect the flux reconstruction; (ii) in convection-dominated flow regions, such as across a shock wave, where the solution is affected by the limiting action, the Euler approximation is fully recovered, and the projection into the characteristic space is a meaningful operation.

3.2 Kinetic Flux Vector Splitting for the Navier-Stokes equations

In this section the splitting technique for the Navier-Stokes equations, devised by Chou and Baganoff (56), is presented. This technique is based on kinetic theory and was meant to be applied in hybrid continuum-particle solvers, i.e. methods that solve the Navier-Stokes equations in flows where the continuum hypothesis holds, while using the Direct Simulation Monte Carlo (DSMC) method to compute regions of rarefied flow. The flux splitting was devised to provide one-side fluxes from the continuum region at the interface between regions where different methods are used.

We first describe briefly how conservation laws of gas dynamics are derived as moments of the Boltzmann equation, giving the kinetic expressions of the fluxes; then introduce the concept of kinetic splitting, valid for any velocity distribution function; finally we give the expression of the split flux when a Chapman-Enskog expansion of the distribution function is assumed, i.e. when the Navier-Stokes equations are the governing equations.

3.2.1 Moments of the Boltzmann equation

In this section some basic concepts of kinetic theory are presented. They can be found in many works on kinetic theory, such as by Grad (67), Chapman and Cowling (61) and Bird (68).

Consider an ideal monatomic gas in the absence of external forces, and assume the gas is sufficiently dilute for binary collisions to dominate. In this case the Boltzmann equation, which governs the gas dynamics, is:

$$\frac{\partial(nf)}{\partial t} + c_k \frac{\partial(nf)}{\partial x_k} = \left[\frac{\partial(nf)}{\partial t}\right]_{\text{coll}}, \tag{3.5}$$

where n is the number density, defined as the number of molecules per unit volume, f is the velocity distribution function, c_k is the component along direction x_k of the molecular velocity, the repeated index k denotes a sum, and the right hand side represents the collision integral. Moment equations are obtained by multiplying the Boltzmann equation by a function of the molecular velocity $Q(c_i)$ and integrating over the velocity space. A moment equation then reads:

$$\frac{\partial}{\partial t}(n < Q >) + \frac{\partial}{\partial x_k}(n < c_k Q >) = \Delta[Q], \tag{3.6}$$

where the two operators are:

$$< Q >= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} Q f dc_1 dc_2 dc_3, \tag{3.7}$$

and

$$\Delta[Q] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} Q \left[\frac{\partial(nf)}{\partial t}\right]_{\text{coll}} dc_1 dc_2 dc_3. \tag{3.8}$$

Molecular mass, momentum and kinetic energy, $Q^{\text{INV}} = \{m, mc_i, mc^2/2\}$, possess the following property: if any of them is chosen as function $Q(c_i)$ to evaluate the moment, then in the corresponding moment equation the collisional term evaluates to zero, $\Delta[Q^{\text{INV}}] = 0$. Hence, they are usually referred to as *collisional invariants*. This result holds for any distribution function function f and any molecular interaction law.

The moment equations generated using the collisional invariants are the conservation laws of gas dynamics:

$$\frac{\partial}{\partial t}(\rho) + \frac{\partial}{\partial x_k}(\rho < c_k >) = 0, \tag{3.9}$$

$$\frac{\partial}{\partial t}(\rho < c_i >) + \frac{\partial}{\partial x_k}(\rho < c_k c_i >) = 0, \tag{3.10}$$

$$\frac{\partial}{\partial t}(\rho < c^2/2 >) + \frac{\partial}{\partial x_k}(\rho < c_k c^2/2 >) = 0, \tag{3.11}$$

where $\rho = mn$ is the mass density. Introducing the peculiar velocity components $C_i = (c_i - u_i)$, where $u_i =< c_i >$ is the mass velocity, one can define the central moments:

$$
\begin{aligned}
P_{i,j} &= \rho < C_i C_j >, && (3.12)\\
p &= P_{k,k}/3, && (3.13)\\
\tau_{i,j} &= -P_{i,j} + p\delta_{i,j}, && (3.14)\\
e &= < C^2/2 >, && (3.15)\\
q_i &= \rho < C_i C^2/2 >, && (3.16)
\end{aligned}
$$

where $P_{i,j}$ is the stress tensor, p is the pressure, $\tau_{i,j}$ is the viscous stress tensor, e is the internal energy for monatomic gas, and q_i is the heat flux vector for a monatomic gas. The conservation laws of gas dynamics can then be written in the familiar form:

$$\frac{\partial}{\partial t}(\rho) + \frac{\partial}{\partial x_k}(\rho u_k) = 0, \tag{3.17}$$

$$\frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_k}(\rho u_k u_i + P_{k,i}) = 0, \tag{3.18}$$

$$\frac{\partial}{\partial t}\left[\rho\left(e + \frac{u^2}{2}\right)\right] + \frac{\partial}{\partial x_k}\left[\rho u_k\left(e + \frac{u^2}{2}\right) + P_{k,i}u_i + q_k\right] = 0. \tag{3.19}$$

If the gas is not monatomic but has an internal structure, the energy $mc^2/2$ must be replaced by $(mc^2/2 + \epsilon)$, where ϵ is the additional internal energy per particle. It is reasonable to argue that the quantities:

$$Q^{\text{INV}} = \{m, mc_i, (mc^2/2 + \epsilon)\}. \tag{3.20}$$

are still conserved in a collision, and so Eq. 3.8 again evaluates to zero. Eqs. 3.9 and 3.10 and, consequently, Eqs. 3.17 and 3.18 are fully recovered, while Eq. 3.11 is replaced by:

$$\frac{\partial}{\partial t}(\rho < c^2/2 > + n < \epsilon >) + \frac{\partial}{\partial x_k}(\rho < c_k c^2/2 > + n < c_k \epsilon >) = 0. \tag{3.21}$$

Substituting the central moments, defined by Eqs. 3.13–3.16, into Eq. 3.21 reproduces the same algebra as for monatomic gas, and leads to

$$
\begin{aligned}
&\frac{\partial}{\partial t}\left[\rho\left(e + \frac{u^2}{2}\right) + \rho e_{\text{int}}\right] + \\
&\quad \frac{\partial}{\partial x_k}\left[\rho u_k\left(e + \frac{u^2}{2}\right) + P_{k,i}u_i + q_k + (n < C_k\epsilon > + \rho u_k e_{\text{int}})\right] = 0,
\end{aligned} \tag{3.22}
$$

where $e_{\text{int}} =< \epsilon > /m$ is the energy due to the internal molecular structure. Therefore, Eq. 3.19 is also recovered if the definition of the internal energy, Eq. 3.15, is replaced by:

$$e =< C^2/2 > +e_{int}, \tag{3.23}$$

and the definition of the heat flux vector, Eq. 3.16, is replaced by:

$$q_i = \rho < C_i C^2/2 > +n < C_i \epsilon > . \tag{3.24}$$

When computing e_{int}, a joint distribution $f(C_i, \epsilon)$ should be considered. However, if all internal molecular energy modes are in equilibrium, both internally and with the translational degrees of freedom, then C_i and ϵ are statistically independent variables, and, therefore, $f(C_i, \epsilon)$ reduces to a product of functions. Based on this assumption, the following expression is obtained:

$$e_{\text{int}} = \frac{1}{2}\left(\frac{5-3\gamma}{\gamma-1}\right) RT, \tag{3.25}$$

where R is the gas constant and T the translational temperature. Note that, for a monatomic gas, $\gamma = 5/3$ and $e_{\text{int}} = 0$.

This derivation shows that the conservation equations, Eqs. 3.17–3.19, hold for any velocity distribution function. For example, choosing the equilibrium distribution, namely the Maxwellian distribution, Eqs. 3.17–3.19 become the Euler equations; choosing the Chapman-Enskog distribution, Eqs. 3.17–3.19 become the Navier-Stokes equations. The expressions of these distribution functions are given in section 3.2.3

3.2.2 Kinetic split fluxes

Moment equations, Eq. 3.6, for the collisional invariants can be expressed as:

$$\frac{\partial U}{\partial t} + \frac{\partial F_k}{\partial x_k} = 0, \tag{3.26}$$

where the state vector U and the total flux vector F_k are:

$$U = n < Q^{\text{INV}} >, \tag{3.27}$$

$$F_k = n < c_k Q^{\text{INV}} > . \tag{3.28}$$

We stress that the five fluxes defined by Eq. 3.28 are total fluxes. These general expressions contain both the inviscid fluxes as well as non-equilibrium components due to viscous stress and heat flux. Kinetic splitting of the total flux is achieved by splitting the integration in velocity space. For example, when splitting the flux along the x_1 direction, the integration is split as:

$$\begin{aligned}&\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\{\ldots\}dc_1dc_2dc_3 \equiv \\ &\quad\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left(\int_{-\infty}^{0}+\int_{0}^{\infty}\right)\{\ldots\}dc_1dc_2dc_3 \equiv \\ &\quad\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left(\int_{-\infty}^{-u_1}+\int_{-u_1}^{\infty}\right)\{\ldots\}dC_1dC_2dC_3.\end{aligned} \tag{3.29}$$

In the following, we consider the splitting along the x_1-direction of the flux component along the same direction, F_1. Analogous considerations can be made for directions x_2 and x_3. Also, for simplicity, we drop the subscript and write F in lieu of F_1.

As in any flux splitting formula, a kinetic split flux is written as: $F = F^+ + F^-$. Peculiar to kinetic splitting, instead, are the use of the kinetic expression for the flux, Eq. 3.28, and the splitting of the integral over velocity space, Eq. 3.29. Combining the two, we obtain:

$$F^- = n\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{-u_1}(C_1+u_1)Q^{\text{INV}}f dC_1dC_2dC_3, \tag{3.30}$$

$$F^+ = n\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-u_1}^{\infty}(C_1+u_1)Q^{\text{INV}}f dC_1dC_2dC_3. \tag{3.31}$$

These relations are actually used to compute F when f is known. We introduce the following physically descriptive notation:

$$F^+_{\text{zero}} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-u_1}^{\infty} f dC_1dC_2dC_3, \tag{3.32}$$

$$F^+_{\text{mass}} = \rho\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-u_1}^{\infty}(C_1+u_1) f dC_1dC_2dC_3, \tag{3.33}$$

$$F^+_{\text{mom}_{x_1}} = \rho\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-u_1}^{\infty}(C_1+u_1)^2 f dC_1dC_2dC_3, \tag{3.34}$$

$$F^+_{\text{mom}_{x_2}} = \rho\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-u_1}^{\infty}(C_1+u_1)(C_2+u_2) f dC_1dC_2dC_3, \tag{3.35}$$

$$\begin{aligned}F^+_{\text{energy}_{\text{tr}}} = \rho\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-u_1}^{\infty}&(C_1+u_1)\frac{1}{2}[(C_1+u_1)^2+ \\ &(C_2+u_2)^2+(C_3+u_3)^2] f dC_1dC_2dC_3,\end{aligned} \tag{3.36}$$

$$F^+_{\text{energy}_{\text{int}}} = n\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-u_1}^{\infty}(C_1+u_1)\epsilon f dC_1dC_2dC_3. \tag{3.37}$$

A notation analogous to Eqs. 3.32–3.37 is introduced for the negative flux, where the integration over C_1 runs from $-\infty$ to $-u_1$.

The quantity F^+_{zero} defines how the velocity distribution function itself is split, and satisfies the normalisation condition $F^+_{\text{zero}}+F^-_{\text{zero}} = 1$ for a probability distribution. The quantities $F^+_{\text{mom}_{x_1}}$ and $F^+_{\text{mom}_{x_2}}$ are the positive fluxes of the momentum components along x_1 and x_2, respectively. The flux of the x_3-component is not listed because its expression is analogous to that of the x_2-component. The quantities $F^+_{\text{energy}_{\text{tr}}}$ and $F^+_{\text{energy}_{\text{int}}}$ are the positive fluxes of the total energy for a monatomic gas and of the energy due to the internal molecular structure, respectively. The positive flux of the total energy for a gas with internal molecular structure is the sum of the two:

$$F^+_{\text{energy}} = F^+_{\text{energy}_{\text{tr}}} + F^+_{\text{energy}_{\text{int}}}. \tag{3.38}$$

When computing the contribution $F^+_{\text{energy}_{\text{int}}}$, the integral is further split into a contribution due to the thermal velocity C_1 and a contribution due to the mass velocity u_1:

$$F^+_{\text{energy}_{\text{int}}} = \Delta q^+_{\text{Eucken}} + \rho u_1 e^+_{\text{int}}. \tag{3.39}$$

This splitting is motivated by the difficulty in modelling the first term, $\Delta q^+_{\text{Eucken}}$, that represents the diffusive flux of energy due to the internal molecular structure. It is the positive part of the second term appearing in the heat flux kinetic expression, Eq. 3.24, and its explicit computation requires a joint distribution function, $f(C_i, \epsilon)$, which is not available. Therefore, Eucken's model is used, which replaces $< C_1\epsilon >$ by a quantity proportional to the temperature gradient:

$$\Delta q_{\text{Eucken}} = \Delta q^+_{\text{Eucken}} + \Delta q^-_{\text{Eucken}} = -K\nabla T. \tag{3.40}$$

Within this approximation, Δq_{Eucken} is absorbed into the diffusive flux of the translational energy, provided a thermal conductivity for a gas with internal structure, $k = k^{(1)} + K$, is used instead of the value for a monatomic gas, $k^{(1)}$.

The term $\rho u_1 e^+_{\text{int}}$, on the other hand, represents the advective flux of the energy due to the internal molecular structure, and it is easily computed as:

$$\rho u_1 e^+_{\text{int}} = F^+_{\text{mass}} e_{\text{int}}, \tag{3.41}$$

where e_{int} is given by Eq. 3.25, since the model assumes equilibrium for the internal degrees of freedom.

Finally, for the purpose of flux splitting, the total flux of the energy due to the internal molecular structure is identified with its advective component:

$$F^{+}_{\mathrm{energy}_{\mathrm{int}}} \equiv F^{+}_{\mathrm{mass}} e_{\mathrm{int}}, \tag{3.42}$$

while its diffusive component is accounted for by considering a modified thermal conductivity.

3.2.3 Chapman-Enskog split fluxes

A gas flow that is in local thermodynamic equilibrium is represented locally by a Maxwellian distribution, and a gas flow that is slightly disturbed from the equilibrium state is represented locally by the Chapman-Enskog (CE) distribution. The CE distribution (61) is obtained as an approximate solution to the Boltzmann equation for a monatomic gas and is expressed as a product of a local Maxwellian distribution and a polynomial function of the thermal velocity components, C_i:

$$f^{\mathrm{CE}} = f^{\mathrm{Max}}(1 + \phi_1 + \phi_2), \tag{3.43}$$

where:

$$\begin{aligned}
f^{\mathrm{Max}} &= (2\pi RT)^{-3/2}\exp(-C^2/2RT), \\
\phi_1 &= -\left(\frac{\rho}{p^2}\right)\left(k^{(1)}\frac{\partial T}{\partial x_k}\right) C_k(C^2/5RT - 1), \\
\phi_2 &= -\left(\frac{\rho}{p^2}\right)\left(\mu^{(1)}\frac{\partial u_j}{\partial x_k}\right)\left(C_j C_k - \frac{1}{3}C^2\delta_{j,k}\right),
\end{aligned}$$

and where $k^{(1)}$ is the coefficient of thermal conductivity and $\mu^{(1)}$ is the coefficient of viscosity as determined by the Chapman-Enskog procedure, and $\delta_{j,k}$ is the Kronecker delta. Both temperature and velocity gradients appear as parameters in f^{CE} and notational efficiency can be gained by introducing the Navier-Stokes-Fourier expressions for stress and heat flux:

$$q_i^{\mathrm{CE}} = -k^{(1)}\frac{\partial T}{\partial x_i}, \tag{3.44}$$

$$\tau_{i,j}^{\mathrm{CE}} = \mu^{(1)}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\mu^{(1)}\left(\frac{\partial u_k}{\partial x_k}\right)\delta_{i,j}. \tag{3.45}$$

Substituting Eq. 3.43 into Eqs. 3.32–3.37, all integrands become products of polynomials in the thermal velocity components, C_i, and the Maxwellian distribution, f^{Max}.

Therefore, the integration is relatively straightforward, and details can be found in the work by Chou (69). Terms containing odd powers of C_2 and C_3 evaluate to zero, because f^{Max} is a symmetric function, while terms containing even powers are known functions of RT. Integration in the C_1 component is split into integrals from $-u_1$ to 0 and from 0 to ∞, obtaining exponential and error functions (68). Finally, the following expressions are obtained:

$$\begin{aligned}
F^{\pm}_{\text{zero}} &= 1/2[(1 \pm \alpha_1) \pm \alpha_2(S_1\hat{\tau}^{\mathrm{CE}}_{1,1} + (2S_1^2 - 1)q_1^{\mathrm{CE}})], && (3.46)\\
F^{\pm}_{\text{mass}} &= \rho\sqrt{RT/2}[(1 \pm \alpha_1)S_1 \pm \alpha_2(1 + \chi_1)], && (3.47)\\
F^{\pm}_{x_1-\text{mom}} &= p[(1 \pm \alpha_1)(S_1^2 + 1/2(1 - \hat{\tau}^{\mathrm{CE}}_{1,1})) \pm \alpha_2(S_1 + \hat{q}_1^{\mathrm{CE}})], && (3.48)\\
F_{x_2-\text{mom}} &= \sqrt{2RT}[S_2F^{\pm}_{\text{mass}}] + 1/2p[-(1 \pm \alpha_1)\hat{\tau}^{\mathrm{CE}}_{1,2} \pm \alpha_2\hat{q}_2^{\mathrm{CE}}], && (3.49)\\
F^{\pm}_{\text{tr-energy}} &= p\sqrt{RT/2}[(1 \pm \alpha_1)(S_2(5/2 + S^2) + \chi_2) \pm \\
&\quad \alpha_2(2 + S^2 + \chi_3)], && (3.50)\\
F^{\pm}_{\text{int-energy}} &= \rho u_1 e^{\pm}_{\text{int}} = \frac{1}{2}\left(\frac{5 - 3\gamma}{\gamma - 1}\right) RTF^{\pm}_{\text{mass}}, && (3.51)
\end{aligned}$$

where

$$\begin{aligned}
\alpha_1 &= \operatorname{erf}(S_1),\\
\alpha_2 &= \frac{1}{\sqrt{\pi}}\exp(-S_1^2),\\
\chi_1 &= S_1\hat{q}_1^{\mathrm{CE}} + \frac{1}{2}\hat{\tau}^{\mathrm{CE}}_{1,1},\\
\chi_2 &= \frac{5}{2}\hat{q}_1^{\mathrm{CE}} - (S_1\hat{\tau}^{\mathrm{CE}}_{1,1} + S_2\hat{\tau}^{\mathrm{CE}}_{1,2} + S_3\hat{\tau}^{\mathrm{CE}}_{1,3}),\\
\chi_3 &= S_2\hat{q}_2^{\mathrm{CE}} + S_3\hat{q}_3^{\mathrm{CE}} - \chi_1(1 + S_2^2 + S_3^2) - \hat{\tau}^{\mathrm{CE}}_{1,1},\\
S_i &= u_i/\sqrt{2RT},\\
S^2 &= S_1^2 + S_2^2 + S_3^2,\\
\hat{\tau}^{\mathrm{CE}}_{i,j} &= \tau^{\mathrm{CE}}_{i,j}/p,\\
\hat{q}_i^{\mathrm{CE}} &= \frac{2}{5}q_i^{\mathrm{CE}}/(p\sqrt{2RT}).
\end{aligned}$$

As with Eqs. 3.32–3.37, the split flux of the x_3-momentum is not listed as its expression is analogous to that of the x_2-momentum. Note that Eqs. 3.47–3.51 satisfies the consistency condition $F^+ + F^- = F$, and Eq. 3.46 satisfy the normalisation condition $F_{\text{zero}} = F^+_{\text{zero}} + F^-_{\text{zero}} = 1$.

Eqs. 3.46–3.50 are not affected by the internal molecular structure of the gas, i.e. by the value of γ. Therefore, they should not contain γ explicitly, when expressed appropriately. This is accomplished by introducing the speed ratio, $S_i = u_i/\sqrt{2RT}$, often employed in kinetic theory instead of the Mach number (that contains γ through the speed of sound).

It is worth recalling the approximation behind these expressions when a gas with internal molecular structure is considered. In this case, Δq_{Eucken} is absorbed into $F_{\text{tr-energy}}$ by replacing $k^{(1)}$ with $k = k^{(1)} + K$ in Eq. 3.45. Even assuming Eucken's approximation holds, this does not guarantee that the splitting is correct, i.e. that, when the modified conductivity is introduced into χ_2 and χ_3 in Eq. 3.50, it will properly account for the absorption of the split quantities $\Delta q^{\pm}_{\text{Eucken}}$ in Eq. 3.51. Furthermore, the expression Eq. 3.25, employed in Eq. 3.51, is based on the assumption that C_i and ϵ are statistically independent variables. If the same assumption were used to evaluate $< C_1\epsilon >$, then we would have $\Delta q_{\text{Eucken}} = n < C_1\epsilon >= n < C_1 >< \epsilon >= 0$, as by definition $< C_1 >= 0$, and Eucken's approximation would be lost.

If the non-equilibrium parameters $\hat{\tau}^{\text{CE}}$ and $\hat{q}^{\text{CE}}$ are set to zero, then the kinetic split flux for a Maxwellian distribution, i.e. the Euler split flux, is recovered:

$$
\begin{aligned}
F^{\pm}_{\text{zero}} &= 1/2(1 \pm \alpha_1), && (3.52)\\
F^{\pm}_{\text{mass}} &= \rho\sqrt{RT/2}[(1 \pm \alpha_1)S_1 \pm \alpha_2], && (3.53)\\
F^{\pm}_{x_1-\text{mom}} &= p[(1 \pm \alpha_1)(S_1^2 + 1/2) \pm \alpha_2 S_1], && (3.54)\\
F_{x_2-\text{mom}} &= \sqrt{2RT}[S_2 F^{\pm}_{\text{mass}}], && (3.55)\\
F^{\pm}_{\text{tr-emergy}} &= p\sqrt{RT/2}[(1 \pm \alpha_1)S_2(5/2 + S^2) \pm \alpha_2(2 + S^2)], && (3.56)\\
F^{\pm}_{\text{int-energy}} &= \frac{1}{2}\left(\frac{5-3\gamma}{\gamma-1}\right) RT F^{\pm}_{\text{mass}}. && (3.57)
\end{aligned}
$$

These expressions, derived as outlined in this section by Chou and Baganoff (56), are also found in the works by Patterson (70), Pullin (71), Mandal and Deshpande (72) and Mallet *et al.* (73).

3.3 Numerical method for the Navier-Stokes equations

In a Cartesian reference frame (x, y, z) the Navier-Stokes equations in vector conservation form read:

$$\frac{\partial Q}{\partial t} + \frac{\partial (E + E_v)}{\partial x} + \frac{\partial (F + F_v)}{\partial y} + \frac{\partial (G + G_v)}{\partial z} = 0. \tag{3.58}$$

The quantities Q, E, F and G were introduced in section 2.3 when discussing the Euler equations, and their expressions are given in Eqs. 2.42. The viscous flux components along x, y and z are:

$$E_v = \left\{ \begin{array}{c} 0 \\ -\tau_{x,x} \\ -\tau_{x,y} \\ -\tau_{x,z} \\ -\tau_{x,x}u - \tau_{x,y}v - \tau_{x,z}w + q_x \end{array} \right\},$$

$$F_v = \left\{ \begin{array}{c} 0 \\ -\tau_{y,x} \\ -\tau_{y,y} \\ -\tau_{y,z} \\ -\tau_{y,x}u - \tau_{y,y}v - \tau_{y,z}w + q_y \end{array} \right\},$$

$$G_v = \left\{ \begin{array}{c} 0 \\ -\tau_{z,x} \\ -\tau_{z,y} \\ -\tau_{z,z} \\ -\tau_{z,x}u - \tau_{z,y}v - \tau_{z,z}w + q_z \end{array} \right\}.$$

The stress tensor τ and the heat flux vector q are:

$$\begin{aligned} \tau &= \begin{bmatrix} \tau_{x,x} & \tau_{x,y} & \tau_{x,z} \\ \tau_{y,x} & \tau_{y,y} & \tau_{y,z} \\ \tau_{z,x} & \tau_{x,y} & \tau_{z,z} \end{bmatrix} = \mu(\nabla V + \nabla V^{\dagger}) - \tfrac{2}{3}\mu(\nabla \cdot V)I, \\ q &= \left\{ \begin{array}{c} q_x \\ q_y \\ q_z \end{array} \right\} = -c_p\mu/Pr\nabla T. \end{aligned} \tag{3.59}$$

In the above equations, V is the velocity vector, with u, v and w its components along the x, y and z directions, respectively, T is the temperature, μ the viscosity, $Pr = c_p\mu/k$ the Prandtl number, k the thermal conductivity, c_p the constant-pressure specific heat, I the identity tensor and superscript $\dagger$ denotes tensor transposition.

In order to solve Eq. 3.58 on domains of arbitrary shape using FD methods, Eq. 3.58 is cast in its general coordinate form:

$$\frac{\partial \tilde{Q}}{\partial t} + \frac{\partial (\tilde{E} + \tilde{E}_v)}{\partial \xi} + \frac{\partial (\tilde{F} + \tilde{F}_v)}{\partial \eta} + \frac{\partial (\tilde{G} + \tilde{G}_v)}{\partial \zeta} = 0. \tag{3.60}$$

The quantities $\tilde{Q}$, $\tilde{E}$, $\tilde{F}$ and $\tilde{G}$ are defined in Eqs. 2.44–2.45, and the viscous flux components in general coordinates are:

$$\tilde{E}_v = \frac{\xi_x E_v + \xi_y F_v + \xi_z G_v}{J},$$
$$\tilde{F}_v = \frac{\eta_x E_v + \eta_y F_v + \eta_z G_v}{J},$$
$$\tilde{G}_v = \frac{\zeta_x E_v + \zeta_y F_v + \zeta_z G_v}{J}.$$

The metrics ξ_x, ξ_y, ξ_z, η_x, η_y, η_z, ζ_x, ζ_y, ζ_z define a transformation from an arbitrarily-shaped physical domain to a cubic domain, and J is the Jacobian of the transformation. In order to use FD formulas on equally spaced grids, this transformation is generally chosen such that the computational grid in physical space maps onto a uniform grid in the computational space.

Using the method of lines to solve Eq. 3.60, the problem is split into three one-dimensional problems, and the algorithm applied to solve the one-dimensional problem along ξ, for example, is applied along η and ζ as well. The 1D problem along ξ reads:

$$\frac{\partial \tilde{Q}}{\partial t} + \frac{\partial (\tilde{E} + \tilde{E}_v)}{\partial \xi} = 0. \tag{3.61}$$

3.3.1 Stress tensor and heat flux vector calculation

The calculation of the stress tensor and the heat flux vector requires the evaluation of velocity and temperature gradients. The derivatives are computed in the computational space using classical FD compact-central formulas, and then projected into the physical space.

Let f be any of the velocity components, u, v, w, or the temperature, T. Let i be the index running from 0 to $N+1$ along a ξ-parallel grid line, h the (constant) grid spacing in computational space, and $f'_i = f_\xi(x_{i,j,k}, y_{i,j,k}, z_{i,j,k})$ the derivative at node (i,j,k). The following sixth-order FD compact-central formula can be found in the work by Lele (3), and is used to compute derivatives at the interior nodes:

$$f'_{i-1} + 3f'_i + f'_{i+1} = \frac{1}{h}\left[\frac{7}{3}(f_{i+1} - f_{i-1}) + \frac{1}{12}(f_{i+2} - f_{i-2})\right], \quad i = 2, \ldots, N-1. \tag{3.62}$$

The basic steps for deriving this formula are outlined in section 1.1. Boundary closures can be derived in a similar fashion, removing the constraints of a symmetric stencil and coefficients. The following sixth-order compact boundary closures can be found in the work by Zhong (66), and are used to compute derivatives at nodes 0 and 1:

$$\begin{aligned} f_0' + 5f_1' &= \frac{1}{h}\left(-\frac{197}{60}f_0 - \frac{5}{12}f_1 + 5f_2 - \frac{5}{3}f_3 + \frac{5}{12}f_4 - \frac{1}{20}f_5\right), \quad (3.63) \\ \frac{1}{8}f_0' + f_1' + \frac{3}{4}f_2' &= \frac{1}{h}\left(-\frac{43}{96}f_0 - \frac{5}{6}f_1 + \frac{9}{8}f_2 + \frac{1}{6}f_3 - \frac{1}{96}f_4\right). \quad (3.64) \end{aligned}$$

The boundary closures for nodes N and $N+1$ can be inferred from these by symmetry considerations. We recall that indices 0 and $N+1$ denote ghost nodes for CU5-TVD-NSK and boundary nodes for CUDA-CU5-TVD-NS. Solving the tri-diagonal linear system of Eqs. 3.62 at interior nodes and Eqs. 3.63 and 3.64 at boundary nodes, the ξ-derivatives at all nodes are computed.

The same procedure is applied along η and ζ grid lines, and, eventually, the derivatives in computational space, f_ξ, f_η and f_ζ, are known at all nodes. The gradient in the physical domain is finally computed using the metrics:

$$\nabla f = \left\{ \begin{array}{c} f_x \\ f_y \\ f_z \end{array} \right\} = \left[\begin{array}{ccc} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{array} \right] \left\{ \begin{array}{c} f_\xi \\ f_\eta \\ f_\zeta \end{array} \right\}.$$

Velocity and temperature gradients are used in Eqs. 3.59 to compute the stress tensor and the heat flux vector, which are then fed into Eqs. 3.66 and 3.67.

3.3.2 Differentiation of the viscous flux

CUDA-CU5-TVD-NS employs the classic operator splitting. In Eq. 3.61 inviscid and viscous fluxes are separated: $\partial_\xi(\tilde{E}+\tilde{E}_v) = \partial_\xi\tilde{E}+\partial_\xi\tilde{E}_v$. The inviscid flux is differentiated as in the Euler solver described in section 2.3, while the viscous flux is differentiated using the compact central formula Eq. 3.62 with boundary closures Eqs. 3.63 and 3.64. We adopted this approach so that we could use the characteristic boundary conditions described in section 2.5.2 with minor modifications.

CU5-TVD-NSK, instead, follows the fully conservative discretisation model, Eq. 3.3. Defining the Navier-Stokes (NS) flux, $\tilde{E}_{NS} = \tilde{E} + \tilde{E}_v$, Eq. 3.61 has the same formal expression as Eq. 2.46, and the same solution algorithm described in section 2.3 can be applied, provided a suitable expression for the NS split flux, $\tilde{E}^{\pm}_{NS}$, is given.

This expression is provided by the kinetic splitting Eqs. 3.47–3.51, when x_1 is the ξ-parallel direction, identified by the unit vector $(n_x, n_y, n_z) = \nabla\xi/|\nabla\xi|$. Transformation from the (x, y, z) reference frame to the (x_1, x_2, x_3) reference frame is achieved through the rotation matrix:

$$B = \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

where:

$$\begin{array}{rclrcl} \sin\phi & = & n_z, & \cos\phi & = & \sqrt{1-\sin^2\phi} \\ \sin\theta & = & n_y/\cos\phi, & \cos\theta & = & n_x/\cos\phi. \end{array}$$

The velocity, heat flux and stress tensor in the (x_1, x_2, x_3) reference frame are computed as follows:

$$\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = B \begin{Bmatrix} u \\ v \\ w \end{Bmatrix}, \tag{3.65}$$

$$\begin{Bmatrix} q_1 \\ q_2 \\ q_3 \end{Bmatrix} = B \begin{Bmatrix} q_x \\ q_y \\ q_z \end{Bmatrix}, \tag{3.66}$$

$$\begin{bmatrix} \tau_{1,1} & \tau_{1,2} & \tau_{1,3} \\ \tau_{2,1} & \tau_{2,2} & \tau_{2,3} \\ \tau_{3,1} & \tau_{3,2} & \tau_{3,3} \end{bmatrix} = B \begin{bmatrix} \tau_{x,x} & \tau_{x,y} & \tau_{x,z} \\ \tau_{y,x} & \tau_{y,y} & \tau_{y,z} \\ \tau_{z,x} & \tau_{z,y} & \tau_{z,z} \end{bmatrix} B^\dagger. \tag{3.67}$$

Once these quantities are known, Eqs. 3.47–3.51 are used to compute $F^\pm_{\text{mass}}$, $F^\pm_{x_1-\text{mom}}$, $F^\pm_{x_2-\text{mom}}$, $F^\pm_{x_3-\text{mom}}$, $F^\pm_{\text{tr-energy}}$ and $F^\pm_{\text{int-energy}}$. The fluxes of the momentum components in the (x, y, z) reference frame are then computed:

$$\begin{Bmatrix} F^\pm_{x-\text{mom}} \\ F^\pm_{y-\text{mom}} \\ F^\pm_{z-\text{mom}} \end{Bmatrix} = B^\dagger \begin{Bmatrix} F^\pm_{x_1-\text{mom}} \\ F^\pm_{x_2-\text{mom}} \\ F^\pm_{x_3-\text{mom}} \end{Bmatrix},$$

and, finally, the NS split flux:

$$\tilde{E}^\pm_{NS} = \frac{|\nabla\xi|}{J} \begin{Bmatrix} F^\pm_{\text{mass}} \\ F^\pm_{x-\text{mom}} \\ F^\pm_{y-\text{mom}} \\ F^\pm_{z-\text{mom}} \\ F^\pm_{\text{tr-energy}} + F^\pm_{\text{int-energy}} \end{Bmatrix}.$$

3.3.3 Boundary conditions

When the Navier-Stokes equations are the governing equations and the continuum hypothesis holds, steady solid walls are modelled as no-slip surfaces. At a no-slip surface, by definition, the velocity is zero.

CU5-TVD-NSK employs a cell-centred FD discretisation and this no-slip condition is enforced by simply assigning to the ghost node a velocity opposite to the one in the first interior node. So, if subscript 0 denotes the ghost node and subscript 1 the first interior node, $V_0 = -V_1$.

A result of the theory of boundary layers over flat surfaces is that the pressure gradient normal to the surface is zero. Although this condition is not exact if the surface has a curvature, it is commonly used in CFD, and easily implemented by simply assigning to the ghost node the same pressure as in the first interior node, i.e. $p_0 = p_1$. For the second thermodynamic variable, two types of no-slip wall are considered:

- **Adiabatic wall**: the condition of zero heat flux normal to the wall is simply enforced by assigning to the ghost node the same temperature as the first interior node, i.e. $T_0 = T_1$.
- **Isothermal wall**: the temperature in the ghost node is linearly extrapolated from the wall temperature, T_w, and the temperature at the first interior node, T_1, i.e. $T_0 = 2T_w - T_1$.

CUDA-CU5-TVD-NS, instead, employs a vertex-based discretisation and characteristic boundary conditions. The Navier-Stokes Characteristic Boundary Conditions (NSCBC) follow the same formulation as the CBC described in section 2.5.2, but in Eq. 2.64 the right-hand-side includes the viscous terms:

$$S = J\left[\breve{E}\frac{\partial\left(|\nabla\xi|/J\right)}{\partial\xi} + \frac{\partial\tilde{F}}{\partial\eta} + \frac{\partial\tilde{G}}{\partial\zeta} + \frac{\partial\tilde{E}_v}{\partial\xi} + \frac{\partial\tilde{F}_v}{\partial\eta} + \frac{\partial\tilde{G}_v}{\partial\zeta}\right].$$

Provided this definition of S is adopted, the formulation for farfield and slip-wall boundaries is identical. A third type of boundary, namely isothermal no-slip wall, has been implemented for viscous simulations. The physical boundary condition to be enforced is zero velocity and fixed wall temperature, T_w. Initialising the flow field at the boundary so that

$$Q = \left\{ \begin{array}{c} \rho \\ 0 \\ 0 \\ 0 \\ \rho c_p T_w \end{array} \right\},$$

the condition

$$\partial_t Q = \partial_t \rho \left\{ \begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ c_p T_w \end{array} \right\} \tag{3.68}$$

must be ensured at any time. Following the NSCBC approach, $\partial_t \rho$ should be computed from the characteristic equation corresponding to the outgoing wave. In our tests, this approach led to a large overestimation of the wall-density and, eventually, to instability. Therefore, we drop the characteristic treatment and simply compute $\partial_t \rho$ from the continuity equation. The excellent agreement between computed wall pressures and experimental measurements, shown in section 6.2.2, demonstrates the validity of this assumption. Finally, the time derivative of the conserved variables vector is computed from Eq. 3.68.

3.3.4 Thermodynamic model

We use the non-dimensional form of the Navier-Stokes equations, i.e. the variables are non-dimensionalised as follows:

$$\rho = \frac{\rho^d}{\rho_{ref}}, \quad p = \frac{p^d}{\rho_{ref} V_{ref}^2}, \qquad T = \frac{T^d}{T_{ref}}, \quad t = \frac{t^d V_{ref}}{L_{ref}},$$

$$\left\{ \begin{array}{c} u \\ v \\ w \end{array} \right\} = \frac{1}{V_{ref}} \left\{ \begin{array}{c} u^d \\ v^d \\ w^d \end{array} \right\}, \qquad \left\{ \begin{array}{c} x \\ y \\ z \end{array} \right\} = \frac{1}{L_{ref}} \left\{ \begin{array}{c} x^d \\ y^d \\ z^d \end{array} \right\},$$

where the superscript d denotes dimensional quantities and the subscript ref denotes reference values, expressed in consistent units. In our codes the adiabatic index, γ, the Prandtl number, Pr, the reference Mach number, $M = V_{ref}/a_{ref}$, and the reference Reynolds number, $Re = \rho_{ref} V_{ref} L_{ref}/\mu_{ref}$, are supplied as input. The reference speed of sound is $a_{ref} = \sqrt{\gamma R_{ref} T_{ref}}$, where R_{ref} is the reference gas constant.

We use the calorically perfect gas model. The non-dimensional equation of state is:

$$p = \rho \frac{1}{\gamma M^2} T,$$

where the non-dimensional gas constant is: $R = R_{ref}T_{ref}/V_{ref}^2 = 1/\gamma M^2$. For this definition of the non-dimensional gas constant, the non-dimensional constant-pressure specific heat is:

$$c_p = \frac{\gamma R}{\gamma - 1} = \frac{1}{(\gamma - 1)M^2}.$$

The non-dimensional viscosity is: $\mu = \mu^d/\mu_{ref}Re$, and is computed through Sutherland's law:

$$\mu(T) = \frac{1}{Re}T^{3/2}\frac{1 + T_S}{T + T_S},$$

where $T_S = 110.4/T_{ref}$, if T_{ref} is expressed in Kelvin.

3.4 Summary

In this chapter we have presented two Navier-Stokes solvers based on the compact-TVD method described in Chapter 2. One method relies on the classic operator splitting to include viscous effects, while the second employs a splitting for the Navier-Stokes flux.

Our main contribution is having identified the possibility to combine the Kinetic FVS for the NS flux, devised by Chou and Baganoff (56), with the compact-TVD method. Combining the two techniques has several advantages:

- the computational cost of the compact scheme, associated with the solution of linear systems to compute derivatives, is reduced because fewer linear systems are solved;
- the loss of accuracy associated with the TVD filter is mitigated, because the filter is applied to the NS flux, which is smoother than the inviscid flux.

The method is validated in Chapter 5.

4

HPC implementations of the Navier-Stokes solver

In this chapter we describe how the Navier-stokes solvers presented in Chapter 3 have been developed in order to take advantage of the latest HPC systems. Original contributions included in this chapter are: 1) a novel algorithm employing a memory-dependent partitioning strategy, suitable for MPP systems of multi-core processors; 2) a novel algorithm that, employing a task-dependent partitioning strategy, is able to exploit the computing power of GPUs.

Compact methods for gas dynamics are computationally intensive and potential applications, such as DNS and LES, challenge the capabilities of modern computers. Therefore, it is essential for numerical codes based on compact methods to be able to use efficiently the most powerful computing facilities and devices.

Despite the success of compact methods in solving relatively simple problems, few large scale simulations of complex problems have been reported. A fundamental obstacle that prevents compact methods from being applied to large scale simulations is their semi-global nature: the derivative of any variable along a grid-line at a node depends on all of the nodes on that grid-line. It is evident that this dependence makes the parallel calculation of the derivative difficult, and why the serial algorithm must be re-designed in order to scale to multiple processing units.

Today, supercomputers are MPP systems hosting up to thousands of interconnected CPUs, each one with its own memory attached. Very often, each of these CPUs hosts several execution units, referred to as *cores*. In some of the most advanced supercomputers, each CPU has also a co-processor attached, which is a powerful parallel machine on its own. These features of modern supercomputers present outstanding challenges for a scientific programmer wishing to make the best use of the available resources.

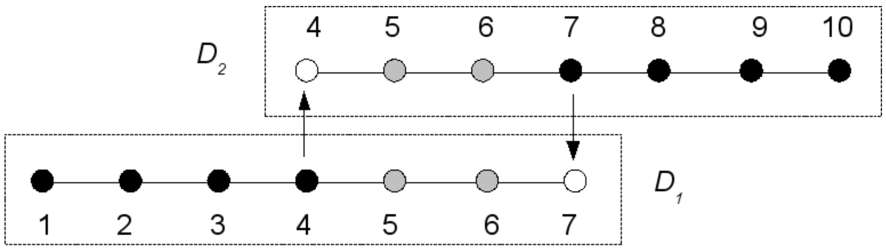
First, we have devised a strategy to parallelise our Navier-Stokes solver, when the computing device is a cluster of inter-connected CPUs, each one with its own memory. Then, we have modified this algorithm to exploit the multi-core architecture of modern CPUs. In order to do so, a different parallelisation strategy, relying on the avalability of a memory shared between several execution units, is nested into the distributed-memory parallel algorithm. These topics are covered in section 4.1. Finally, we have devised an implementation of the Navier-Stokes solver that is able to use efficiently the computing power of a GPU co-processor. A new code has been developed for this purpose, because the GPU architecture is not as flexible as the CPU one, and therefore the code performance is extremely sensitive to the implementation. Details of the issues to be faced and of the implemented solutions are given in section 4.2.

4.1 Parallel implementation of the Navier-Stokes solver

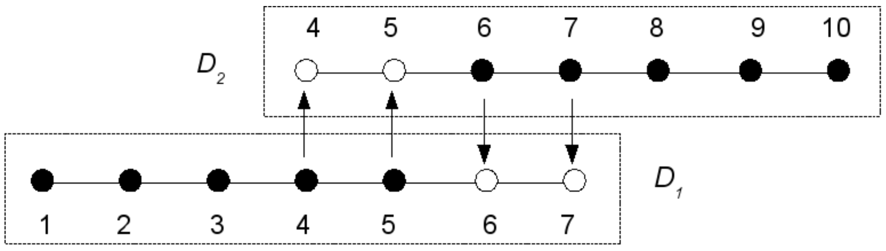
In section 4.1.1 we describe several parallel algorithms for compact schemes based on structured-block domain-decomposition that is suitable for a distributed-memory computing arcitecture. In section 4.1.2 a different parallisation approach, suitable for a shared-memory environment, is described. In section 4.1.3 our Navier-Stokes solver employing dual-level parallelism is presented.

4.1.1 Multi-block domain decomposition algorithms

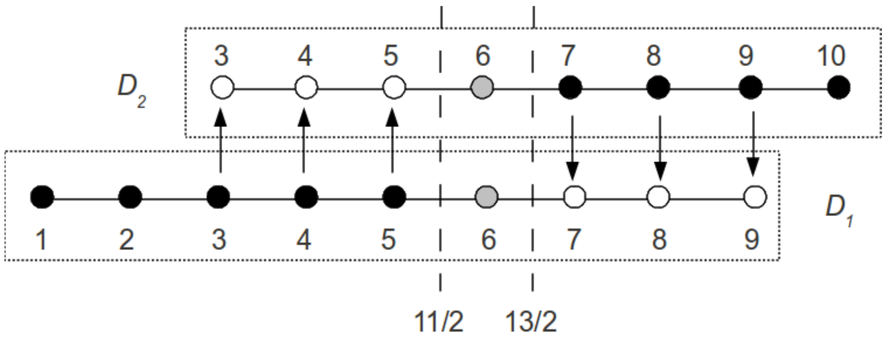
The first attempts to parallelise compact methods aimed to solve in parallel the underlying linear system. In our case, for example, the system of Eqs. 2.35–2.38 could be solved in parallel. Such parallel algorithms were developed during the 1990s: Sun and Moitra (29) devised the reduced Parallel Diagonal Dominant (PDD) algorithm for the parallel inversion of tri-diagonal linear systems, and Povitsky (30) devised the Parallel Thomas Algorithm (PTA). Both algorithms are direct solvers and scale very poorly with the number of processors, as shown by Ladeinde *et al.* (31). These authors conducted a systematic comparison of the two solvers for flow simulations, concluding that the PDD method scales better than the PTA, but with measured speed-ups of



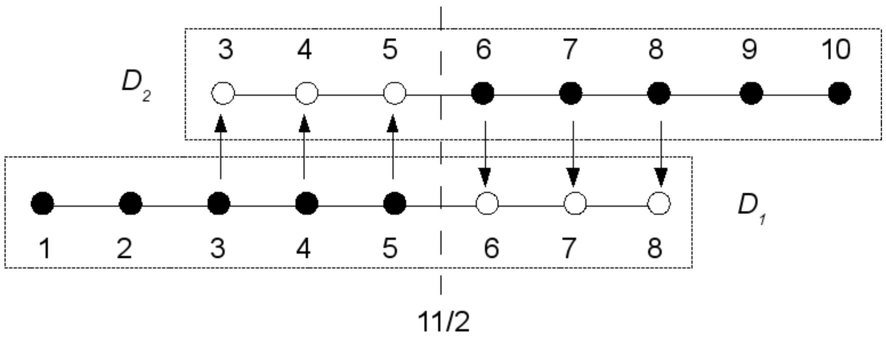
(a) Schwarz method.



(b) Gaitonde's method (CC6-MB).



(c) Chao's method.



(d) Present method (CU5-MB).

Figure 4.1: Parallel algorithms for compact schemes based on structured-block domain decomposition.

only about 50% on 16 processors. It is, therefore, evident that these direct solvers are not suitable for massively parallel computing.

For compact finite-difference methods, several authors have considered a multi-block approach, widely used for finite-difference and finite-volume explicit schemes. In a multi-block approach, the computational domain is partitioned into several sub-domains or *blocks*. From the computing point of view, if several processors are available each one is assigned the task of solving the governing equations in one of the sub-domains. Domain-partitioning explicitly enforces data locality, and so it suits the distributed-memory architecture of large clusters, where the bottle-neck for achieving good parallel performance is the data transfer between processors.

When solving a problem of a global nature using domain decomposition, the *Schwarz method* is commonly used. It is an iterative procedure to find an approximate solution to a given problem by solving many problems of smaller size. In principle, it can be applied as a serial method, but it is popular in parallel computing because the smaller problems can be solved independently by several processors with a few communications per iteration. For compact schemes, the global problem to be solved is the linear system to compute the derivatives. Consider, for example, the model problem pictured in Fig. 4.1(a): it is a grid line with 10 nodes where derivatives must be computed using a compact scheme and two sub-domains. This model problem is used in this section to compare several parallelisation strategies. The sub-domains D_1 and D_2 overlap and, for this example, the width of the overlap region is arbitrarily set to 4 nodes. According to the Schwarz method, the derivatives of a variable at the nodes are computed by solving two linear systems: $\mathscr{P}_1$ in sub-domain D_1 and $\mathscr{P}_2$ in sub-domain D_2. In general, two different solutions will exist in the overlap region. In Fig. 4.1(a), nodes 5 and 6, coloured in grey, are *shared* nodes: they are used to assess the consistency between the two computed solutions. If they do not agree within a certain tolerance, the two linear systems are solved again with modified interface-boundary conditions: the solution of the problem $\mathscr{P}_2$ at node 7 is used as a boundary condition to modify $\mathscr{P}_1$, while the solution of the problem $\mathscr{P}_1$ at node 4 is used as a boundary condition to modify $\mathscr{P}_2$. Indeed, the nodes 4 and 7 are *interface-boundary* nodes and are identified by the colour white. The process is repeated until the solutions to the local problems in the overlap region agree within a certain tolerance.

From a parallel computing point of view, in a Schwarz procedure the cost of communication is very limited: once per iteration the values of the solution at two shared nodes must be communicated between processors for the consistency check, while the values at two interface-boundary nodes must be communicated to supply boundary values for the next iteration. The major drawback is obviously the iterative nature of the procedure.

Gaitonde (32) devised a non-iterative Schwarz-like procedure for compact schemes, that we call CC6-MB. A schematic of this procedure is shown in Fig. 4.1(b): Gaitonde removed the shared nodes (grey), substituted them with interface-boundary nodes (white) and set the width of the overlap region to 4. The author was concerned with the parallelisation of a FD code based on the sixth-order compact central formula Eq. 3.62 with a five-points stencil, repeated here for the sake of clarity:

$$f'_{i-1} + 3f'_i + f'_{i+1} = \frac{1}{h}\left[\frac{7}{3}(f_{i+1} - f_{i-1}) + \frac{1}{12}(f_{i+2} - f_{i-2})\right].$$

In this equation f is a fluid dynamic variable, f' its derivative and h the (uniform) grid spacing. In Gaitonde's approach, Eq. 3.62 is applied up to node 5 in sub-domain D_1, and from node 6 in sub-domain D_2, and so the effect of the boundary closures at the block-interface is minimised. They are used at the interface-boundary nodes, where the computed solution is overwritten by that computed in the neighbour sub-domain, and therefore they have a non-direct impact only, through the computed solution at internal nodes. Indeed, Gaitonde's method represents a cost-effective approach to the solution of the global problem, as, unlike the Schwarz method, it does not require iterating. Also, it has the same communication cost as a Schwarz iteration: the fluid dynamic variables at four interface-boundary nodes must be communicated between processors to simulate the next time step.

Gaitonde attempted to minimise the effect of the boundary closures at the block-interface while keeping a narrow overlap region, i.e. a low communication cost. His attempt was only partially successful: he found that smooth flow features could be distorted because of the block-interface treatment, and an accurate solution could not be obtained without appropriate filtering. An obvious cure to this problem would be to widen the overlap region, but this would increase the communication cost and ultimately worsen the scalability of the algorithm.

An attempt to improve the block-interface treatment was made by Sengupta *et al.* (33). These authors identified a weak point in the different dissipation characteristics of left and right boundary closures. The opposite signs of the FD coefficients at opposite boundaries are responsible for a lack of dissipation at the left boundary and an excess of dissipation at the right boundary. This is a general issue that becomes critical when domain decomposition is employed. It is responsible for the distortion of the flow features observed at the block-interface in Gaitonde's method. Sengupta *et al.* simulated the propagation of a wave-packet through a multi-block domain: when the packet crosses a block-interface a spurious wave-packet is generated because of the different dissipation properties of the boundary closures employed at the interface-boundary nodes in the neigbouring sub-domain. Sengupta *et al.* solved this problem by performing the compact differentiation along a grid line twice with inverse index orientation. The derivative at a node was then set to the average between the two computed values. Indeed, the resulting compact scheme has symmetric properties and, when applied in a multi-block framework using Gaitonde's procedure, does not generate spurious wave-packets. This multi-block compact solver was tested on a supersonic cone-cylinder geometry. Although the test appeared to be successful, the method has a non-conservative block-interface treatment, and so its general applicability to compressible flow simulations is questionable.

The class of compact-upwind methods described in section 2.1.1 has opened new possibilities for the application of a classical and efficient parallel multi-block strategy to compact schemes. These methods compute a numerical flux function, unlike others which compute derivatives, and so they are more suitable for the developement of a conservative block-interface treatment. Chao *et al.* (35) have exploited this possibility by developing a multi-block compact method. Their parallel algorithm is based on the compact-WENO method proposed by Ren (22), and uses the fifth-order compact formula, Eq. 2.9, for the discretisation of the positive inviscid flux. Eq. 2.9 at the intermediate node $i + 1/2$ reads:

$$9\hat{f}_{i-1/2} + 18\hat{f}_{i+1/2} + 3\hat{f}_{i+3/2} = f_{i-1} + 19f_i + 10f_{i+1},$$

where $\hat{f}$ represents the numerical flux function. A schematic of Chao's procedure is shown in Fig. 4.1(c). Node 6 is shared between D_1 and D_2, and so the solution at this node must be unique for the two sub-domains. Since the derivative at this node depends on the numerical flux at the intermediate nodes, 11/2 and 13/2, the consistency condition is enforced by using the fifth-order explicit formula, Eq. 2.19, at these intermediate nodes, repeated here for clarity:

$$\hat{f}_{i+1/2} = \frac{1}{30}f_{i-2} - \frac{13}{60}f_{i-1} + \frac{47}{60}f_i + \frac{9}{20}f_{i+1} - \frac{1}{20}f_{i+2}.$$

Provided the fluid dynamic variables are consistent between the two sub-domains from node 3 to node 8, the same values of the numerical flux function at the intermediate nodes 11/2 and 13/2 are computed independently in the two sub-domains. Analogous considerations for the negative flux lead to the conclusion that the fluid dynamic variables must be consistent from node 3 to node 9.

Chao's method can be seen as a Schwarz procedure where the original linear system is perturbed at the block-interface, in order to automatically guarantee the consistency of the solution at the shared node and avoid iterating. The overlap region is seven-nodes wide, but the solution at the six interface-boundary nodes only must be exchanged between processors: the seventh node is the shared node, where the consistency is automatically guaranteed. The overlap region is wider than in Gaitonde's method because of the non-symmetric stencils of the upwind formulas, and so the communication cost is slightly higher. On the other hand, these upwind formulas are preferable for the simulation of compressible flows for the reasons discussed in Chapter 2.

Independently from Chao *et al.*, we have conducted a similar effort, starting from the compact-TVD method proposed by Tu and Yuan (25) that we modified as explained in section 2.3.3. We have faced the same basic problem, i.e. the parallelisation of the compact scheme given by Eq. 2.9, employed in Tu and Yuan's compact-TVD method as well as in Ren's compact-WENO method. A schematic of our procedure, which we call CU5-MB, is pictured in Fig. 4.1(d). We do not employ any shared node, and so there is no need to enforce the consistency of the solution between blocks. Instead, we enforce the continuity of the numerical flux function across blocks by using the explicit formula, Eq. 2.19, to compute the numerical flux at the block-interface. This can be seen as a consistency condition: neighbouring sub-domains do not share any node, but they do share an intermediate node; since the numerical flux is computed at the intermediate nodes, it must be consistent between blocks.

Comparing Fig. 4.1(c) and Fig. 4.1(d), the difference between Chao's method and ours is clear. In our procedure, Eq. 2.19 is used at the intermediate node 11/2 only, and the same value of the numerical flux is computed independently in the sub-domains D_1 and D_2, provided the fluid dynamic variables are consistent from node 3 to node 8. Therefore, the overlap region is six nodes wide, one node narrower than in Chao's method. This does not reduce the accuracy, because in Chao's method one node is shared and its computation in both sub-domains is redundant. Having a narrower overlap region does not impact the communication cost of our method but it decreases its computational cost, because one of the sub-domains is narrower. In 1D this is a negligible saving, but in 3D a whole plane of nodes for each block-interface is not required to be computed. However, the most interesting difference is perhaps the smaller impact of the domain-decomposition on the computed solution: the explicit formula is applied at one intermediate node instead of two, therefore a smaller perturbation is introduced on the original linear system.

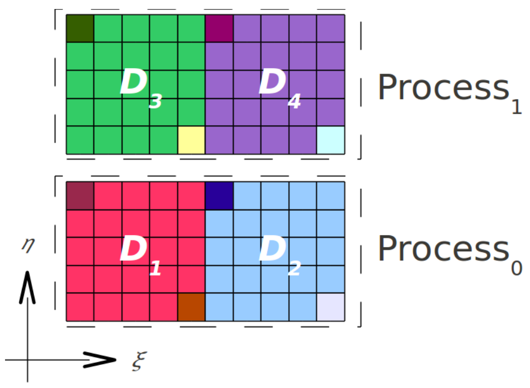
Summarising, both Chao's multi-block method and ours have several advantages with respect to Gaitonde and Sengupta's methods:

- they do not use boundary closures at the block-interface, and so issues like the distortion of smooth flow features and the formation of spurious wave-packets do not arise;
- they are fully conservative, because the single-block method as well as the block-interface treatment is conservative.

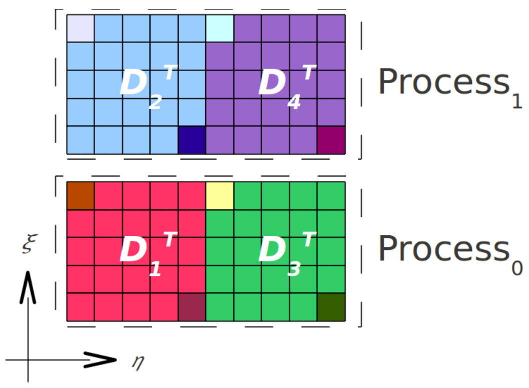
The only disadvantage is the higher communication cost, as the solution at six nodes instead of four is exchanged between neighbouring processors. Comparing Chao's method and ours, we believe that ours better exploits the numerical-flux-based formulation of Eq. 2.9: it does not employ any unnecessary shared node, has a lower computational cost and, more importantly, introduces a smaller perturbation on the original linear system.

4.1.2 Slab/drawer domain decomposition algorithm

The algorithms described in section 4.1.1 solve in parallel the linear system to compute the derivatives using compact FD formulas. They have been explained by reference to a one-dimensional model problem. The extension to a multi-dimensional case is straightforward and involves the parallel solution of several linear systems to compute derivatives along different gridlines and directions.



(a) Processing along ξ-direction



(b) Processing along η-direction

Figure 4.2: Schematic of the data layout in main memory for the distributed-memory slab decomposition algorithm (34).

In a multi-dimensional case a linear system must be solved for each grid line: while processing a certain direction, all grid lines parallel to it can be processed independently in parallel. This idea has been exploited by Laizet *et al.* (34), who have proposed a dual-domain decomposition, namely *slab* decomposition, which changes during a single time step depending on the spatial direction computed. A schematic of the slab decomposition algorithm is in Fig. 4.2. A two-dimensional model problem is considered, where the domain is made of 100 cells: 10 cells per direction. The cell centres are the nodes of the finite-difference discretisation. The work-load is shared between two *processes*. We use the term *process* instead of *processor* to stress the fact that a process could be a task assigned to a core of a multi-core processor.

The algorithm is conceptually very simple: each process is assigned 5 grid lines parallel to the direction ξ, when computing the derivatives along ξ, and 5 grid lines parallel to the direction η, when computing the derivatives along η. A remarkable complexity of implementation results from such conceptual simplicity: in general the processes do not have access to the same memory and therefore global transposition of the variables involved in the computation must be performed each time the processing direction changes. Such complexity is evident even for the model problem represented in Fig. 4.2. When swapping the processing direction from ξ to η, Process$_0$ transposes the variables on sub-domain D_1, sends the variables on sub-domain D_2 to Process$_1$, receives the variables on sub-domain D_3 from Process$_1$ and transposes them.

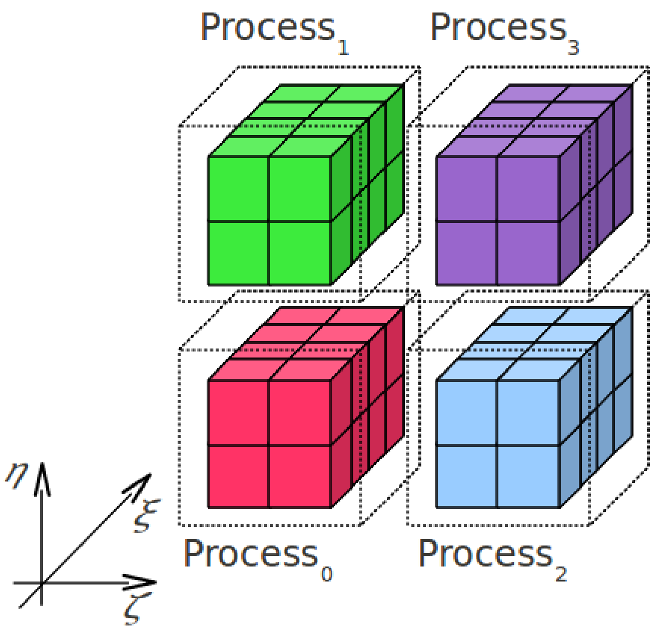
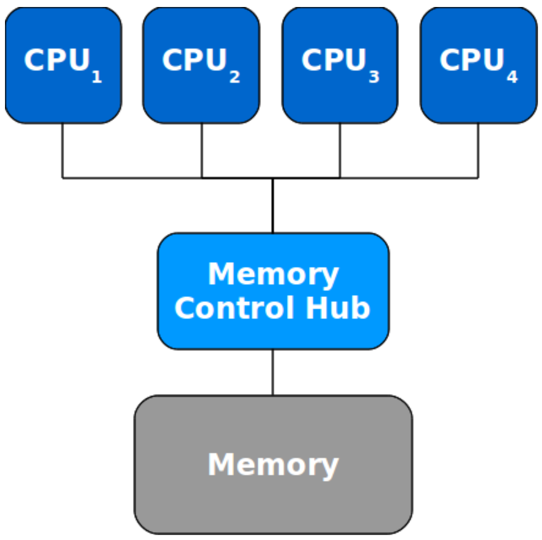
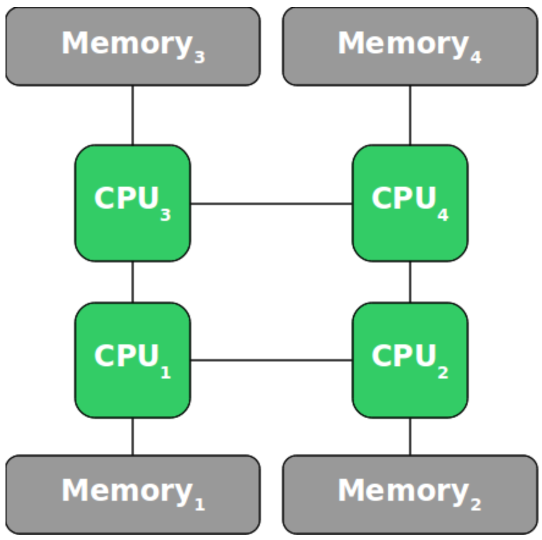


Figure 4.3: Schematic of drawer domain decomposition (74). A model domain of (4×4×4) cells is partitioned in 4 sub-domains when ξ-parallel grid-lines are processed.

The global transpose is a very expensive operation. If a 2D computational-grid of $(N_pN)^2$ nodes were assigned to N_p processes, the global transposition operation would require each process to exchange N^2 values with $(N_p - 1)$ processes and transpose N_p matrices of size $(N \times N)$. For the same grid and computational structure, a multi-block algorithm employing an overlap region N_o nodes wide would require a process to exchange $N_o\sqrt{N_p}N$ values with 4 processes. For $N_p \gg 1$, the ration between the communication cost of the slab decomposition and that of a multi-block algorithm is $\sqrt{N_p}N/4N_o$, which evaluetes to about 40 for typical values such as $N_p = N = 100$ and $N_o = 6$. Laizet *et al.* (34) implemented slab decomposition in their incompressible DNS code and reported that in some tests communication took up to 40% of the total simulation time. Nevertheless, the strong scalability of their code is not as bad as one may expect (90% parallel efficiency on 1024 compute-cores for a 4 billion node mesh). This is due to the high computational cost of the algorithm combined with a careful optimisation: the computation time per process is high and so it is comparable to the communication time per process only at high process counts; also, the communication is carefully overlapped with the local transposition of the matrices.



(a) SMP quad-core processor (Intel Xeon)



(b) NUMA quad-core processor (AMD Opteron)

Figure 4.4: Schematics of quad-core processor architecture.

An additional issue with the slab decomposition is that the maximum number of processes is limited by the minimum grid dimension: referring to the model problem in Fig. 4.2, not more than 10 processes (1 per grid line) can be employed, although 100 nodes must be computed. This is an issue, expecially for two dimensional problems. For three dimensional problems, the cap on the process count is the minimum number of nodes in a plane, a much less restrictive constraint. The 3D extension of slab decomposition is usually referred to as *drawer* decomposition (74), because the partitioned domain resembles a set of drawers. The basic principle is that if, for instance, the direction ξ is being processed, the domain can be partitioned along both η and ζ directions. A schematic of drawer domain decomposition is shown in Fig. 4.3.

The main advantage of slab/drawer decomposition is that, unlike the multi-block methods described in section 4.1.1, it does not introduce any perturbation on the original linear system, i.e. parallel and serial solutions are identical.

4.1.3 Hybrid parallelisation of the Navier-Stokes solver

We have parallelised CU5-TVD-NSK exploiting both distributed- and shared-memory models. We refer to the parallel implementation as CU5-TVD-NSK-MB. Two different parallelisation strategies are employed, depending on the memory model.

In a distributed-memory environment, multi-block domain decomposition is used. Our implementation follows the SPMD programming model and uses the Message Passing Interface (MPI) (`www.mcs.anl.gov/research/projects/mpi`). For each stage of the Runge-Kutta method, an MPI-process executes the following tasks:

1. Enforces boundary conditions at domain boundaries.
2. Exchanges fluid dynamic variables at interface-boundary nodes with neighbours.
3. Computes velocity and temperature gradients using the CC6-MB method.
4. Computes stress tensor and heat flux vector at internal nodes.
5. Exchanges stress tensor and heat flux vector at interface-boundary nodes with neighbours.
6. Computes the Navier-Stokes split flux at internal and interface-boundary nodes.
7. Computes the high-order numerical flux function using the CU5-MB method.
8. Applies the characteristic-based TVD limiter to the numerical flux.
9. Computes the time-derivative at internal nodes.

Indeed, tasks 2 and 5 involve communication between processors, and therefore represent the bottle neck for the scalability of the code. Task 5 could perhaps be avoided, and our tests have revealed that it has a negligible effect on the results. Nevertheless, we recommend performing it to enforce more precisely the continuity of the numerical flux across blocks. Overlap between computation and communication has not been implemented and is left as a future developement.

The CC6-MB method is modified to employ six overlap nodes instead of four: these nodes are necessary for the CU5-MB method and so using them in the CC6-MB method improves accuracy at no additional cost. Sengupta's fix (33) is not considered as the CC6-MB method is used only for viscous quantities.

In a shared-memory environment the slab/drawer domain decomposition is used. Our implementation follows the SMP programming model and is based on the Open Multi-Processing (OpenMP) interface (`http://openmp.org`). Since the OpenMP processes, or *threads*, have access to a shared memory-space, **data transposition is not**

necessary and therefore is not performed: this simplifies the algorithm, whose implementation reduces to careful unrolling of loops over the grid-lines.

Tasks 1, 3, 4 and 6–9 are performed in parallel by the available threads, each one assigned a certain number of grid-lines parallel to the processing direction. The multi-thread execution follows a *fork-join* model: the master thread executes the serial tasks; a team of threads is created before executing a parallel task and destroyed when the task is completed. This model presents two issues for code scalability: first, the serial tasks do not scale with the number of threads; second, thread creation and destruction introduce an overhead. The first issue is not important in our case, because all tasks but those involving MPI communication can be performed in parallel. We have addressed the second issue by compounding tasks 6–8 in a single OpenMP parallel region. Tasks 1, 3, 4 and 9 are relatively light and the overhead of thread creation and destruction essentially nulls the speed-up of the multi-thread execution, as seen in the test discussed in section 6.1.4. A third, and more important, issue of the multi-thread parallelisation is that the SMP programming model may not reflect the processor architecture. Fig. 4.4 compares two common implementations of a quad-core processor. Fig. 4.4(a) is a schematic of a true SMP architecture, where several CPUs share a Memory Control Hub (MCH) to access the same main memory. Memory latency and bandwidth are *symmetric*, i.e. they are the same for all CPUs. Fig. 4.4(b), instead, is a schematic of a Non-Uniform Memory Access (NUMA) architecture. Each CPU has its own memory attached and memory sharing is implemented at software level. NUMA architectures have the following advanges over the SMP ones: access of a CPU to its own memory is direct, and so quicker; CPUs do not compete to use a shared MCH. On the other hand, memory latency and bandwidth are strongly non-symmetric: accessing Memory$_2$ and Memory$_3$ for CPU$_1$ is slower than accessing Memory$_1$, and accessing Memory$_4$ is even slower. In a fork-join execution, assuming the master thread is assigned to CPU$_1$, most data is likely to be stored in Memory$_1$; this means that during the parallel execution all CPUs compete for the use of the memory controller of CPU$_1$. No effort has been made to tackle this problem, as we do not target one architecture in particular.

The shared-memory parallel algorithm is nested into the distributed-memory parallel algorithm in a *funnelled* mode. Each MPI-process has a master thread and several slave threads: all of the threads process local data in parallel, but the master thread only is allowed to communicate with other MPI-processes.

4.2 GPU-accelerated Navier-Stokes solver

The GPU architecture is very *stiff* to program: great performance can be achieved if the code implementation meets certain requirements, but there is modest or no gain in speed with respect to CPUs if such requirements are missed. This point is explained in section 4.2.1.

In order to comply with these strict requirements, we had to write a new code, different from CU5-TVD-NSK, that we call CUDA-CU5-TVD-NS. This code uses a slighly different numerical method and the differences have been discussed in section 3.3.

CUDA-CU5-TVD-NS is tailored for the GPU architecture and its development represents, in our opinion, an interesting case study. In section 4.2.2 we give some implementation details. These details are the key to understanding how it is possible to obtain the results shown in section 6.2.3, and may form a useful guideline for scientific programmers addressing similar problems.

4.2.1 GPU hardware and programming models

In this section we describe the GPU hardware and programming models, and provide some specifications of the GPU we have used to develop and test CUDA-CU5-TVD-NS: the Nvidia Tesla S1070.

A Tesla S1070 comprises four devices like the one sketched in Fig. 4.5. Each device comprises N multi-processors ($N = 30$ for the Tesla S1070). Each multi-processor comprises M scalar processors ($M = 8$ for the Tesla S1070 as well as most GPUs). These processors are slow compared to CPU cores (for the Tesla S1070 they are clocked at 1.44 GHz versus 2–3 GHz typical of modern CPUs) and are managed by a single instruction unit: this means that they work in parallel only if they are able to execute the same instruction.

The programming model corresponding to this architecture is called Single Instruction Multiple Thread (SIMT). A procedure executing a SIMT program is called a *kernel*. A kernel is executed in parallel by creating a computational structure made of several *thread-blocks*. Threads within a thread-block are organised in a one-, two- or three-dimensional structure. The thread-blocks are organised in a one- or two-dimensional structure, referred to as *block-grid*. Indeed, this computational structure maps on the architecture: each block runs on a multi-processor and each thread on a processor. In

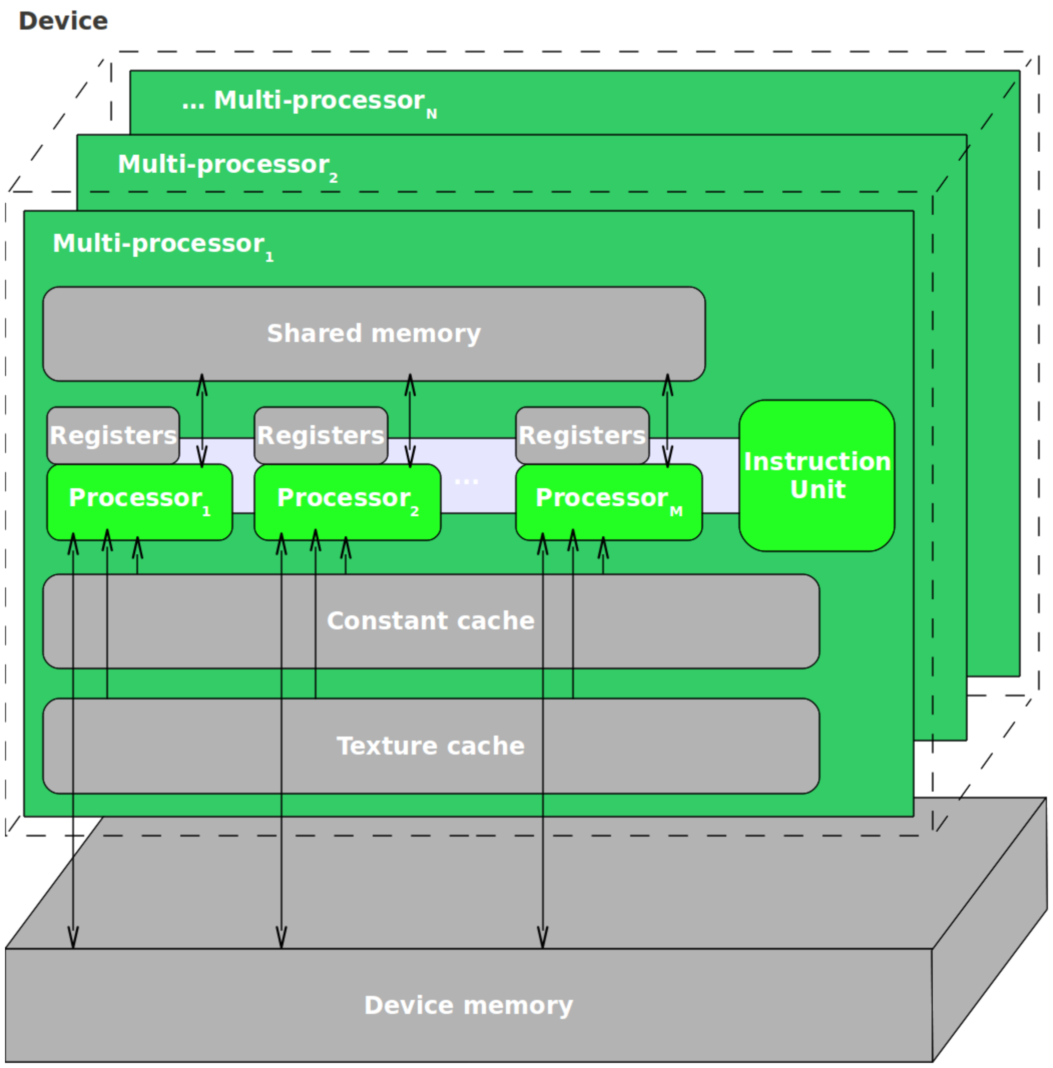


Figure 4.5: Schematic of the GPU architecture.

order to avoid confusion, when talking about GPUs we will use the term *grid* to define an organised structure of thread-blocks, and *computational-grid* to define a set of FD nodes. The SIMT model is a generalisation of the SIMD model adopted to program vector processors: threads may execute independent serial tasks as well as data-parallel instructions. In practice, SIMT programs achieve the best performance when running in a SIMD fashion.

The device has a very deep memory hierarchy, comprising:

- **Registers**. The role of registers is the same as in CPUs. In the Tesla S1070 each processor has 2048 32-bit registers.
- **Shared memory**. This is a low-latency low-capacity memory shared among processors belonging to the same multi-processor. Temporary variables necessary to the kernel execution should be stored here. Each multi-processor has 16 KB of shared memory divided into 16 banks.
- **Constant cache**. This is read-only, shared among processors belonging to the same multi-processor, and can be used to store variables that remain constant during the kernel execution. Each multi-processor has 8 KB of constant cache.
- **Texture cache**. This is read-only and shared among processors belonging to the same multi-processor. It has a built-in linear interpolation mechanism and so it can be used to compute fast one-, two- and three-dimensional linear interpolations on data that remain constant during the kernel execution. Each multi-processor has 8 KB of texture cache.
- **Device memory**. This is a high-latency high-capacity memory, shared among multi-processors belonging to the same device. It plays the same role as the main memory does in CPUs. The difference between the two is that the device memory is not cached, and so any expression making explicit reference to a variable residing in device memory results in an expensive memory transaction. The Tesla S1070 has 4 GB of device memory per device.

A multi-processor can perform 8 accesses to either shared or device memory per clock cycle. On the other hand, accessing the shared memory has zero latency, while accessing the device memory has a latency of 400–800 clock cycles. It follows that most

GPU performance issues are related to the device memory access. Indeed, algorithms amenable to GPU computing are those with high *arithmetic intensity*, defined as the ratio of the number of arithmetic operations to the number of memory transactions. This is by far the most important factor impacting the kernel performance, and is related to the nature of the task the kernel executes. Other fundamental guidelines for performance are given in the following, in order of importance.

1. **Use the shared memory as workspace**
   Using the shared memory as workspace aims to minimise the device memory access. A typical kernel workflow is:

   - load data from device memory to shared memory;
   - synchronise all threads within a block to ensure each thread can safely read the data loaded by other threads;
   - process data in shared memory;
   - synchronise threads again to ensure all data has been written to shared memory;
   - write data back to device memory.

   The synchronisation steps can be removed, with performance gain, if the threads work on independent chunks of data.

2. **Issue coalesced device memory transactions**
   A multi-processor can access up to 16 32-bit words (16 single-precision floating-point numbers, for example) residing in the device memory, with a single memory transaction. Such a memory transaction is said to be *coalesced* and takes place only if certain constraints are satisfied. These constraints depend on the *compute capability* of the device, that is an index used to identify different classes of GPUs. For the least capable devices, the constraints are verified if 16 consecutive threads within a block access, in order, 16 words stored in contiguous memory locations. For scientific programming, where data is arranged in large arrays, satisfying the coalescing constraints usually means mapping thread-blocks on contiguous segments of arrays. This concept is illustrated in Fig. 4.6. If the coalescing constraints are not verified, 16 separate memory transactions are issued, one per word, with dramatic (negative) impact on performance.

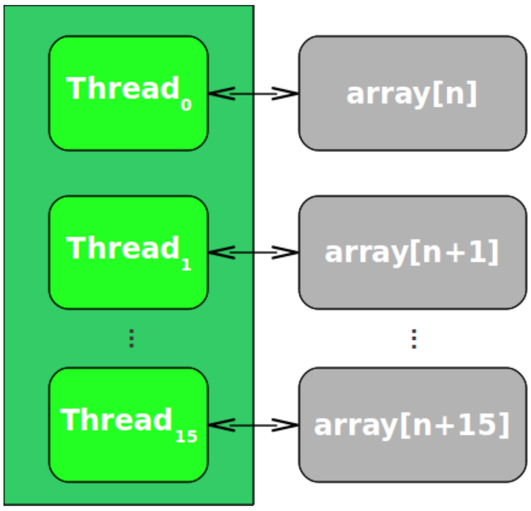


Figure 4.6: Memory access pattern for coalesced device memory transactions.

3. **Maximise multi-processor occupancy**

   The slow device memory access can be hidden by overlapping computation and device memory transactions. This is accomplished by assigning multiple threads to a single processor. Suppose a block of 16 threads is assigned to a multi-processor so that each of the 8 processors is assigned two threads. First, the 8 processors load data from device memory for threads 0–7. Then, the data loading for threads 8–15 is overlapped with the computation for threads 0–7. Indeed, a block-size of 16 threads is sub-optimal, because a memory transaction comprises 8 words at best (i.e. if it is coalesced). Therefore, the minimum block-size for performance is 32 threads, also called a *warp*: in this case, coalesced memory transactions comprise 16 words and memory transactions for threads 16–31 are overlapped with computation for threads 0–15. In fact, the instruction unit schedules and manages threads in warps, and so the block-size should always be a multiple of 32.

   A multi-processor can run many warps concurrently. The maximum number of active warps per multi-processor is a hardware limitation and is 32 for a Tesla S1070. The *occupancy* for a certain kernel is defined as the ratio of the number of active warps per multi-processor during the kernel execution to the maximum number of active warps. The number of active warps during the kernel execution depends on how much of the memory resources (registers and shared memory) the kernel allocates per thread. Having high occupancy is expecially important for kernels with low arithmetic intensity, but it is secondary for kernels with high arithmetic intensity, where the amount of computation on a few active warps may be already large enough to hide the memory latency.

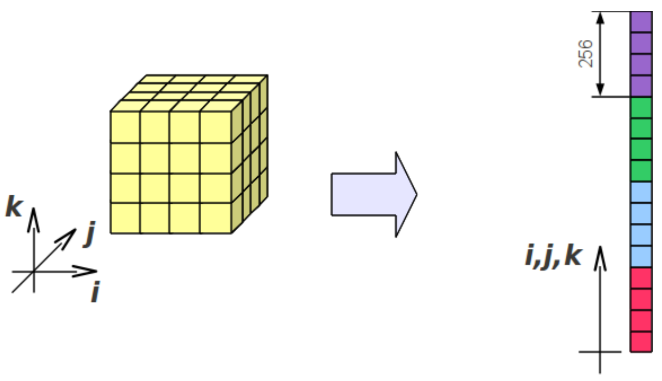
4. **Minimise shared-memory bank-conflicts**
   The shared memory is organised in 16 banks of 1 KB capacity each. As for the device memory, 16 32-bit words can be accessed in a single memory transaction if certain constraints are satisfied. For shared memory, the 16 words must reside in different banks. If not, 16 separate memory transactions are issued. Shared memory allocation complies with the following rule: 16 contiguous 32-bit words are stored by default in 16 consecutive memory banks. Methods to realise memory access free from bank conflicts are strongly algorithm dependent.

4.2.2 Key implementation details of the GPU Navier-Stokes solver

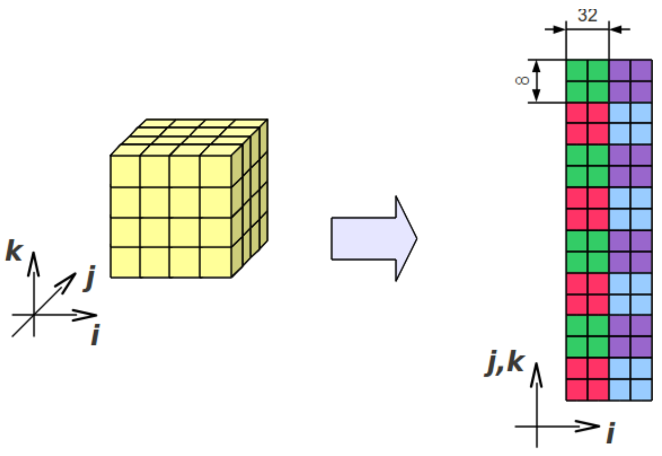
Assuming a computational grid of $(N_\xi \times N_\eta \times N_\zeta)$ nodes, the pseudo-code for CU5-TVD-NSK reads:

```
for $k$ from 1 to $N_\zeta$
  for $j$ from 1 to $N_\eta$
    process $\xi$-parallel grid-line $(j, k)$
  end
end
for $i$ from 1 to $N_\xi$
  for $k$ from 1 to $N_\zeta$
    process $\eta$-parallel grid-line $(k, i)$
  end
end
for $j$ from 1 to $N_\eta$
  for $i$ from 1 to $N_\xi$
    process $\zeta$-parallel grid-line $(i, j)$
  end
end
```

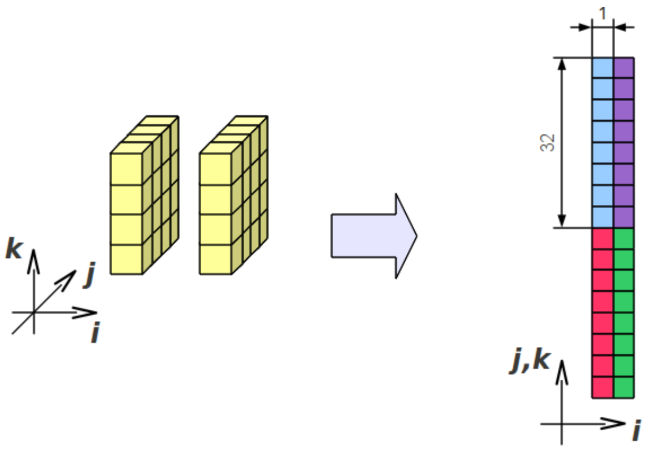
This code structure is unsuitable for GPU acceleration. The GPU has a many-core architecture and one may be tempted to assign each core one or more grid lines to process, in a drawer-decomposition fashion. This approach would not work for two reasons:



(a) Kernels approaching the mesh as a set of nodes.



(b) Kernels approaching the mesh as a set of grid-lines.



(c) Kernels approaching the mesh as a set of planes.

Figure 4.7: Thread-blocking strategy for different kernels. Each colour identifies a multiprocessor. The dimension-lines indicate the dimensions of thread-blocks.

- access to the device memory would be coalesced for one dimension and serialised for the other two;
- processing a grid line requires much temporary data, which does not fit in the shared memory and would be allocated in the device memory.

Alternatively, one may think of having all cores processing in parallel the same grid-line. This approach would not work either, because:

- a single grid-line would not supply the GPU with enough data and computation to exploit its potential;
- the overhead of repeated kernel launches (one per grid line per dimension) would significantly increase the runtime.

This is why we had to re-engineer the code from its very basics, in order to make its structure amenable to GPU computing. The new code, CUDA-CU5-TVD-NS, is written in C language to expose the data layout and favour structured programming. In the following we present its key implementation details by systematically discussing how the guidelines given in section 4.2.1 impact its structure.

1. **The shared memory is used as workspace**
   This requirement has a fundamental effect on the code structure. The shared memory is relatively small (16 KB per multi-processor) and therefore a complicated CFD algorithm, with several MB of temporary data, cannot be carried out in a single kernel. Instead, the algorithm is split into several elementary tasks, each one coded into a separate kernel. Indeed, structured programming is more appropriate in this case than object-oriented programming. The tasks are highly data-parallel, because they are performed for all of the nodes of the computational grid. In this case, the SIMT kernels run in a SIMD fashion, for maximum performance.

   We use single-precision arithmetic. The Tesla S1070 is capable of double-precision arithmetic, but the computing-speed is significantly lower than for single-precision. Furthermore, in double-precision the memory requirement doubles, leading to occupancy issues. On the other hand, Hagen *et al.* (54) simulated a shock-jet interaction both in single- and double-precision using a compact-WENO scheme and their results were not affected by the reduced precision. We have validated our code against third-party double-precision numerical results and experimental data, and some of these tests are presented in sections 6.2.1 and 6.2.2.

The kernels are written in CUDA and have a C Application Programming Interface (API). They are collected in a library, that we call CUDA-CFD. It comprises the following procedures:

- `cudaCons2prim()`: computes primitive fluid dynamic variables from the conserved variables.
- `cudaSplitFlux()`: computes the inviscid kinetic split flux.
- `cudaTVD()`: applies the characteristic-based TVD limiter to the numerical flux function.
- `cudaODErhs()`: computes the spatial derivative from the numerical flux function.
- `cudaViscosity()`: computes the viscosity from the temperature.
- `cudaNSCBC()`: enforces the boundary conditions (NSCBC).
- `cudaSdmv()`: executes the component-by-component vector operation: $w_i = \alpha w_i + \beta u_i v_i$, where u, v, w are vectors and α, β scalar constants.

Other tasks can be expressed as algebraic vector-vector operations. The CUBLAS library, provided with the CUDA Toolkit (`http://developer.nvidia.com/object/cuda_archive.html`), is a collection of CUDA kernels with C API performing algebraic matrix and vector operations. It is a highly optimised GPU version of the well known Basic Linear Algebra Subprograms (BLAS) library (`www.netlib.org/blas`). The CUBLAS subroutines `cublasSaxpy()`, `cublasScopy()` and `cublasSscal()` are extensively used in our code.

The tri-diagonal matrices for the compact reconstructions are inverted before the beginning of the time loop, and the inverse matrices are used in the time loop. A matrix-vector multiplication involves more floating-point operations than the inversion of a tri-diagonal system, and our choice is motivated by convenience.

The development of a fast GPU tri-diagonal solver is not a trivial matter and it is still a research topic (75, 76). The CULA library (`www.culatools.com`) is a GPU version of the well known Linear Algebra Package (LAPACK) (`www.netlib.org/lapack`). Although LAPACK includes a tri-diagonal solver, CULA does not but it is reasonable to believe that one will be developed: our code could be easily modified in the future to use such a solver. On the other hand, highly optimised subroutines for matrix-vector and matrix-matrix products are available in the CUBLAS library: `cublasSgemv()` and `cublasSgemm()`, respectively. Finally, the test presented in section 6.2.3 shows that computing the compact reconstruction using the inverse matrix is relatively inexpensive with respect to other tasks, and so further speed-up of this task will only have a minor impact on the CFD code runtime.

2. **Device memory access is coalesced**
   Several tasks approach the computational-grid as a collection of grid-lines parallel to the processing direction. This is the case with the compact reconstruction, the TVD limiting and the computation of the spatial derivatives from the numerical flux. The boundary condition enforcement, instead, approaches the computational-grid as a collection of planes orthogonal to the processing direction. All of these tasks require different data layouts for different processing directions in order to achieve coalesced device memory access. This is accomplished by transposing data on swapping processing direction.

   Source code and documentation of an optimised GPU matrix transpose are included in the Nvidia GPU Computing Software Developement Kit (`http://developer.download.nvidia.com/compute/cuda/sdk/website/samples.html`). We have wrapped part of that source code into a procedure with C API, that we have called `cudaTranspose()`. Data transposition introduces additional computation, but it is largely justified by the performance gain due to the coalesced device memory access. Data transposition also allows us to perform a compact reconstruction for the whole computational-grid with one call to `cublasSgemm()`, instead of calling `cublasSgemv()` once per grid-line. This delivers a great improvement in performance, not only because the overhead of the kernel call is minimum but also because `cublasSgemm()` has much higher occupancy than `cublasSgemv()`.

Provided the data is properly laid out in memory, an appropriate thread-blocking strategy must be adopted to ensure coalesced memory access. Since our kernels run in a SIMD fashion, thread-blocks map onto sub-sets of the computational-grid (the structured-blocks used in domain-decomposition algorithms such as those discussed in section 4.1.1). From this point of view, the GPU-accelerated tasks can be divided into three groups.

Tasks belonging to the first group approach the computational-grid as a collection of nodes. The kernels corresponding to these tasks are: `cudaCons2prim()`, `cudaSplitFlux()`, `cudaViscosity()` and `cudaSdmv()`. A simple thread-blocking strategy is adopted for these kernels: the data is seen as a 1D array, and so it is partitioned in a 1D grid of 1D thread-blocks. In this case, achieving coalesced memory access is trivial and there is no need for any particular data layout. This concept is illustrated in Fig. 4.7(a).

Tasks belonging to the second group approach the computational-grid as a collection of grid-lines parallel to the processing direction. The kernels corresponding to these tasks are `cudaTVD()`, `cudaODErhs()` and `cudaTranspose()`. The data is seen as a 2D array, and so it is partitioned in a 2D grid of 2D thread-blocks. This concept is illustrated in Fig. 4.7(b).

The data layout in the device memory must be such that nodes on a grid-line parallel to the processing direction map onto contiguous memory locations. If so, coalesced data access is ensured by setting the first thread-index to run parallel to the processing direction (direction i in Fig. 4.7(b)).

Finally, in the third group is `cudaNSCBC()` only. Enforcing the boundary conditions is a task approaching the computational-grid as a collection of planes orthogonal to the processing direction. The data layout must be such that nodes belonging to a plane orthogonal to the processing direction map onto contiguous memory locations. The domain comprises the first and last plane only, and it is partitioned in a 2D grid of 1D thread-blocks. This concept is illustrated in Fig. 4.7(c). Coalesced memory access is realised setting the thread index to run orthogonal to the processing direction (in Fig. 4.7(c), assuming (j, k, i) layout and column-major ordering, the thread index runs parallel to j and within (j, k) planes).

3. **The occupancy is maximised**
   Occupancy data for our kernels are shown in Table 4.1. When discussing occupancy issues, it is useful to group the kernels by the thread-blocking strategy.

   The kernels employing a 1D grid of 1D thread-blocks are the easiest to optimise. They are `cudaCons2prim()`, `cudaSplitFlux()`, `cudaViscosity()` and `cudaSdmv()`. The block size is chosen to maximise the occupancy, and ranges from 128 to 256. All of these kernels have 100% occupancy except `cudaSplitFlux()`: this kernel performs a more demanding task and uses more registers than the others. Therefore, for this kernel the occupancy is limited by the maximum number of registers available.

   The kernels employing a 2D grid of 1D or 2D thread-blocks present a greater challenge. These are `cudaODErhs()`, `cudaTranspose()`, `cudaTVD()` and `cudaNSCBC()`. The first block-dimension is set to the warp size and the second block-dimension is chosen to maximise the occupancy. Both `cudaODErhs()`, `cudaTranspose()` have an optimum block-size of (32×8), achieving 100% and 75% occupancy, respectively. The different occupancy is due to `cudaTranspose()` using more shared memory per block. The procedures `cudaTVD()` and `cudaNSCBC()` are the most problematic, because the characteristic decomposition uses much temporary data. For both kernels, a warp requires nearly half of the shared memory to run; therefore, only two active warps per multi-processor are allowed, achieving 6% occupancy. We set the block-size to (32×1), but a block-size (32×2) would achieve exactly the same performance, while a block-size $(32 \times N)$, with $N > 2$, would cause the kernel to fail with a segmentation fault.

4. **Shared memory access is free of bank conflicts**
   We explain how bank-conflict-free access is achieved with three examples.

   A scalar variable like the density is allocated in shared memory as a 1D array:

   ```
   __shared__ float density[SIZE];
   ```

   Each thread stores its own value of the density as a component of this array, and so `SIZE` is the total number of threads within a thread-block, which in turn is a multiple of the warp-size. Since a single-precision floating-point number is

Procedure	Threads per block	Registers per thread	Shared memory per block	Occupancy
`cudaCons2prim()`	256	10	88 B	100%
`cudaSplitFlux()`	128	21	112 B	66%
`cudaTVD()`	32	50	6520 B	6%
`cudaODErhs()`	256	12	56 B	100%
`cudaViscosity()`	256	5	40 B	100%
`cudaNSCBC()`	32	62	7836 B	6%
`cudaSdmv()`	128	7	68 B	100%
`cudaTranspose()`	256	10	4264 B	75%

Table 4.1: Occupancy for the custom kernels employed by the GPU Navier-Stokes solver.

a 32-bit word, threads within half-warp access 16 different banks, with no bank conflicts.

In a similar fashion, a vector variable like the velocity is allocated as a 2D array:

```
__shared__ float velocity[SIZE][DIMS];
```

The array is stored in row-major order, so each thread stores its velocity components in `DIMS` successive banks. Since there are $16 = 2^4$ banks, if `DIMS` is odd, threads within the half-warp access 16 different banks. If `DIMS` is even, the array is set to occupy more memory than needed:

```
__shared__ float velocity[SIZE][DIMS+1];
```

This is a standard technique often referred to as *padding*.

A tensor variable like the eigenvector matrix is allocated as a 2D array:

```
__shared__ float eigVecMatrix[SIZE][DIMS*DIMS];
```

Each thread stores its matrix as a 1D array of `DIMS*DIMS` components, and appropriate indexing is used to map the matrix on this 1D array. With regard to the memory access, the same considerations as for vector variables hold.

4.3 Summary

Novel contributions presented in this chapter are:

- An algorithm to parallelise our compact-TVD method. The algorithm is based on structured-block partitioning and improves previous algorithms that employ the same approach (32, 33, 35). Our algorithm is fully conservative and does not distort smooth flow features, although introduces a perturbation on the original linear system to compute derivatives. The effect of this perturbation is assessed in section 6.1.

- A strategy to parallelise our compact-TVD method over a cluster of multi-core processors. We propose to combine our multi-block parallel algorithm with the slab/drawer decomposition algorithm: the small communication cost of the multi-block algorithm makes it ideal to divide the work between processors; the slab/drawer decomposition is used to divide the work between the cores of a processor. The performance of the hybrid algorithm is assessed in section 6.1.4.

- A strategy to parallelise our compact-TVD method in the complex computing environment of a GPU. Our strategy consists of two steps:

  - algorithm break-down into elementary tasks;
  - adoption of a task-dependent domain partitioning.

  Breaking the algorithm into elementary tasks allows us to meet the tight memory constraints of the GPU. The runtime of each task is minimised by choosing the appropriate domain partitioning. The performance of this algorithm is assessed in section 6.2.3.

5

Validation of the compact-TVD method

In this chapter we assess the accuracy of the inviscid flow solver that was the subject of Chapter 2, and validate the viscous flow solver that was described in Chapter 3. The simulations have been carried out using the CPU code CU5-TVD-NSK, whose main features are:

- differentiation of the Navier-Stokes kinetic split flux through a compact-TVD scheme;
- 2D-axisymmetric formulation;
- cell-centred FD discretisation;
- boundary conditions enforcement by extrapolating fluid dynamic variables at extra-boundary nodes.

The simulation of inviscid flows is the subject of section 5.1. Tu and Yuan (25) have presented several standard 1D and 2D inviscid test cases, and we have repeated most of them. However, in this chapter we just show three inviscid tests, delivering new information with respect to Tu and Yuan's analysis. The first test case is a shock tube problem. The results for this test are shown in Tu and Yuan's work, and we present ours here for two reasons. First, the consistency between our results and theirs confirms the correctness of our implementation. Second, as mentioned in section 2.3.2, by changing some of the techniques the compact-TVD method relies on, several versions of it can be generated. We use this classic 1D test to assess the impact of these different techniques. The second test case is supersonic flow over a forward-facing step. It is a classic test and we compare our results to time-accurate results obtained by several authors. We use this test to conduct a first resolution study. The third test is the double Mach-reflection of a strong shock. This is a classic test as well, and so we can compare our results to those obtained by other authors with different methods of comparable accuracy. We use this test to conduct a further resolution study.

In section 5.2 we conduct a validation study of the compact-TVD method for viscous flow simulation. Motivation and merits of this study are:

- The compact-TVD method has only been applied previously to the Navier-Stokes equations in a single case, without characteristic decomposition and with the classical operator splitting, by Tu *et al.* (60). Comparison to experimental data or analytical solution has not been shown before. Therefore the validation and assessment of the method for viscous flow simulation is an original contribution of this thesis work.

- Experience with the Navier-Stokes flux splitting technique is extremely limited, so it is interesting to further assess its capabilities. Furthermore, the technique has only been used in combination with second-order MUSCL interpolation. Therefore, to our knowledge, this study is the first application of this splitting in combination with a scheme that is higher than second-order accurate.

The tests are presented in order of complexity. We start with a one-dimensional test case, namely the simulation of a shock layer, for which an analytical solution exists. We then proceed to consider a two-dimensional test case, namely the supersonic boundary layer over a flat plate. For this case an analytical solution exists as well. Then, we consider the interaction between an oblique shock wave and a supersonic boundary layer. This interaction results in a very complex flow-field, for which experimental results are available. Finally, we present a test case representative of a typical hypersonic simulation: the hypersonic flow past a blunted-cone. For this case experimental results are also available.

5.1 Simulation of inviscid flows

Our simulations have been run at Courant number $CFL = 0.4$. On a 2D finite-difference grid with nodes indexed by (i, j), the Courant number is defined as $CFL = \max(CFL_\xi, CFL_\eta)$, where:

$$\begin{aligned} CFL_\xi &= \max_{i,j}\left(V_{i,j} \cdot \nabla\xi_{i,j} + a_{i,j}|\nabla\xi_{i,j}|\right)\tfrac{\Delta t}{\Delta\xi}, \\ CFL_\eta &= \max_{i,j}\left(V_{i,j} \cdot \nabla\eta_{i,j} + a_{i,j}|\nabla\eta_{i,j}|\right)\tfrac{\Delta t}{\Delta\eta}. \end{aligned}$$

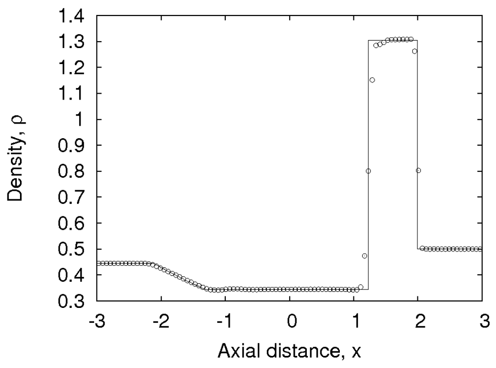
In this definition of CFL_ξ and CFL_η, V is the velocity vector, $\nabla\xi$ and $\nabla\eta$ the metrics, a the speed of sound, $\Delta\xi$ and $\Delta\eta$ the (uniform) grid spacings along ξ and η directions, respectively, and Δt the time step. The adiabatic index was set to $\gamma = 7/5$.

5.1.1 The Lax shock tube

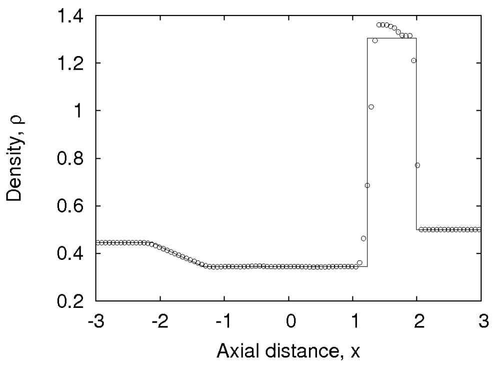
The first test case is the one-dimensional Riemann problem proposed by Lax (77). A diaphragm placed at $x = 0$ separates two regions where a gas is at different states. In the Lax problem the left state is $\rho = 0.445$, $u = 0.698$, $p = 3.528$, and the right state is $\rho = 0.5$, $u = 0$, $p = 0.571$. At time $t = 0$ the diaphragm is removed and the two states are free to interact. This test case is a good benchmark for shock capturing methods because a strong contact discontinuity develops, which is usually smeared over several cells. The state of the system at $t = 0.8$ has been computed on a grid of 100 cells, equally spaced over the domain $x \in [-3, 3]$. Comparison of density, velocity and pressure profiles with the exact solution is shown in Figs. 5.1–5.3.

The method is shown to be very accurate, delivering a sharp description of discontinuities, capturing the shock wave in two points and the contact surface in three points. A minor problem appears at the tail of the left-running expansion fan, where the method fails to completely suppress a spurious oscillation. This is barely visible in the density profile (shown also by Tu and Yuan (25)), while it is clear in the pressure and velocity profiles (not shown by Tu and Yuan). Modifications of the original method proposed by Tu and Yuan may be considered: changing the flux splitting technique, the limiter, and the characteristic decomposition. We have investigated the effect of these and the following considerations arise from the analysis of Figs. 5.1–5.3.

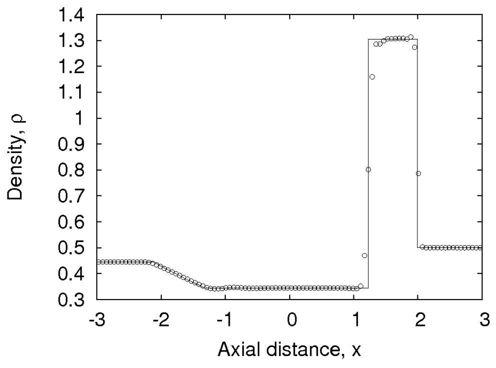
The characteristic decomposition is a key feature of the method. If the flux is limited component-wise, i.e. no projection into characteristic space is performed, spurious low-amplitude oscillations appear in the velocity and pressure, and a strong overshoot



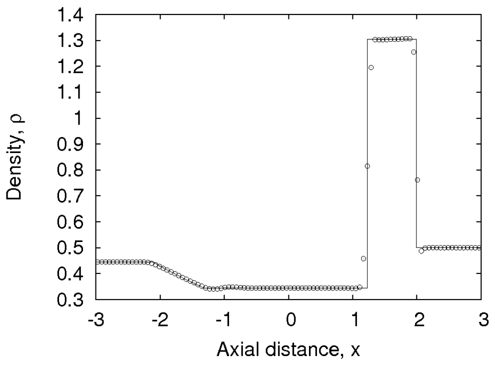
(a) Steger-Warming FVS, limiter B, upwind characteristic decomposition.



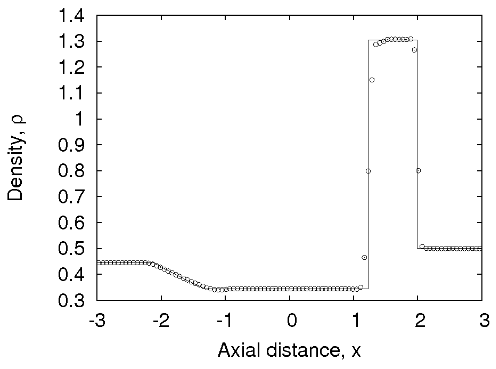
(b) Steger-Warming FVS, limiter B, no characteristic decomposition.



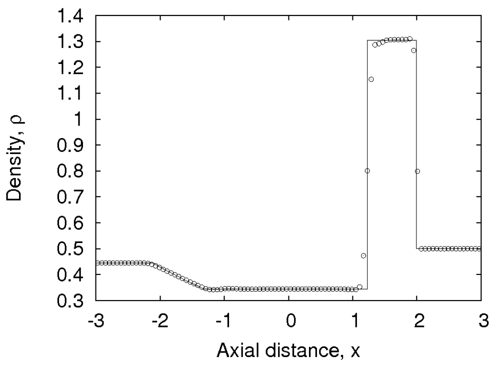
(c) Steger-Warming FVS, limiter A, upwind characteristic decomposition.



(d) Steger-Warming FVS, limiter B, central characteristic decomposition.

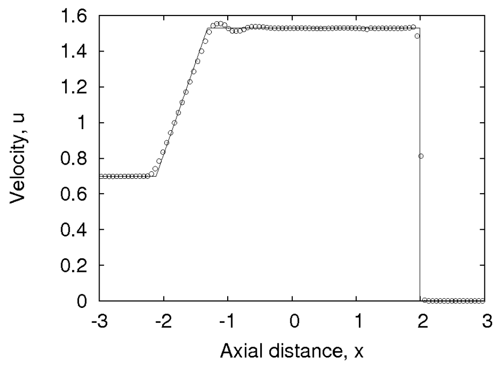


(e) Van Leer FVS, limiter B, upwind characteristic decomposition.

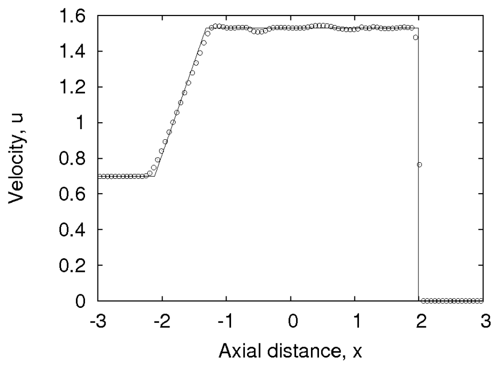


(f) Kinetic FVS, limiter B, upwind characteristic decomposition.

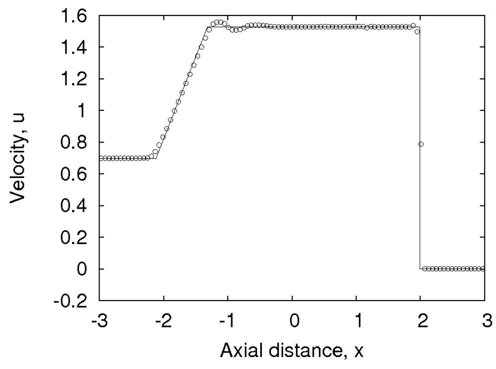
Figure 5.1: Density at time $t = 0.8$ for the Lax problem: —, analytical solution; ○, numerical solution.



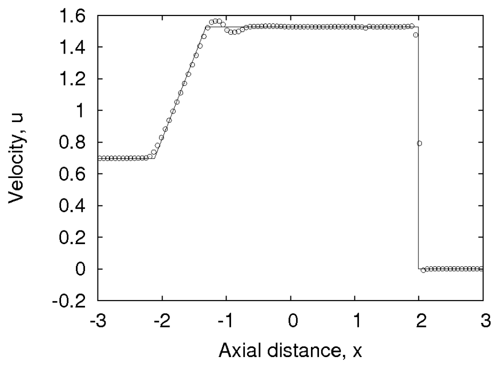
(a) Steger-Warming FVS, limiter B, upwind characteristic decomposition.



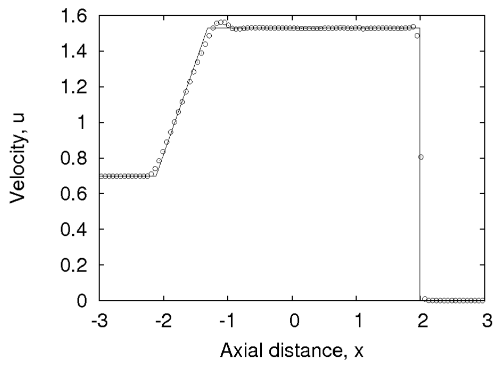
(b) Steger-Warming FVS, limiter B, no characteristic decomposition.



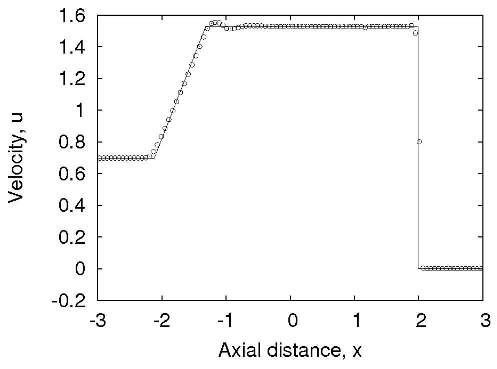
(c) Steger-Warming FVS, limiter A, upwind characteristic decomposition.



(d) Steger-Warming FVS, limiter B, central characteristic decomposition.

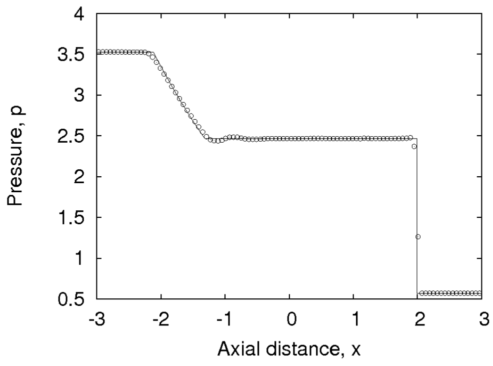


(e) Van Leer FVS, limiter B, upwind characteristic decomposition.

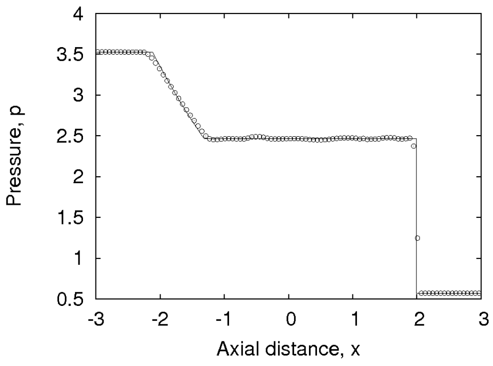


(f) Kinetic FVS, limiter B, upwind characteristic decomposition.

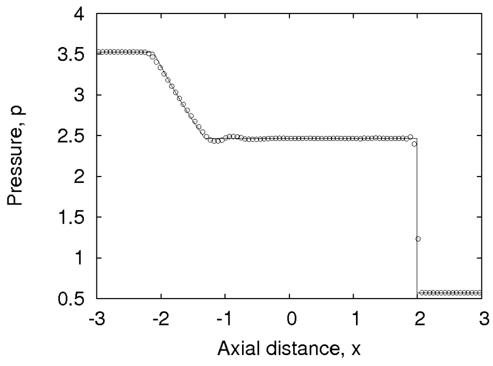
Figure 5.2: Velocity at time $t = 0.8$ for the Lax problem: —, analytical solution; $\circ$, numerical solution.



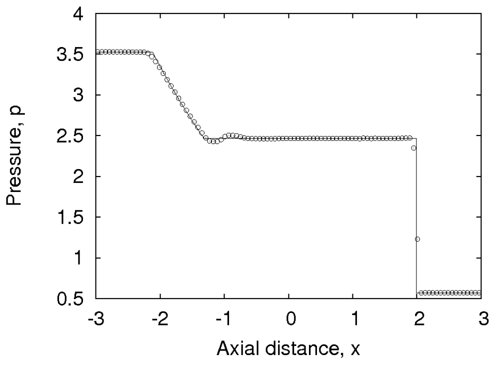
(a) Steger-Warming FVS, limiter B, upwind characteristic decomposition.



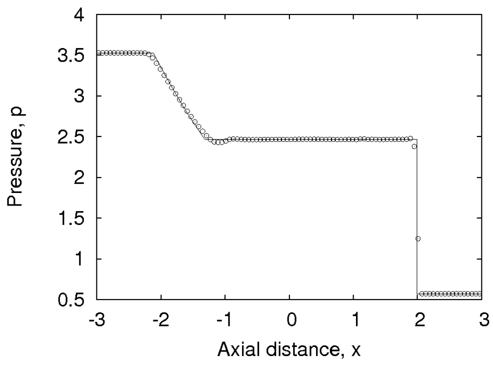
(b) Steger-Warming FVS, limiter B, no characteristic decomposition.



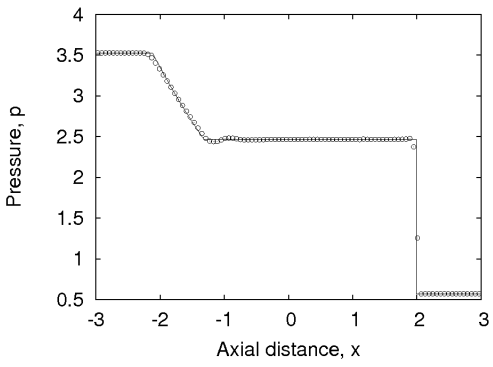
(c) Steger-Warming FVS, limiter A, upwind characteristic decomposition.



(d) Steger-Warming FVS, limiter B, central characteristic decomposition.

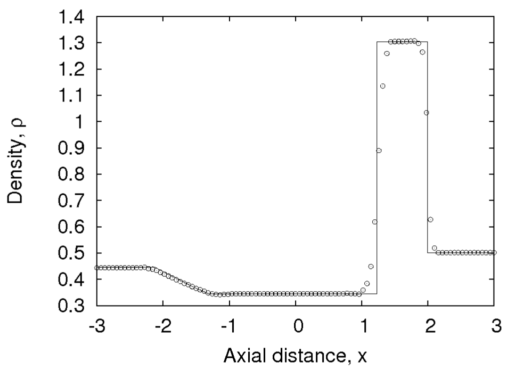


(e) Van Leer FVS, limiter B, upwind characteristic decomposition.

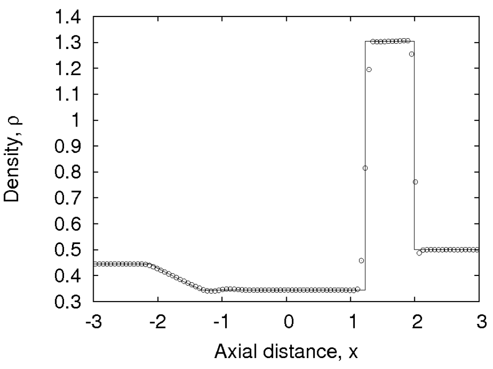


(f) Kinetic FVS, limiter B, upwind characteristic decomposition.

Figure 5.3: Pressure at time $t = 0.8$ for the Lax problem: —, analytical solution; $\circ$, numerical solution.



(a) Fifth-order WENO



(b) Fifth-order compact + TVD.

Figure 5.4: Density at time $t = 0.8$ for the Lax problem: —, analytical solution; ○, numerical solution.

in the density profile appears between the shock and the contact surface. If the characteristic processing is performed, the upwind evaluation of the eigenvector matrix provides a more diffusive scheme, giving a slightly less accurate description of the constant state between the contact surface and the shock, but also a lower-amplitude spurious oscillation at the tail of the expansion fan.

The effect of the flux vector splitting technique is not as remarkable. However, a slight improvement of the kinetic splitting over van Leer and Steger-Warming in controlling the unphysical oscillation at the tail of the expansion fan is observed. Changing the limiter is shown to have a negligible effect.

In Fig. 5.4, a fifth-order WENO scheme (19) and the current method are compared in terms of predicted density profiles: it can be seen that, even though the WENO scheme is uniformly high-accuracy, our compact-TVD method better captures the shock and the contact discontinuity, and, ultimately, has better agreement with the exact solution.

In the simulations presented in the rest of this thesis, kinetic splitting, upwind characteristic decomposition and type B limiter are used. Upwind characteristic decomposition and type B limiter improve the stability: since we target hypersonic flows, a slightly more dissipative method is likely to operate correctly at higher Mach numbers, where stronger shocks form. The use of the kinetic splitting, instead, is motivated by consistency with the viscous solver. The viscous solver employs a splitting for the

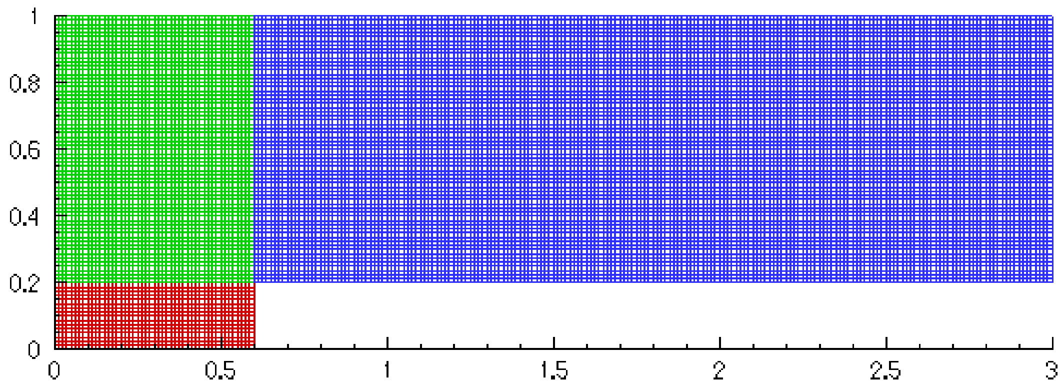


Figure 5.5: Mach 3 forward-facing step in a wind tunnel: the multi-block mesh (three different colours).

Navier-Stokes flux that reduces to the kinetic splitting for the Euler flux in the inviscid case.

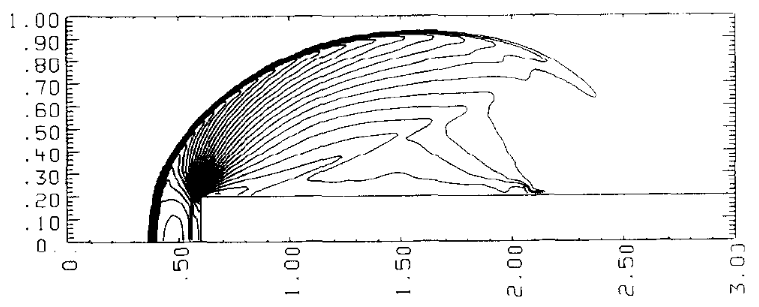
5.1.2 The forward-facing step

The test discussed in this section was identified by Woodward and Colella (78) as a benchmark for compressible high-order numerical methods. The geometry is rather simple: a supersonic wind tunnel with a step on the lower wall. The wind tunnel is 1 length-unit high and 3 length-units long. The step is 0.2 length-units high and it is located at 0.6 length-units from the left end of the tunnel. The free-stream Mach number is $M_\infty = 3$, and the free-stream primitive variables are

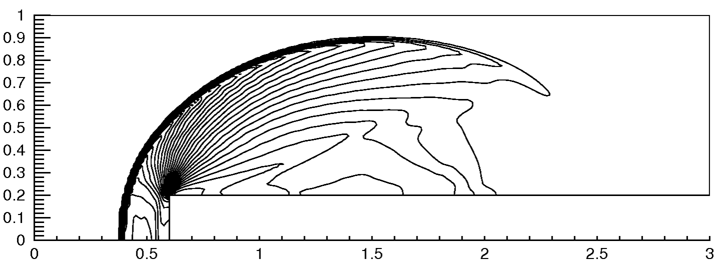
$$\rho_\infty = 7/5, \qquad u_\infty = 3, \qquad v_\infty = 0, \qquad p_\infty = 1.$$

The simulation is started impulsively, i.e. at time $t = 0$ the flow is at the free-stream conditions everywhere. The inlet of the tunnel is a supersonic inflow, and the outlet a supersonic outflow. The tunnel walls are modelled as slip-walls. The domain is partitioned into three blocks as shown in Fig. 5.5, and the multi-block version of the code, CU5-TVD-MB, is used. The grid has uniform spacing $\Delta x = \Delta y = 1/80$.

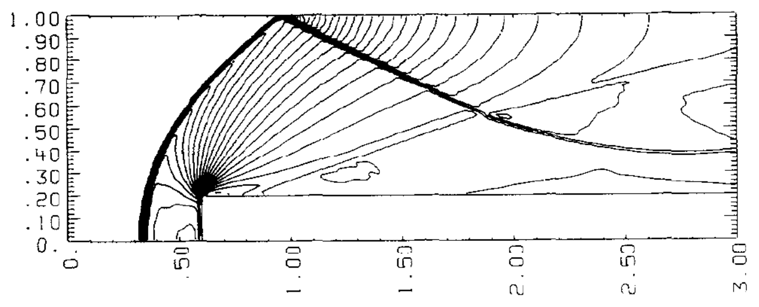
The flow reaches the steady state after about 12 time-units, but the steady flow has a relatively simple structure. For this reason, Woodward and Colella suggested computing the time evolution up to 4 time units, which is characterised by complicated



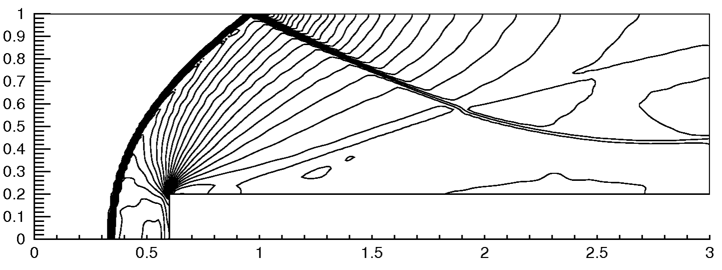
(a) PPMLR, $t = 0.5$.



(b) CU5-TVD-MB, $t = 0.5$.

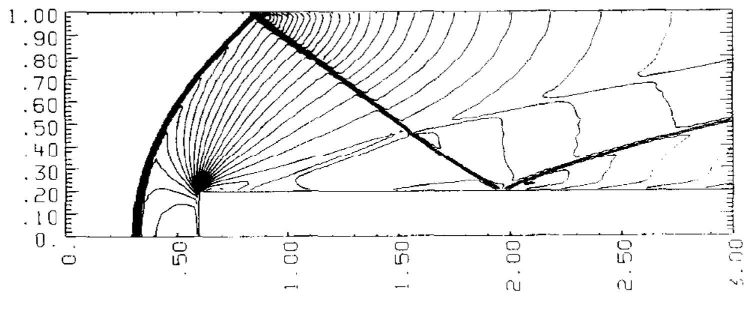


(c) PPMLR, $t = 1$.

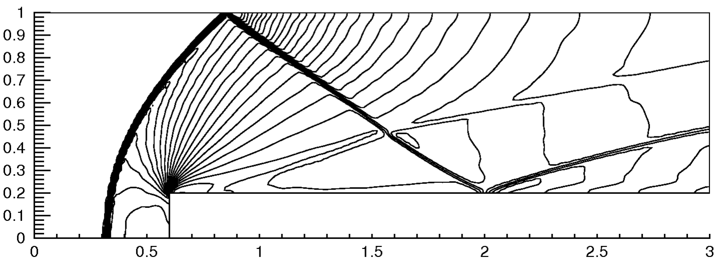


(d) CU5-TVD-MB, $t = 1$.

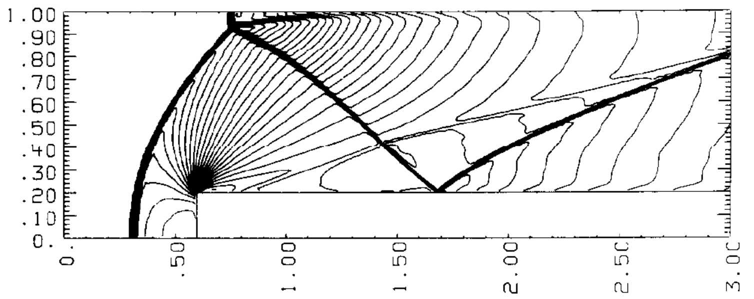
Figure 5.6: Mach 3 forward-facing step in wind tunnel. Density contours, 30 equally spaced levels between: 0.2365 and 5.647, (a)–(b); 0.2628 and 7.564, (c)–(d). Mesh resolution: 80 × 240 cells. PPMLR results from Woodward and Colella (78).



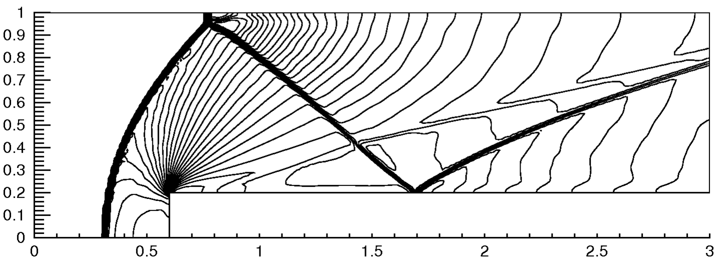
(a) PPMLR, $t = 1.5$.



(b) CU5-TVD-MB, $t = 1.5$.

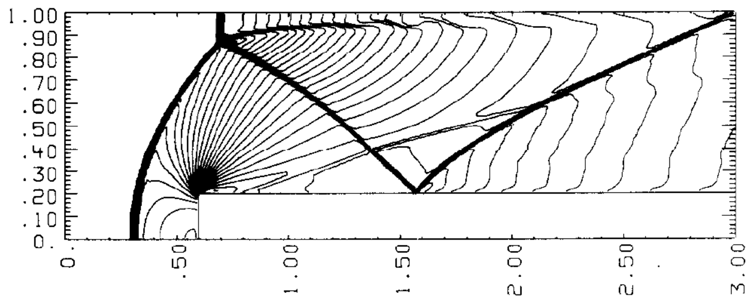


(c) PPMLR, $t = 2$.

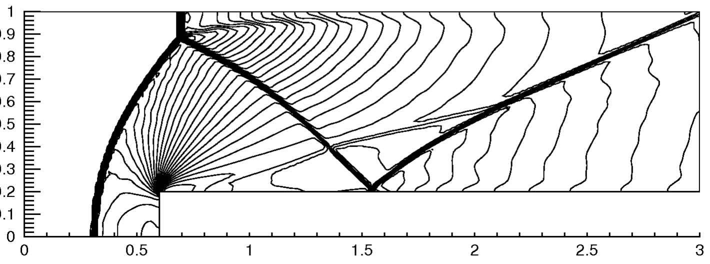


(d) CU5-TVD-MB, $t = 2$.

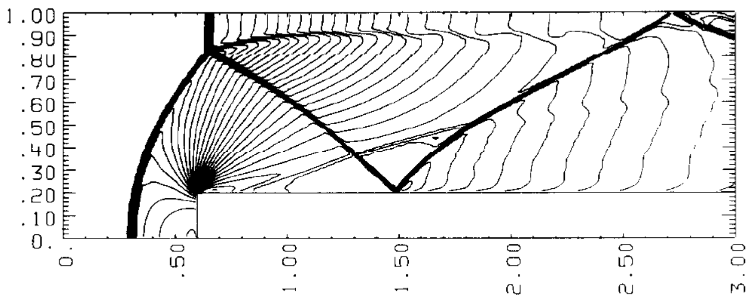
Figure 5.7: Mach 3 forward-facing step in wind tunnel. Density contours, 30 equally spaced levels between: 0.2805 and 7.717, (a)–(b); 0.2669 and 6.650, (c)–(d). Mesh resolution: 80×240 cells. PPMLR results from Woodward and Colella (78).



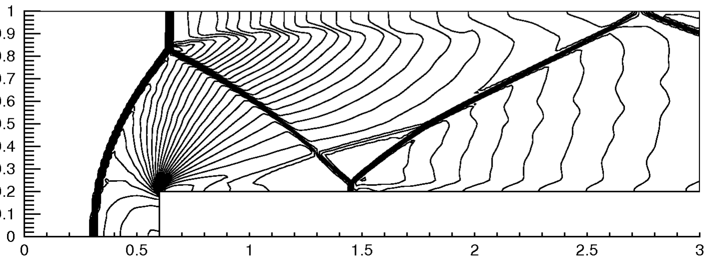
(a) PPMLR, $t = 2.5$.



(b) CU5-TVD-MB, $t = 2.5$.

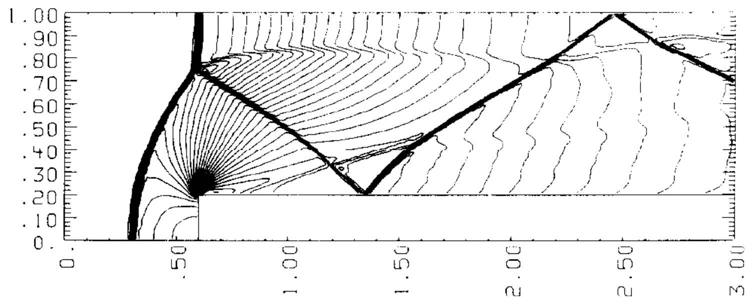


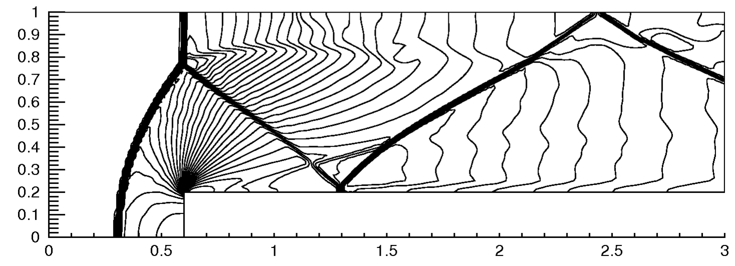
(c) PPMLR, $t = 3$.

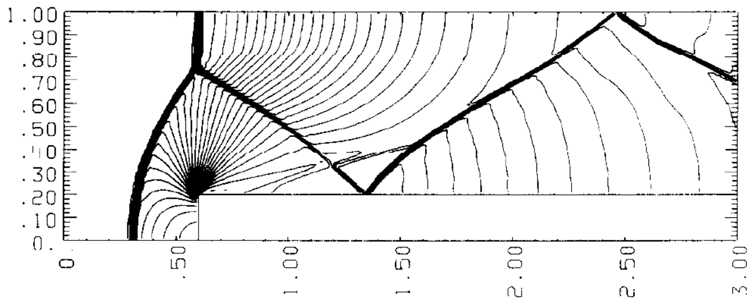


(d) CU5-TVD-MB, $t = 3$.

Figure 5.8: Mach 3 forward-facing step in wind tunnel. Density contours, 30 equally spaced levels between: 0.2688 and 6.602, (a)–(b); 0.2673 and 6.383, (c)–(d). Mesh resolution: 80×240 cells. PPMLR results from Woodward and Colella (78).


(a) PPMLR, density at $t = 4$.


(b) CU5-TVD-MB, density at $t = 4$.


(c) PPMLR, pressure at $t = 4$.

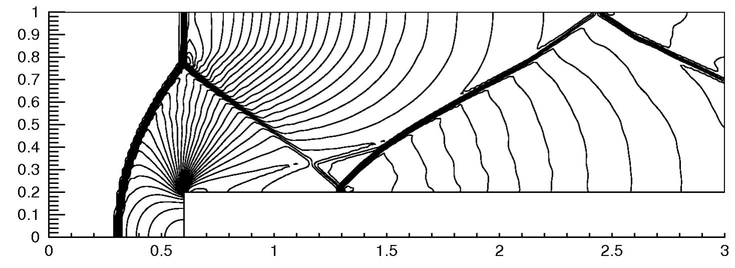
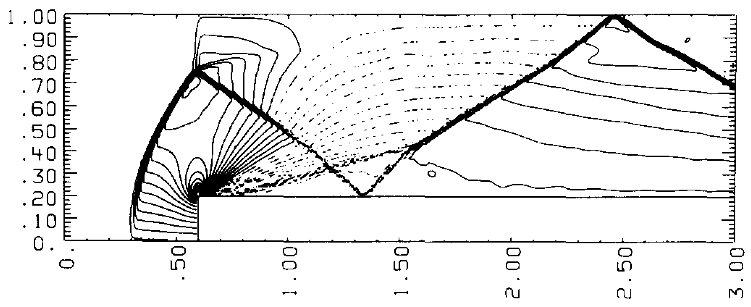
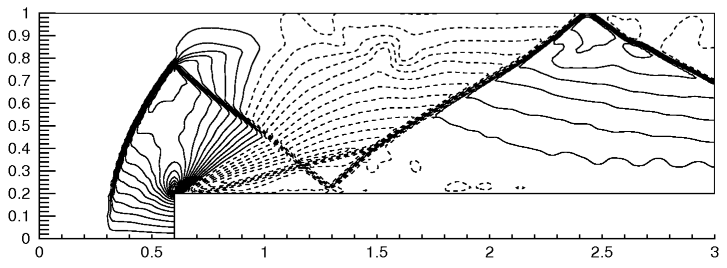

(d) CU5-TVD-MB, pressure at $t = 4$.

Figure 5.9: Mach 3 forward-facing step in wind tunnel, 30 equally spaced contours of: density between 0.2568 and 6.067, (a)–(b); pressure between 0.2752 and 11.82, (c)–(d). Mesh resolution: 80×240 cells. PPMLR results from Woodward and Colella (78).



(a) PPMLR



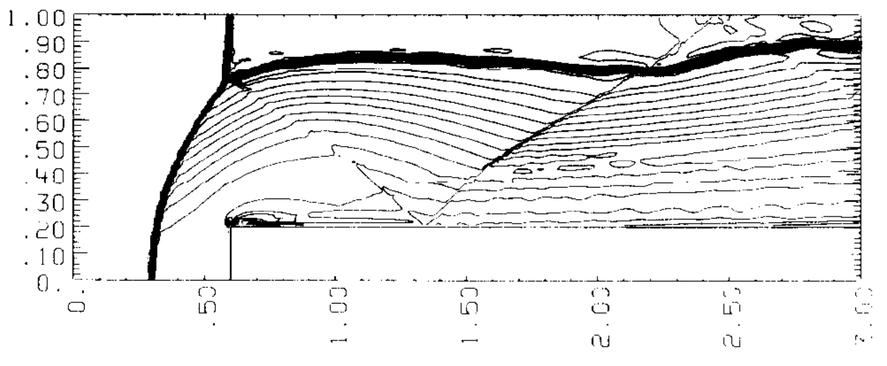
(b) CU5-TVD-MB

Figure 5.10: Mach 3 forward-facing step in wind tunnel. Vertical velocity contours at time $t = 4$: 30 equally spaced levels between -1.005 and 1.461. Dashed lines denote negative values. Mesh resolution: 80×240 cells. PPMLR results from Woodward and Colella (78).

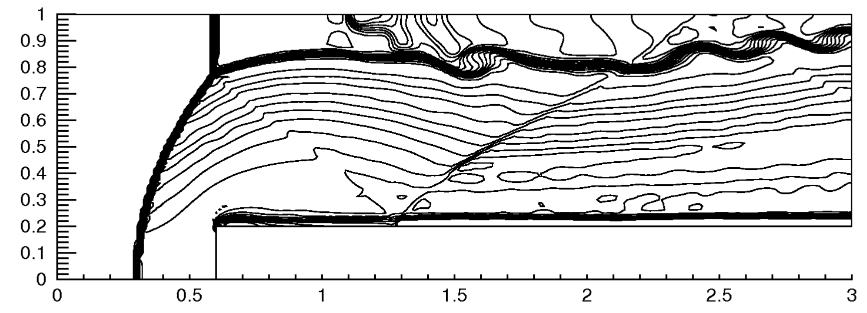
flow structures. This time evolution is shown in terms of density contours in Figs. 5.6-5.9. These figures include our results as well as those obtained by Woodward and Colella using the PPMLR method (78). The evolution can be summarised as follows. A bow shock forms ahead of the step (Figs. 5.6(a)–5.6(b)), and is reflected on the upper wall of the tunnel (Figs. 5.6(c)–5.6(d)) and curved by the interaction with the expansion fan generated at the corner of the step. The reflected shock then hits the surface of the step (Figs. 5.7(a)–5.7(b)) and is further reflected. Later, the regular reflection on the upper wall turns into a Mach-reflection (Figs. 5.7(c)–5.7(d)) and a contact discontinuity develops (Figs. 5.8(a)–5.8(b)). Finally, a second reflection appears on the upper wall (Figs. 5.8(c)–5.8(d)), while the Mach-stem moves upstream (Figs. 5.9(a)–5.9(b)).

This evolution is very difficult to simulate for several reasons:

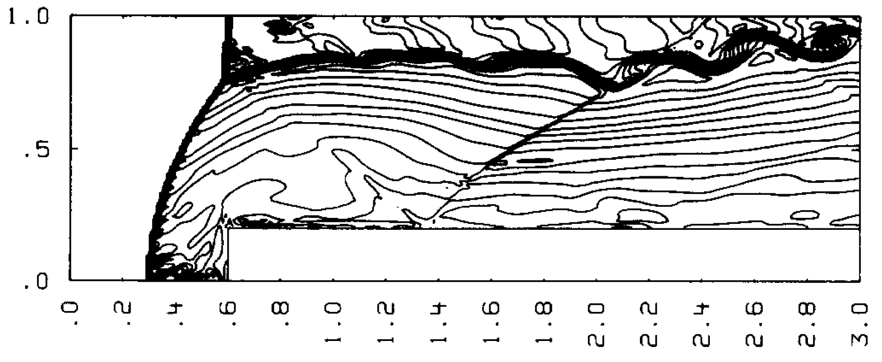
- The Mach-stem is nearly orthogonal to the wall and moves slowly upstream. Many numerical methods fail to predict slowly moving discontinuities parallel to the grid lines. Some Riemann solvers are unstable in these conditions (81).



(a) PPMLR

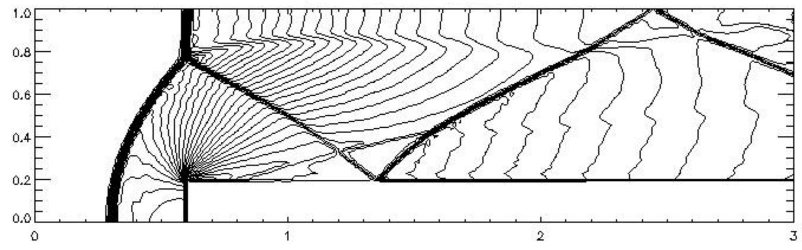


(b) CU5-TVD-MB

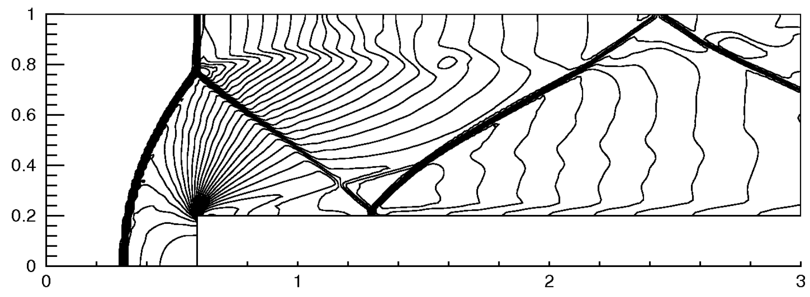


(c) MUSCL

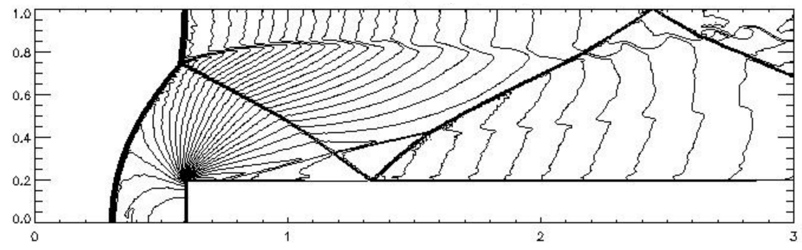
Figure 5.11: Mach 3 forward-facing step in wind tunnel. Contours of entropy ratio, $s = p/\rho^{\gamma}$, at $t = 4$: 30 equally spaced levels between 0.6325 and 1.115. Mesh resolution: 80×240 cells. PPMLR and MUSCL results from Woodward and Colella (78).



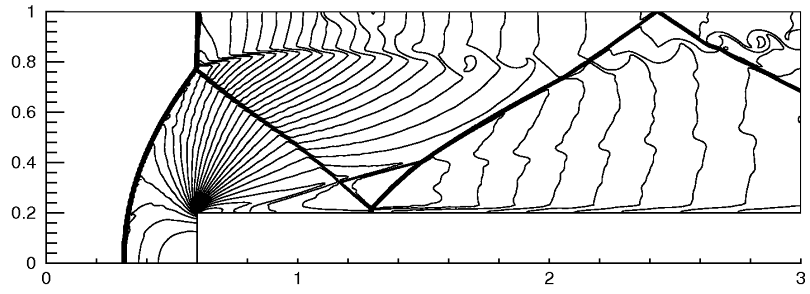
(a) Fourth-order ADER-WENO, 80 × 240 cells.



(b) CU5-TVD-MB, 80 × 240 cells.

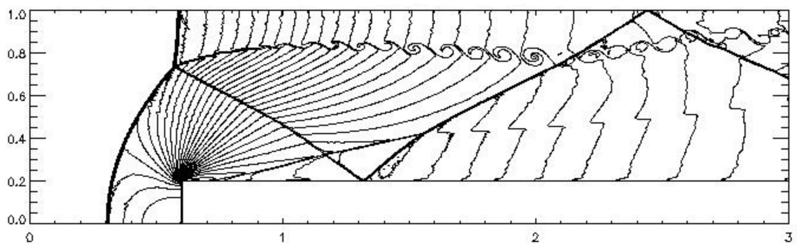


(c) Fourth-order ADER-WENO, 160 × 480 cells.

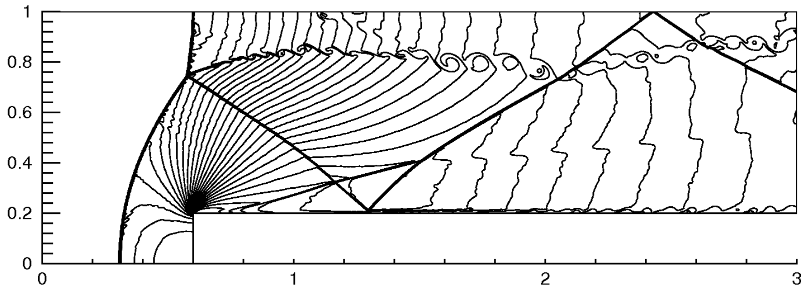


(d) CU5-TVD-MB, 160 × 480 cells.

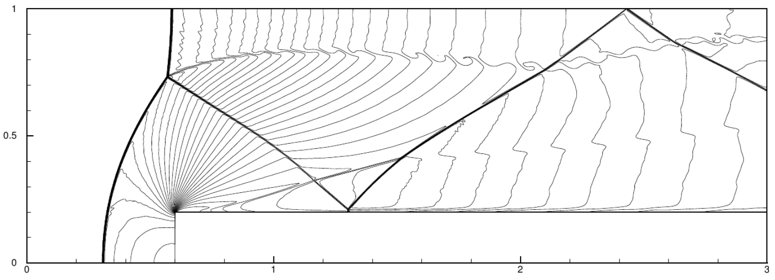
Figure 5.12: Mach 3 forward-facing step in wind tunnel. Density contours at time $t = 4$: 30 equally spaced levels between 0.090338 and 6.2365. ADER-WENO results from Balsara *et al.* (79).



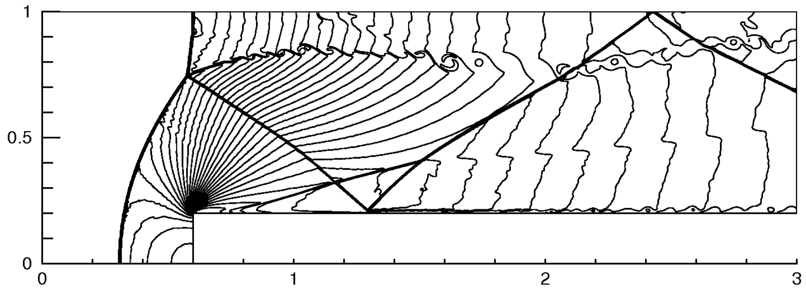
(a) Fourth-order ADER-WENO



(b) CU5-TVD-MB



(c) Fifth-order explicit-WENO



(d) CU5-TVD-MB

Figure 5.13: Mach 3 forward-facing step in wind tunnel. Density contours at time $t = 4$, 30 equally spaced levels between: 0.090338 and 6.2365, (a)–(b); 0.32 and 6.15, (c)–(d). Mesh resolution: 320×960 cells. ADER-WENO results from Balsara *et al.* (79), explicit-WENO results from Li and Qiu (80).

- The formation of the Mach-stem is a threshold phenomenon. If the bow shock is smeared over too many cells, the Mach-stem forms too late, and, even if the method is capable of correctly predicting its motion, it does not reach its correct position.

- Downstream from the reflection, a contact discontinuity develops. Sharp detection of such a discontinuity is extremely difficult.

- The contact discontinuity is a vortex sheet and suffers from the Kelvin-Helmholtz instability. This physical instability results in the roll-up of the vortex sheet and may be triggered by post-shock oscillations. The appearance of this instability depends on how the numerical method controls these oscillations, and its development depends on the dissipation characteristics of the numerical method.

The corner of the step is the centre of an expansion fan, and so a singular point for the flow-field. Large numerical errors are introduced at this point, affecting the simulated flow field to an extent that depends on the numerical method. The most common error observed is the formation of a numerical boundary layer on the step surface. The interaction between a shock wave and this numerical boundary layer may dramatically alter the simulated evolution. Woodward and Colella tailored their code to achieve the best results on this simulation, and, in order to minimise the error inevitably introduced at the corner, explicitly imposed the conservation of entropy and total enthalpy in the cells neighbouring the corner. The cells involved were the first four cells from the corner in the first row above the step surface, and the first two cells from the corner in the second row above the step surface. In these cells the density was reset to enforce entropy conservation:

$$\rho_{right} = \rho_{left} \left(\frac{p_{right}}{p_{left}} \right)^{\frac{1}{\gamma}},$$

while the velocity magnitude was reset to enforce total enthalpy conservation:

$$V^2_{right} = V^2_{left} + \frac{2\gamma}{\gamma - 1} \left(\frac{p_{left}}{\rho_{left}} - \frac{p_{left}}{\rho_{left}} \right).$$

Even adopting this solution, an over-expansion is predicted at the corner, as well as a weak oblique shock downstream where the over-expanded flow hits the step surface.

The PPMLR solver is based on a Lagrangian method employing two successive piece-wise parabolic reconstructions and a Riemann solver. Also, in the remapping step grid jittering is performed in order to improve the computation of slowly moving discontinuities. In addition, as stated before in this section, Woodward and Colella implemented a “patch” for entropy and total enthalpy at the corner. For these reasons, their results represent an excellent reference solution. In Figs. 5.6-5.11 for our results we use the same contour levels as Woodward and Colella, and so the figures are a direct comparison between the CU5-TVD-MB and PPMLR method.

Our results compare very well to Woodward and Colella’s. The discontinuities are captured as sharply, with no appreciable post-shock oscillations. Speed and location of discontinuities agree at any time. The Mach-stem forms at time $t = 2$ in both simulations, and its final location at time $t = 4$ is the same for both methods. The CU5-TVD-MB method is somewhat less effective in capturing the contact discontinuity, as its description is not quite as sharp. This could be attributed to the boundary closures: as the triple-wave point moves from the top wall towards the step, the resolution of the contact discontinuity improves.

The worse resolution of the contact surface could also be linked to its instability. The PPMLR method seems to predict the onset of the Kelvin-Helmholtz instability between time 2.5 and 3 (Figs. 5.8(a) and 5.8(c)), but there is no trace of it at time 4 (Fig. 5.9(a)). In our simulation the instability does manifest: it is hardly visible from the density contours, but it is clear in the vertical-velocity contours, shown in Fig. 5.10. Woodward and Colella also simulated the phenomenon using a second-order MUSCL scheme that did detect the instability. They state that the post-shock oscillations trigger the instability in the MUSCL simulation, and these oscillations are better controlled by the PPMLR method. In our case though, there are no appreciable post-shock oscillations and more likely the higher accuracy and better resolution properties are responsible for the onset of the instability. This statement is corroborated by comparing the contours of the entropy ratio, $s = p/\rho^{\gamma}$, for the three methods in Fig. 5.11. This comparison also highlights the presence of a numerical boundary layer on the step surface in our simulation, where no special fix has been adopted at the corner. This boundary layer is responsible for the Mach-reflection of the shock hitting the step surface, visible in both density and pressure contours, but has no major effect far from the step surface.

We have used this test to perform a resolution study. Three grids are considered, where $\Delta x = \Delta y = 1/80, 1/160, 1/320$. The results are shown in Figs. 5.12 and 5.13.

Increasing the grid resolution, the flow structure does not change but:

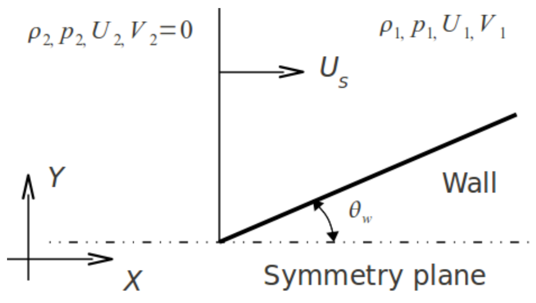
- The discontinuities thin. The thickness of discontinuities, expressed in number of grid points, is a property of the method, and therefore the finer the grid the sharper the discontinuities.
- The numerical boundary layer on the step surface thins. As with the discontinuities, its thickness expressed in number of grid-points is a property of the method.
- The roll-up of the contact surface is better captured.

In Figs. 5.12, 5.13(a) and 5.13(b), our results are compared to those obtained by Balsara *et al.* (79) using a fourth-order ADER-WENO method. The same contour levels are used and therefore the two methods can be directly compared. On the coarse mesh (Figs. 5.12(a) and 5.12(b)), CU5-TVD-MB is more effective in controlling the high-frequency spurious oscillations. The two methods show similar roll-up of the slip-line and similar grid-convergence property. The ADER-WENO method does not predict a numerical boundary layer on the step surface.

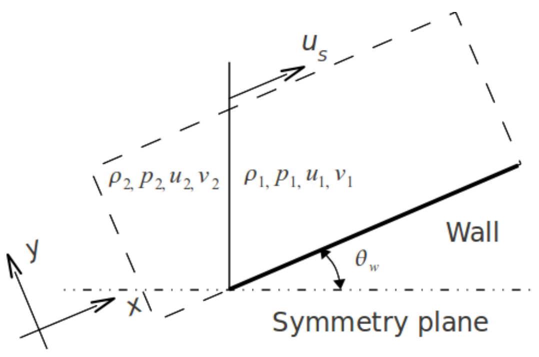
Finally, in Figs. 5.13(c) and 5.13(d) our results on the finest grid are compared to those obtained by Li and Qiu (80) using a fifth-order explicit-WENO method. The same contour levels are used for the two contour plots. The results are virtually identical away from the slip-line, indicating that for smooth flows the explicit formula Eq. 2.19 (employed by Li and Qiu), and the compact formula Eq. 2.9, are essentially equivalent. Indeed, the roll-up of the slip-line is better captured by the compact-TVD method. Both methods predict a numerical boundary layer that determines a Mach-reflection at the step surface. However, downstream from this second Mach-reflection, the density is smooth in Li and Qiu's results, while it is oscillating in ours: this is likely due to the instability of the slip-line departing from this second triple-wave point.

5.1.3 Double Mach-reflection

The last inviscid test we present is the reflection of a strong shock over a wedge. A schematic of the physical problem is shown in Fig. 5.14(a). The shock wave travels at speed U_s and at time $t = 0$ hits the wedge. This phenomenon has been extensively studied in the past both analytically and experimentally, see for example the work



(a) Schematic of the physical problem.



(b) Schematic of the computational problem .

Figure 5.14: Physical and computational problems for the simulation of the shock reflection over a wedge.

by Ben-Dor (83). The reflection has a different structure depending upon the Mach number of the incident shock, M_s, and the wedge angle, θ_w. These configurations are shown in Fig. 5.15 and are:

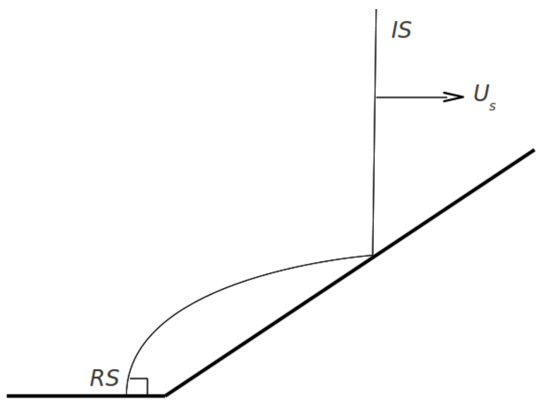
(a) Regular reflection. The incident shock (*IS*) is reflected at the wedge and the reflected shock (*RS*) is curved all the way to the symmetry plane, allowing the flow to be parallel to the wedge at the wedge surface behind the incident shock.

(b) Mach-reflection. A Mach-stem (*MS*) forms, locally normal to the wedge surface, intersecting the incident and reflected shocks into a triple-wave point. From this point also a curved slip-line (*SL*) departs.

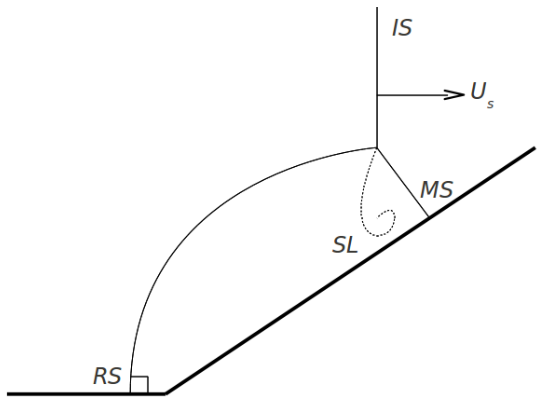
(c) Complex Mach-reflection. The reflected shock shows a kink absent in the simple Mach-reflection.

(d) Double Mach-reflection. The slip-line generated at the triple-wave point is reflected on the wedge surface as a shock. This reflected shock (RS_2) intersects the first reflected shock (*RS*), and a second triple-wave point forms instead of the kink. From this point a second slip-line (SL_2) departs.

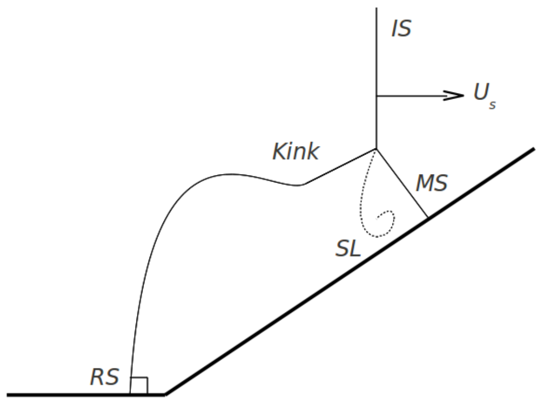
In Fig. 5.15 the reflected shock is detached from the wedge and, at the symmetry plane, is normal to the flow direction. In fact, this feature appears if a deviation equal to the wedge angle is not possible through an oblique shock for the Mach number of the flow



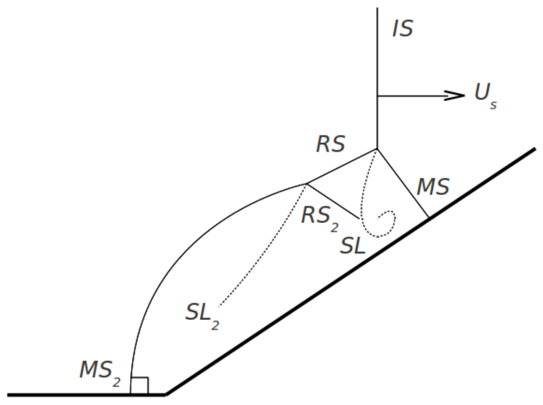
(a) Regular reflection



(b) Mach-reflection.



(c) Complex Mach-reflection

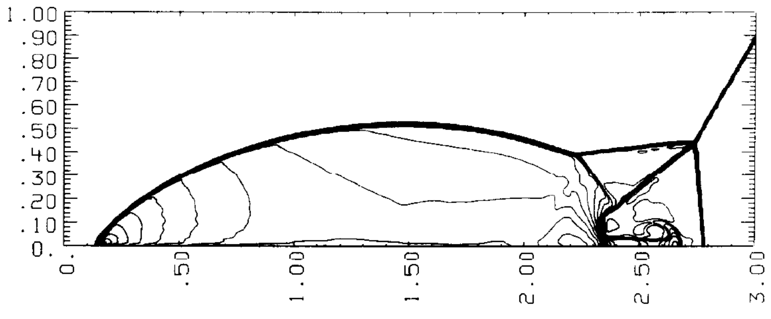


(d) Double Mach-reflection.

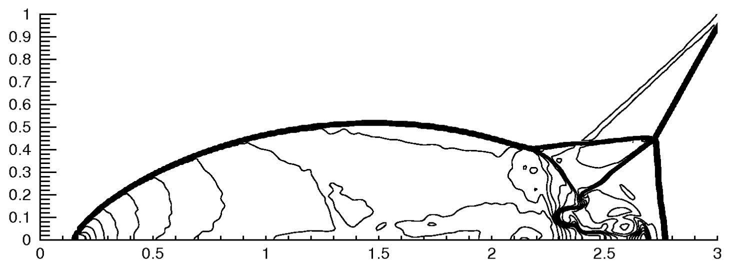
Figure 5.15: Different configurations of shock reflection over a wedge. *IS* is the incident shock trevelling at speed U_s, *RS* and RS_2 are reflected shocks, *MS* and MS_2 are Mach-stems, and *SL* and SL_2 are slip-lines.

behind the incident shock. Otherwise the reflected shock is attached to the leading edge and locally not normal to the flow direction.

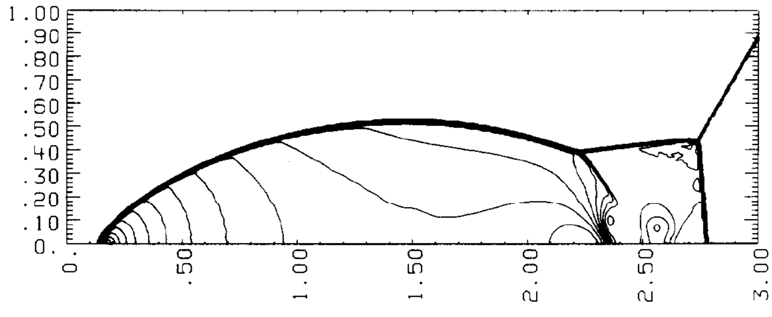
Starting from a regular reflection, the configuration evolves into a Mach-reflection, a complex Mach-reflection and a double Mach-reflection, if either the Mach number of the incident shock increases, or the wedge angle decreases. For all reflections the flow is self-similar: as the shock moves downstream, the flow structure does not change but simply "stretches".



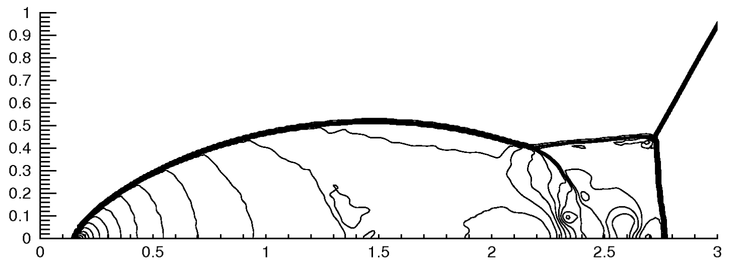
(a) PPMDE, density.



(b) CU5-TVD, density.

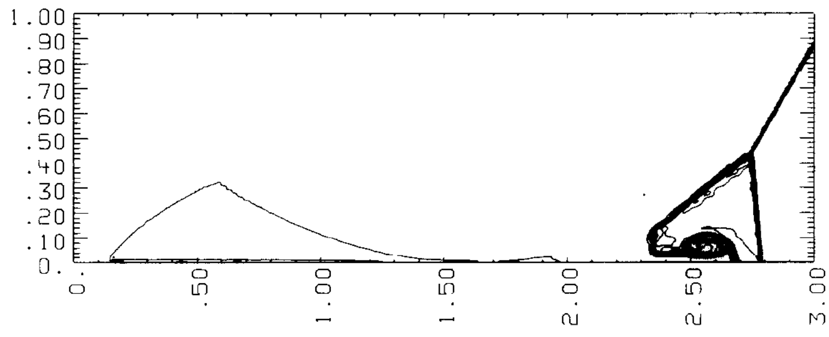


(c) PPMDE, pressure.

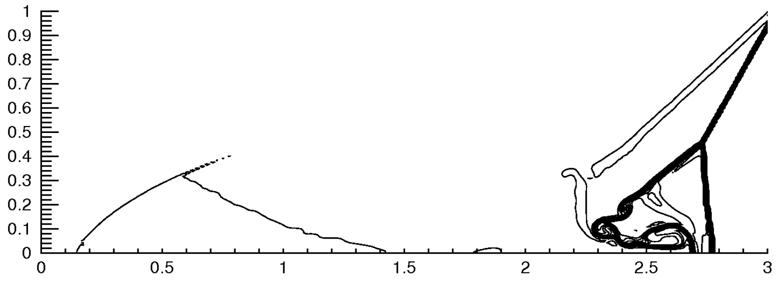


(d) CU5-TVD, pressure.

Figure 5.16: Double Mach-reflection, 30 equally spaced contours of: density between 1.731 and 20.92; pressure between 10.30 and 549.4. Mesh resolution: 120 × 480 cells. PPMDE results from Woodward and Colella (78).



(a) PPMDE



(b) CU5-TVD

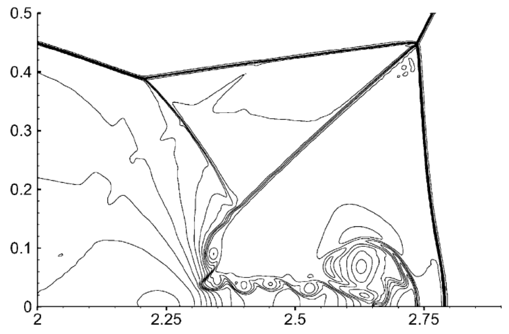
Figure 5.17: Double-Mach reflection. Contours of entropic ratio, $s = p/\rho^\gamma$: 30 equally spaced levels between and 0.8190 and 12.11. Mesh resolution: 120×480 cells. PPMDE results from Woodward and Colella (78).

Following Woodward and Colella (78), we simulate the double Mach-reflection, obtained by setting:

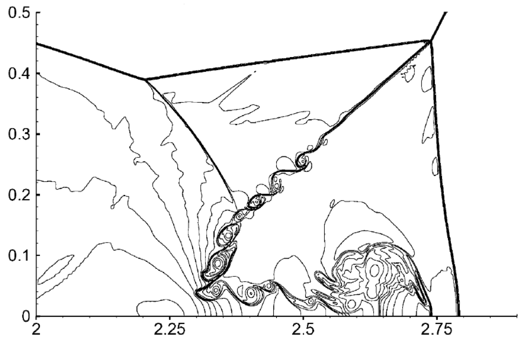
$$M_s = 10, \qquad \theta_w = \frac{\pi}{6}.$$

In order to avoid meshing a domain where the boundaries are not orthogonal to each other, the computational domain is set as a rectangle whose long edges are parallel to the wedge, as shown in Fig. 5.14(b). The new reference frame (x, y) is rotated by $\pi/6$ with respect to the original reference frame (X, Y). We denote by lowercase letters, namely (u, v), the components of the velocity vector in the (x, y) reference frame, and by capital letters, namely (U, V), those in the (X, Y) reference frame.

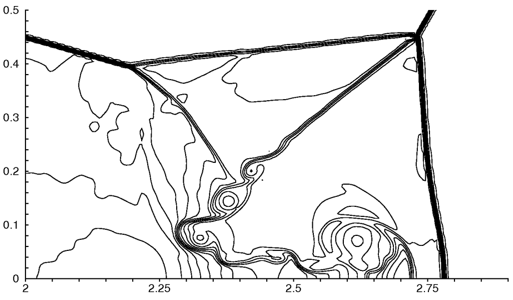
The left boundary in Fig. 5.14(b) is a supersonic inflow boundary, where the inflow conditions are set equal to the exact post-shock conditions, denoted by subscript "2", according to the standard notation for steady shock waves. According to the same convention, the pre-shock conditions are denoted by subscript "1". The pre-shock



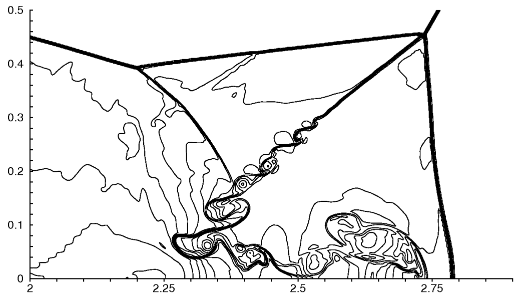
(a) Fifth-order WENO, 480 × 1920 cells



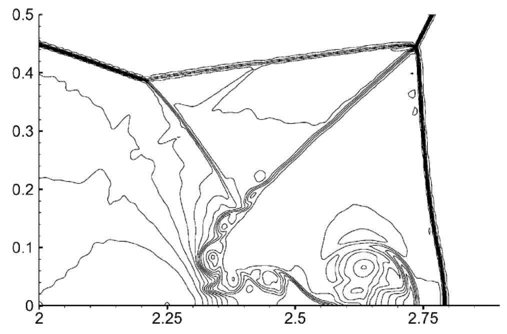
(b) Fifth-order WENO, 960 × 3840 cells



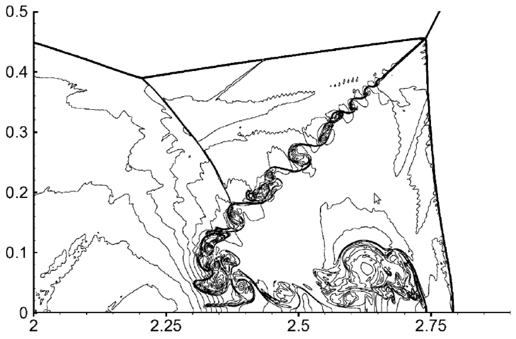
(c) CU5-TVD, 240 × 960 cells



(d) CU5-TVD, 480 × 1920 cells

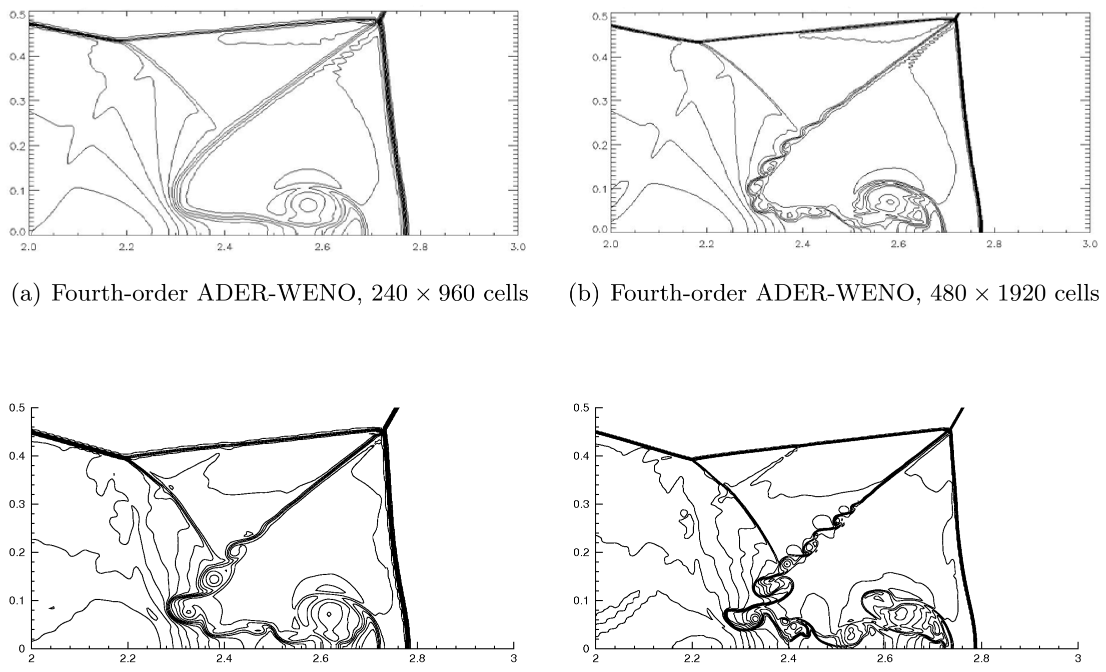


(e) Ninth-order WENO, 240 × 960 cells



(f) Ninth-order WENO, 960 × 3840 cells

Figure 5.18: Double Mach-reflection. Density contours: 30 levels equally spaced between 1.5 and 22.9705. WENO results from Shi *et al.* (82).



(a) Fourth-order ADER-WENO, 240 × 960 cells (b) Fourth-order ADER-WENO, 480 × 1920 cells

(c) CU5-TVD, 240 × 960 cells (d) CU5-TVD, 480 × 1920 cells

Figure 5.19: Double Mach-reflection. Density contours: 30 levels equally spaced between 1.3965 and 22.882. ADER-WENO results from Balsara *et al.* (79).

conditions are:

$$p_1 = 1, \qquad \rho_1 = 1.4, \qquad u_1 = v_1 = 0,$$

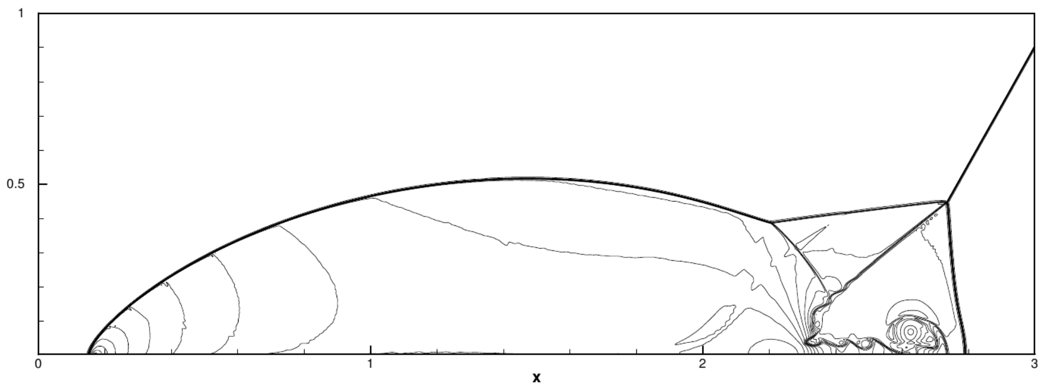
and the post-shock conditions are:

$$p_2 = 116.4, \qquad \rho_2 = 8, \qquad u_2 = 7.14471, \qquad v_2 = -4.125.$$

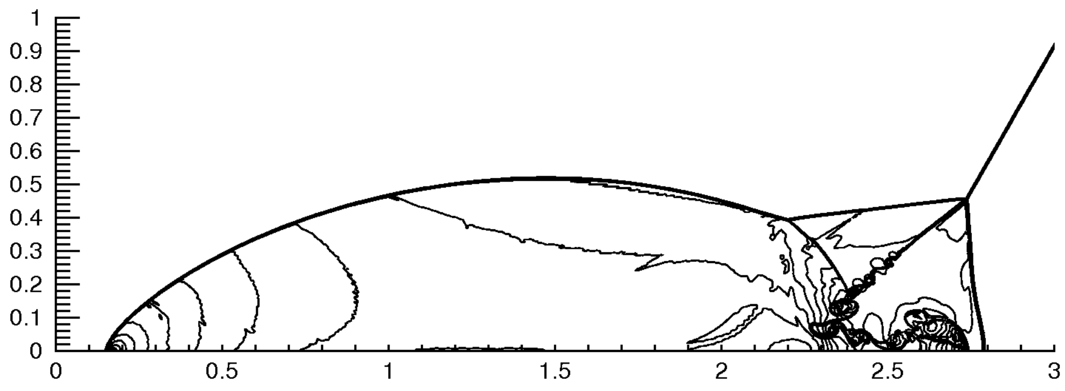
The lower boundary is a supersonic inflow up to the corner, located at $x = 1/6$, and then is a slip-wall. This forces the second Mach-stem (MS_2 in Fig. 5.15(d)) to be attached to the wedge. At the right boundary the zero-gradient condition is enforced. The exact motion of the shock wave is assigned at the top boundary. In the (x, y) reference frame the shock speed is:

$$u_s = \frac{M_s a_1}{\cos \pi/6},$$

where a_1 is the speed of sound in the pre-shock conditions. The solution at time $t = 0.2$ has been computed on the domain $(x, y) \in [0, 4] \times [0, 1]$, using a uniform grid



(a) Fifth-order explicit-WENO



(b) CU5-TVD

Figure 5.20: Double Mach reflection. Density contours: 30 levels equally spaced between 1.5 and 22.7. Mesh resolution: 480 × 1920 cells. Explicit-WENO results from Li and Qiu (80).

with $\Delta x = \Delta y = 1/120$, and is shown in Figs. 5.16 and 5.17. Our solution is compared to the reference solution computed by Woodward and Colella (78) using the PPMDE method (the Eulerian variant of the PPMLR method). The same contour levels for the two methods are used, and so the figures are directly comparable.

The slip-line that originates from the main Mach-stem suffers from the Rayleigh-Taylor (RT) instability: this is a well known phenomenon characterising interfaces between fluids with different densities. Commonly, such instability is triggered by gravity, but in this case the driving force is the stream-wise pressure gradient at the wedge surface. The instability results in a jet of denser fluid, usually referred to as a *finger*, penetrating the lower density region. The slip-line is also a vortex-sheet, and so affected by the Kelvin-Helmholtz (KH) instability. This instability results in the roll-up of the slip-line.

These two features are perhaps the most difficult to predict in this simulation, as numerical methods introduce locally high numerical dissipation to capture discontinuities, and such dissipation tends to stabilise the slip-line. On the other hand, many Riemann solvers do not introduce enough dissipation and over-predict the length of the RT finger which protrudes even beyond the Mach-stem.

Indeed, PPMDE and CU5-TVD methods predict the same flow structure. Minor differences in density and pressure contour lines (Fig. 5.16) are observed in the region underneath the second Mach-stem. The length and structure of the RT jet are very similar. The major difference is the shape of the slip-line: while PPMDE does not detect the KH instability, CU5-TVD clearly shows the roll-up of the slip-line, even at such a low resolution. The different shape of the slip-line is particularly evident in the contours of the entropic ratio, $s = p/\rho^{\gamma}$, shown in Fig. 5.17. The roll-up also causes the slip-line to protrude upstream. Finally, we note that the slip-line originating from the second triple-wave point is very weak and not visible in the density contours.

We have used this test case to perform a further resolution study. The flow over a supersonic step discussed in section 5.1.2 does not possess many small-scale features and a good second-order scheme is sufficient to simulate it. On the other hand, the small-scale structures, arising from the combination of RT and KH instabilities, require the use of higher-order schemes.

In Fig. 5.18 our results are compared to those obtained by Shi *et al.* (82) who used fifth- and ninth-order WENO schemes. Fig. 5.18(f) represents a reference solution, computed by Shi *et al.* on a very fine grid of 960×3840 cells using the ninth-order WENO scheme. Note that the main features of the slip-line are well described by CU5-TVD on the 480×1960 grid (Fig. 5.18(d)), as well as by the fifth-order WENO scheme on the 960×3840 grid (Fig. 5.18(b)). It is interesting to observe that CU5-TVD achieves on the 240×960 grid (Fig. 5.18(c)) a resolution close to the ninth-order WENO scheme on the same grid (Fig. 5.18(e)), and to the fifth-order WENO scheme on the 480×1960 grid (Fig. 5.18(a)).

In Fig. 5.19 our results are compared to those obtained by Balsara *et al.* (79) who used a fourth-order hybrid ADER-WENO method, and our results clearly show better resolution for all grid sizes. Finally, Fig. 5.20 compares CU5-TVD to the hybrid fifth-order explicit-WENO method used by Li and Qiu (80). As for the supersonic step, away from the slip-line the two solutions are virtually identical, confirming that the fifth-order explicit formula Eq. 2.9 (used by Li and Qiu), and the fifth-order compact formula Eq. 2.19, are equivalent for smooth flows. The roll-up of the slip-line, instead, is better described by the compact-TVD method than by the explicit-WENO method.

Summarising, this resolution study indicates that:

- a fifth-order WENO scheme needs twice as many nodes per direction to achieve the same resolution as our compact-TVD method;
- the compact-TVD method performs better than hybrid methods of comparable accuracy, combining a high-resolution formula and a WENO scheme;
- a lack of dissipation in the compact-TVD method may be responsible for the slip-line protruding upstream in the simulation of a double-Mach-reflection.

5.2 Simulation of viscous flows

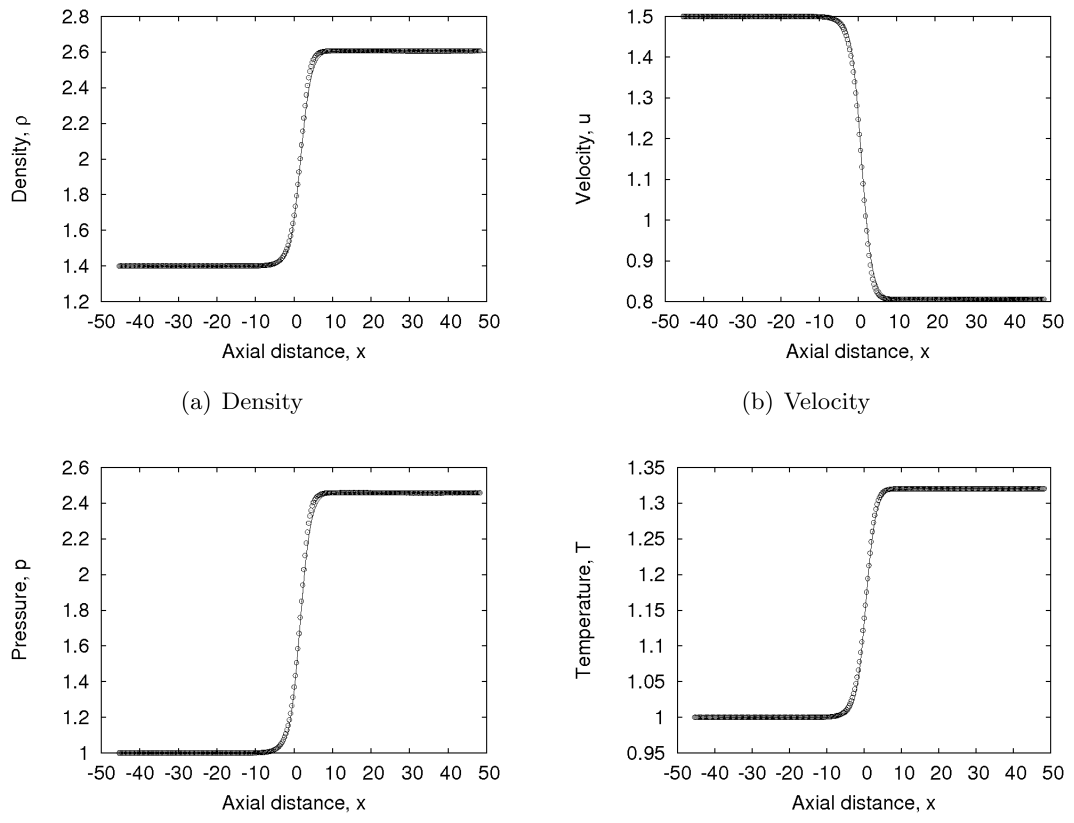
Viscous flow simulations reported in this section were run at a Courant number $CFL = 0.1$ and a Neumann number $Neu = 0.05$. The Courant number has been introduced in section 5.1. The Neumann number is used to ensure the numerical stability of the viscous discretisation. On a 2D FD grid where the nodes are indexed by (i, j) it is defined as:

$$Neu = \max_{i,j} \left\{ \nu_{i,j} \left[\left(\frac{|\nabla \xi_{i,j}|}{\Delta \xi} \right)^2 + \left(\frac{|\nabla \eta_{i,j}|}{\Delta \eta} \right)^2 \right] \right\} \Delta t,$$

where $\nu = \mu/\rho$ is the dynamic viscosity. The tests presented here required the computation of a steady state. The numerical method we use is for time-accurate simulations and hence not efficient for steady-state calculations because there is a severe time-step restriction for stability. Convergence was assumed to be achieved when no appreciable variation in the quantities of interest was observed on the numerical grid used.

5.2.1 Shock layer

The shock wave formation is an inviscid phenomenon, as it is related to the coalescence of compression waves. While shock waves are discontinuities in inviscid flows,



(a) Density

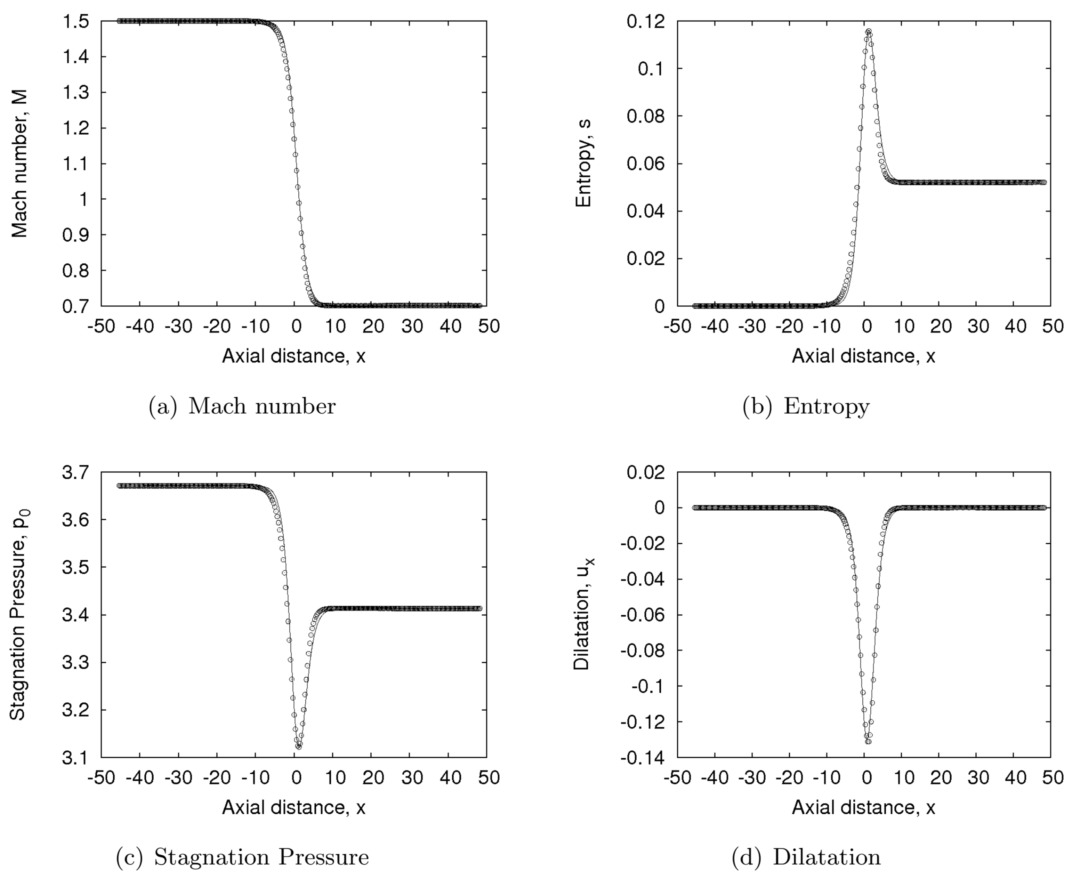
(b) Velocity

(c) Pressure

(d) Temperature

Figure 5.21: Solutions to the shock layer problem: —, analytical solution; ∘, our numerical solution.

in viscous flows they are regions where the variation of fluid dynamic variables occurs within a spatial distance much smaller than the length scale of the flow field. The region where the fluid evolves from the state upstream of the shock wave to the downstream state is called the *shock layer*. Its thickness can be as small as a few molecular mean free paths, even in a macroscopically continuum flow, and so the Navier-Stokes equations are generally not suitable for describing the shock layer. An analytical solution to the Navier-Stokes equations is available for the one-dimensional shock layer problem (84). The solution procedure is based on the self-similar function technique, and the complete derivation can be found in Lagerstrom (84). While this solution can be of limited physical significance, because of the breakdown within the shock layer of the continuum hypothesis, which the Navier-Stokes equations rely on, it can be em-



(a) Mach number

(b) Entropy

(c) Stagnation Pressure

(d) Dilatation

Figure 5.22: Solutions to the shock layer problem: —, analytical solution; ○, our numerical solution.

ployed as a benchmark for numerical methods. Denoting by subscripts "1" and "2" the states upstream and downstream of the steady shock wave, respectively, and defining the following quantities:

$$m = \rho_1 u_1 = \rho_2 u_2, \quad U_a = \frac{u_1 + u_2}{2}, \quad V = \frac{u_1 - u_2}{2}, \quad H = h_1 + \frac{u_1^2}{2} = h_2 + \frac{u_2^2}{2},$$

the shock layer is described by the following equations (84):

$$\rho(\xi)u(\xi) = m, \tag{5.1}$$

$$u(\xi) - U_a = V \tanh\left[\frac{-Vm(\gamma+1)\xi/U_a}{2\gamma + 2(\gamma-1)\left(\frac{1}{Pr_{\text{tot}}} - 1\right)}\right], \tag{5.2}$$

$$h(\xi) + \frac{u(\xi)^2}{2} = H. \tag{5.3}$$

The quantity ξ is a transformed spatial variable such that:

$$x(\xi) = \int_0^{\xi} \mu_{\text{tot}} d\tilde{\xi}. \tag{5.4}$$

The modified viscosity μ_{tot} incorporates the bulk viscosity μ_b, i.e. $\mu_{\text{tot}} = 2\mu + \mu_b$, and the modified Prandtl number is $Pr_{\text{tot}} = c_p\mu_{\text{tot}}/k$, where c_p is the constant pressure specific heat and k the thermal conductivity. The self-similar nature of the solution is clear from Eqs. 5.1–5.3: the fluid dynamic variables are functions of ξ, whose spatial scale is in turn a function of the modified viscosity μ_{tot}: $d\xi = \mu_{\text{tot}} dx$.

We have computed a steady shock layer, with shock Mach number $M_s = 1.5$, as in the work by Lele (3). The problem solved is dimensionless, and a simple linear viscosity law, $\mu = T$, is assumed, so that the integral in Eq. 5.4 can be solved analytically. We set $\mu_b = -2/3\mu$, according to the Stokes hypothesis, the Prandtl number is $Pr = 3/4$ and the isentropic index is $\gamma = 7/5$. The upstream conditions are:

$$\rho_1 = 7/5, \qquad u_1 = 1.5, \qquad p_1 = 1.$$

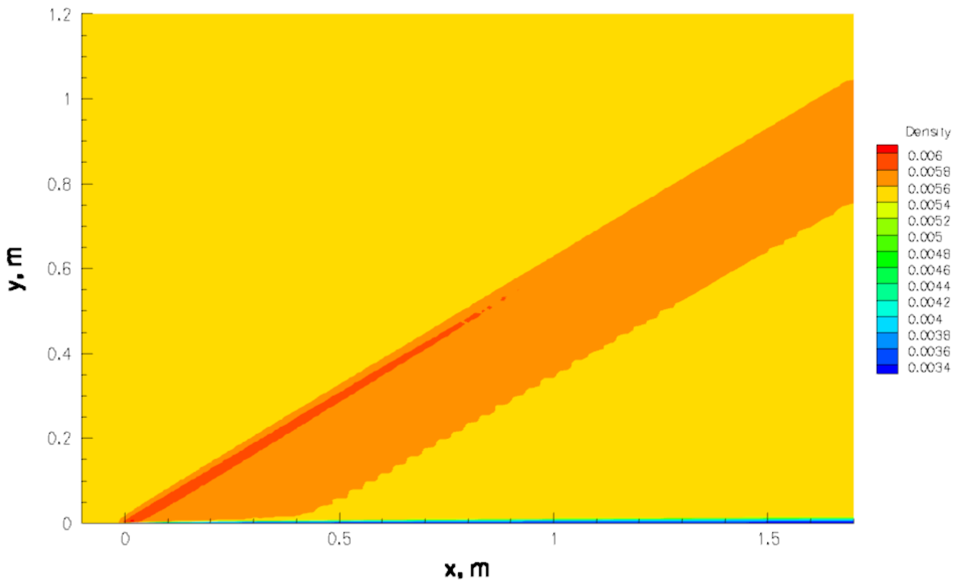
The gas constant is set to $R = 5/7$, so that $T_1 = 1$, and the downstream conditions are given by the Rankine-Hugoniot relations. In Figs. 5.21 and 5.22 the results obtained on 300 uniformly spaced cells over the domain $x \in [-45, 48]$ are compared to the exact analytical solution.

Variables such as density, velocity, pressure, temperature and Mach number are monotonic, and so easy to capture. The challenge in this test case is the correct prediction of dilatation, $\partial_x u$, entropy and stagnation pressure. These quantities experience a peak within the shock thickness, that can only be captured if such a small length-scale is well resolved. Figs. 5.21 and 5.22 show excellent agreement between the numerical and analytical solutions for all variables.

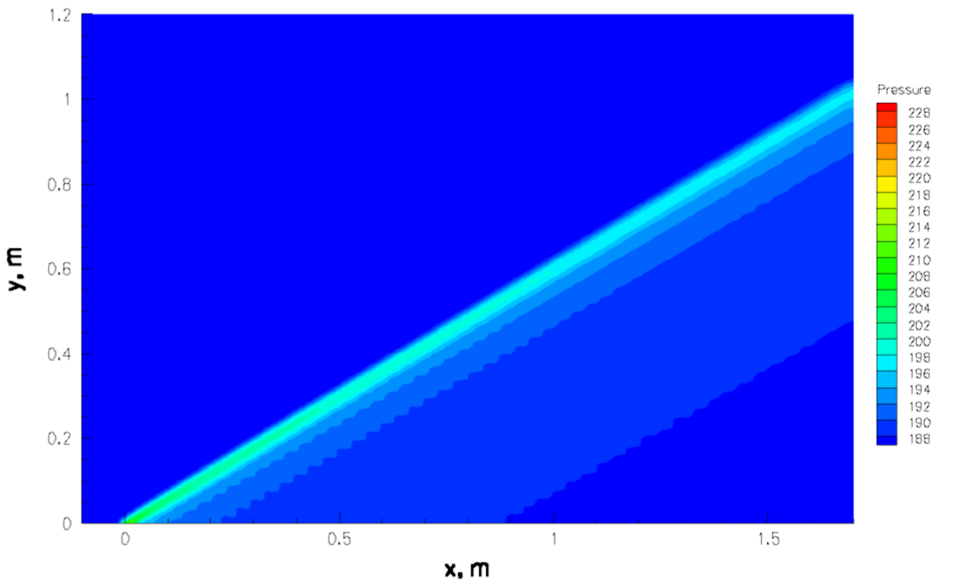
5.2.2 Self-similar boundary layer

In this section we present the simulation of a steady supersonic laminar boundary layer over a flat plate, and we validate our results against an exact solution, namely the compressible self-similar boundary layer.

Under certain assumptions, i.e. steady two-dimensional flow, zero stream-wise pressure gradient and thin boundary layer, the Navier-Stokes equations can be simplified



(a) Density, kg/m^3



(b) Pressure, Pa

Figure 5.23: Supersonic flow over a flat plate: steady state solution computed on 108×270 cells.

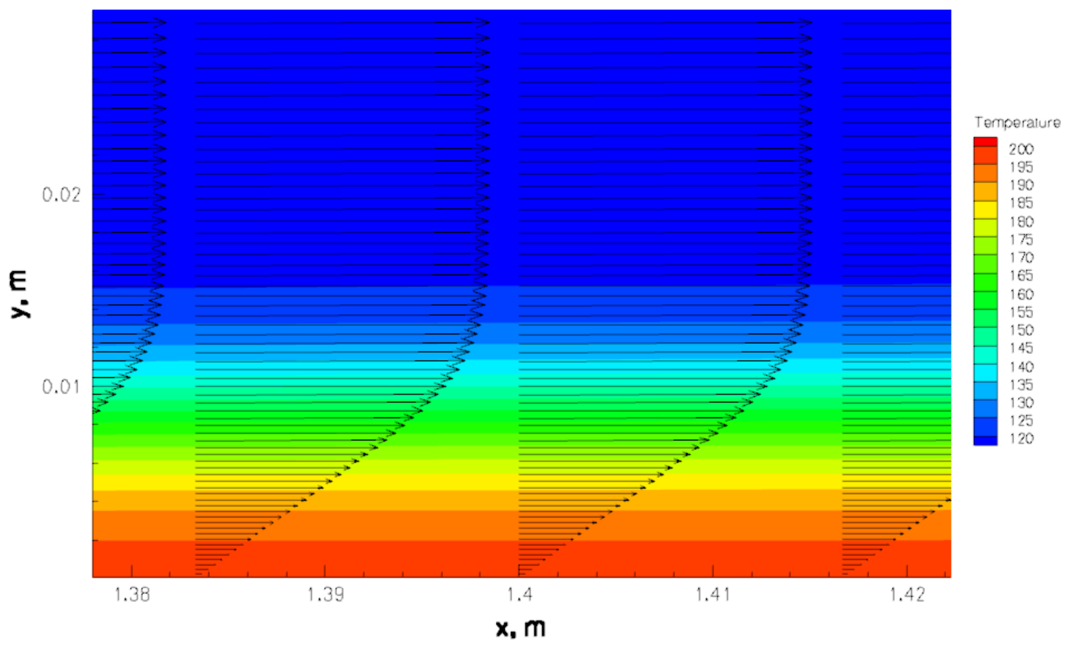


Figure 5.24: Temperature contours (Kelvin) and velocity profiles in the boundary layer for the supersonic flat plate.

into the so called *boundary layer equations*:

$$\frac{\partial \rho u}{\partial x} + \frac{\partial \rho v}{\partial y} = 0, \tag{5.5}$$

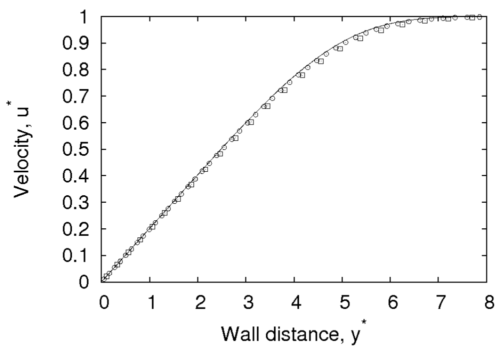
$$\rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} = \frac{\partial}{\partial y}\left(\mu \frac{\partial u}{\partial y}\right), \tag{5.6}$$

$$\frac{\partial p}{\partial y} = 0, \tag{5.7}$$

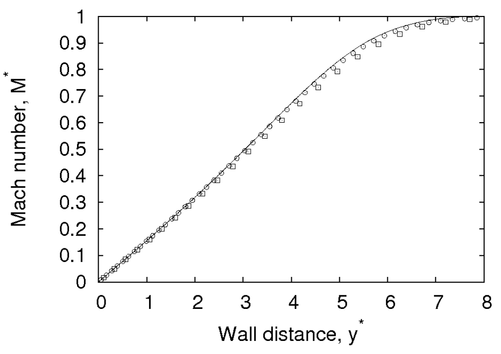
$$\rho u \frac{\partial h}{\partial x} + \rho v \frac{\partial h}{\partial y} = \mu \left(\frac{\partial u}{\partial y}\right)^2 + \frac{1}{Pr}\frac{\partial}{\partial y}\left(\mu \frac{\partial h}{\partial y}\right). \tag{5.8}$$

These equations can be expressed in terms of transformed coordinates (ξ, η), such that in the transformed plane the solution does not depend on the ξ coordinate. The appropriate transformation from the physical plane (x, y) to the transformed plane (ξ, η) is based on the work started in 1940 by Illingworth, Stewartson, Howarth and Dorodnitsyn, and eventually systematised by Levy and Lees (1). The transformation, often referred to as the *Lees-Dorodnitsyn transformation*, is:

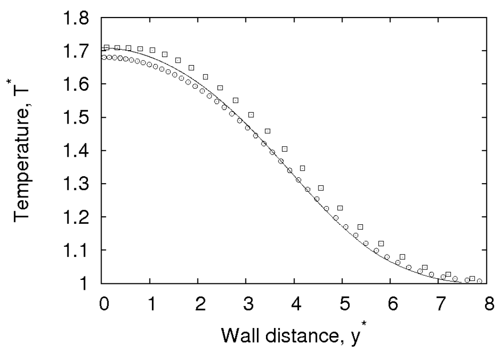
$$\xi = \int_0^x \rho_e u_e \mu_e dx, \quad \eta = \frac{u_e}{\sqrt{2\xi}} \int_0^y \rho dy, \tag{5.9}$$



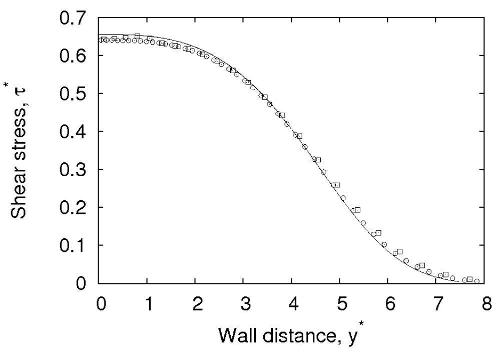
(a) Non-dimensional velocity



(b) Mach number



(c) Non-dimensional temperature



(d) Friction coefficient

Figure 5.25: Boundary layer profiles: —, self-similar solution; □, numerical solution on 54×135 cells; ○, numerical solution on 108×270 cells.

where ρ_e, u_e and μ_e are, respectively, density, x-velocity and viscosity at the edge of the boundary layer. We note that, as these variables depend just on x, ξ does not depend on y, i.e. $\xi = \xi(x)$. It can be proved that, for certain types of wall-bounded flows, such as the flow past a flat plate, the ratios of velocity and specific enthalpy to their respective values at the edge of the boundary layer do not depend on ξ. Therefore, two functions $f(\eta)$ and $g(\eta)$ can be defined so that:

$$u/u_e = f'(\eta), \qquad h/h_e = g(\eta). \tag{5.10}$$

In this type of boundary layer the pressure is uniform, i.e. $\partial p/\partial\xi = \partial p/\partial\eta = 0$. These mathematical properties of the boundary layer equations describe a precise physical phenomenon: moving along the body surface, the profiles of the fluid dynamic variables along the direction normal to the surface do not change if they are scaled according to the local flow conditions.

Applying the transformation 5.9 to Eqs. 5.5–5.8, we obtain a system of two ordinary differential equations (ODE) in the variables f and g:

$$\left\{\begin{array}{rcl} (Cf'')' + ff' & = & 0, \\ \left(\frac{C}{Pr}g'\right) + fg' + C\frac{u_e^2}{h_e}(f'')^2 & = & 0, \end{array}\right. \tag{5.11}$$

where $C = \rho\mu/\rho_e\mu_e$. The full mathematical derivation is omitted here, as it can be found in most fluid mechanics books, for example the one by Anderson (1). The ODE system 5.11 is third order in f and second order in g, so three boundary conditions for f and two for g must be specified to close the problem. These boundary conditions are:

$$\left\{\begin{array}{lllll} f(0) = f'(0) = 0 & \text{(no-slip wall)}, & & & \\ f(\infty) = g(\infty) = 1 & \text{(boundary layer edge)}, & & & \\ g(0) = h_w/h_e & \text{(isothermal-wall) or} & & g'(0) = 0 & \text{(adiabatic-wall)}. \end{array}\right. \tag{5.12}$$

Self-similar solutions of the compressible boundary layer equations have been used in the past to study the effect of increasing Mach numbers and wall cooling on friction and heat transfer (85). We compute compressible self-similar boundary layer profiles and use them to validate our CFD code. The problem given by the ODE system 5.11 and the boundary conditions 5.12 is not an initial value problem (IVP), because not all boundary conditions are specified on the same side of the domain. Therefore, it cannot be solved with standard techniques. We follow Van Driest (85) and solve the problem via a shooting method (86). The self-similar boundary layer profiles are expressed in terms of the following non-dimensional variables:

$$u^* = u/u_\infty, \qquad T^* = T/T_\infty, \qquad M^* = u/\sqrt{\gamma p/\rho} \qquad \tau^* = \frac{\tau_{x,y}\sqrt{Re_{\infty,x}}}{1/2\rho_\infty u_\infty^2},$$

where $Re_{\infty,x}$ is the free-stream Reynolds number based on the distance from the leading edge.

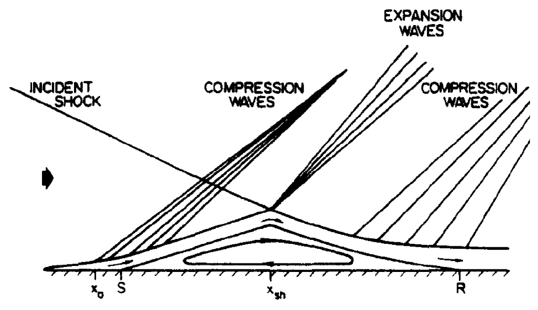
The free-stream conditions for the CFD simulation are taken from an experiment studying the shock-boundary layer interaction, presented in section 5.3. The domain is $(x, y) \in [-0.1, 1.7] \times [0, 1.2]m^2$. The free-stream Mach number, Reynolds number per metre and temperature are, respectively:

$$M_\infty = 2, \qquad Re_{\infty,1} = 2.96 \times 10^5 m^{-1}, \qquad T_\infty = 117K.$$

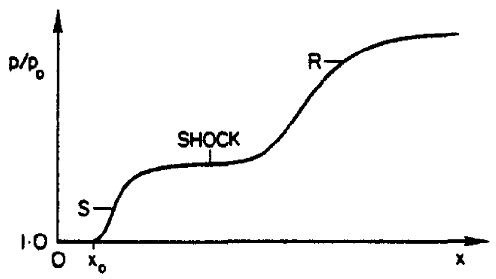
Top and left boundaries are modelled as supersonic inflow boundaries. The bottom boundary is a slip-wall up to $x = 0$, and a no-slip adiabatic wall for $x > 0$. At the left boundary the zero-gradient condition is enforced. The fluid is air, with isentropic index $\gamma = 7/5$, gas constant $R_{air} = 287.06 m^2/s^2 K$ and Prandtl number $Pr = 0.725$. Density and pressure contours of the steady-state solution are shown in Fig. 5.23.

Two grids of different resolution have been considered. The coarse grid consists of 54×135 cells; the grid is equally spaced in the x-direction and refined near the wall in the y-direction, with minimum cell size $\Delta y_{\min} = 5 \times 10^{-4} m$. In the fine grid the spacing is halved in both directions.

Data along y at a certain x station must be extracted and post-processed in order to compare the CFD results to the self-similar profiles. One of the underlying hypotheses of the boundary layer equations is the negligibility of the stream-wise pressure gradient. This hypothesis is not applicable in the region near the leading edge, where the attached shock interacts with the boundary layer. A viscous interaction parameter $\bar{\chi} = M_\infty^3 \sqrt{C/Re_{\infty,x}}$, where $C = \rho_w \mu_w / \rho_e \mu_e$, can be used to assess the nature of the interaction (1). High values of $\bar{\chi}$, typically $\bar{\chi} > 3$, indicate a strong interaction between the external field and the boundary layer. We extract profiles at the station $x = 1.4m$, where the viscous interaction parameter is $\bar{\chi} = 0.01224$, well below the threshold value for the weak viscous interaction. Temperature contours and velocity profile at this station are shown in Fig. 5.24. The data extracted is used to calculate Mach number, M^*, non-dimensional velocity, u^*, non-dimensional temperature, T^*, and friction coefficient, τ^*. These quantities are compared to the self-similar profiles in Fig. 5.25, and good agreement is observed on the coarse grid. The solution is essentially converged on this grid, and grid refinement only slightly improves the temperature profile. The wall temperature computed on the finest grid is slightly lower than the analytical one. This can be explained as follows: in the CFD simulation the Mach number of the external flow is slighly less than the free-stream Mach number, because of the shock wave at the leading edge, while the analytical solution is computed for a Mach number of the external flow equal to the free-stream Mach number.



(a) Schematic of the interaction.



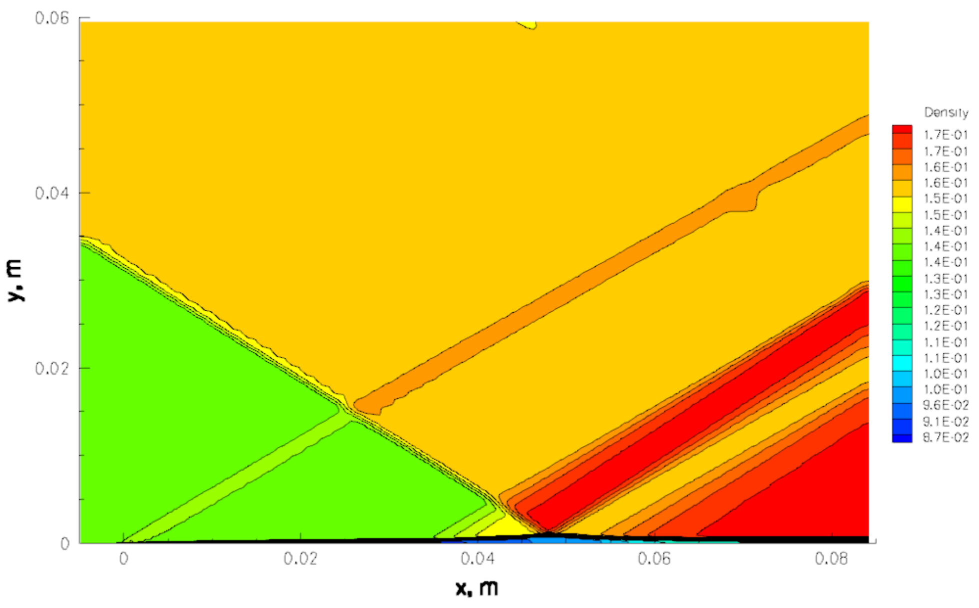
(b) Typical wall-pressure profile.

Figure 5.26: Interaction of an oblique shock wave with a laminar boundary layer over a flat plate. Reproduced from Reyhner and Flügge-Lotz (87).

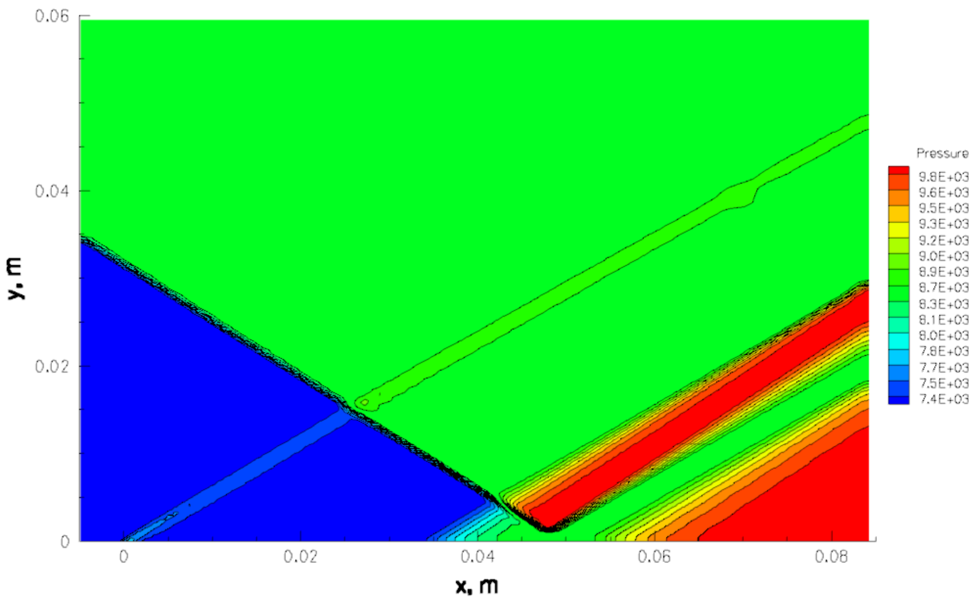
5.3 Laminar shock-boundary layer interaction

The shock-boundary layer interaction (SBLI) is one of the most extensively studied phenomena in compressible fluid dynamics. This type of interaction occurs in a very wide range of applications, such as supersonic and hypersonic propulsion and flight. It often determines boundary layer separation, deteriorating the aerodynamic efficiency that is important for supersonic applications, and increasing the local heat transfer that is critical in hypersonic applications. In this section we consider the two-dimensional interaction of an oblique shock with a laminar boundary layer over a flat plate. A schematic of this interaction is shown in Fig. 5.26, reproduced from Reyhner and Flügge-Lotz (87).

The shock wave impinges on the boundary layer and applies a strong adverse pressure gradient on it. If the shock is sufficiently strong, a separation bubble forms. The bubble starts upstream from the shock impingement location, as the high pressure behind the shock feeds upstream through the subsonic portion of the boundary layer. At the location where the boundary layer separates, identified by the letter S in Fig. 5.26, the external supersonic flow deviates from the main flow direction, and so compression waves form, which, at a certain distance from the interaction, coalesce into the so-called separation shock. These compression waves cause a rise in the wall pressure. The separated boundary layer subsequently turns back toward the plate, and an expansion fan forms in the external flow. The corresponding pressure variation does not reach the wall, shielded by the separation bubble, and therefore a plateau, characteristic of the SBLI, appears in the wall pressure profile. Finally, the boundary layer reattaches to the

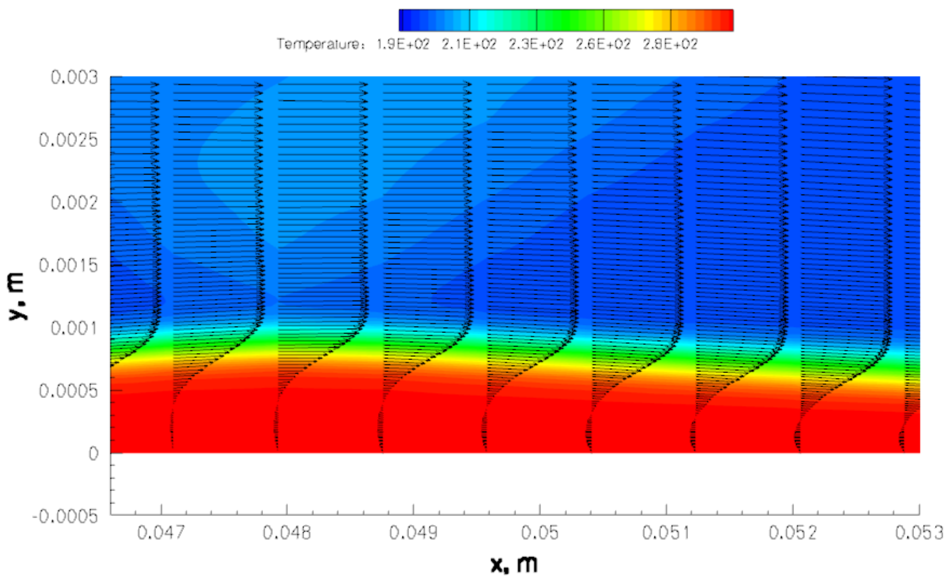


(a) Density, kg/m^3

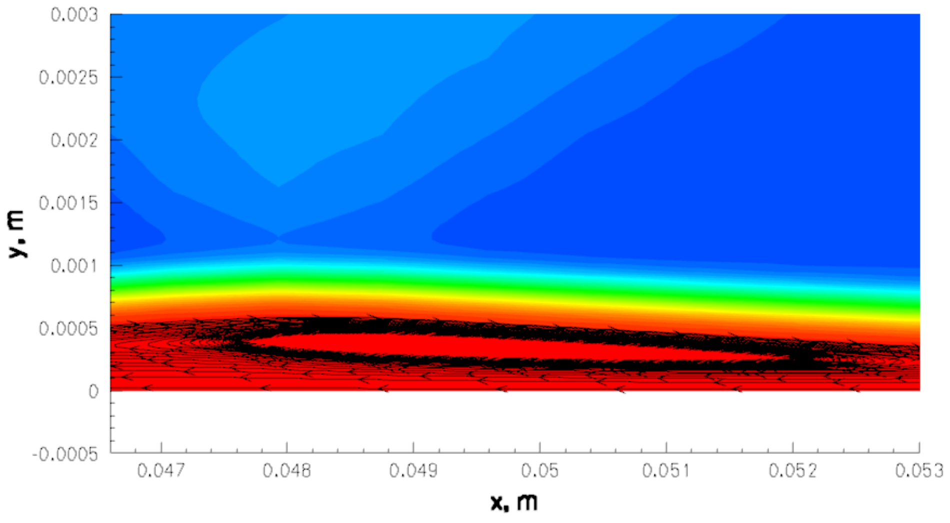


(b) Pressure, Pa

Figure 5.27: Shock-boundary layer interaction: steady-state solution computed on 108 $\times$ 270 cells.

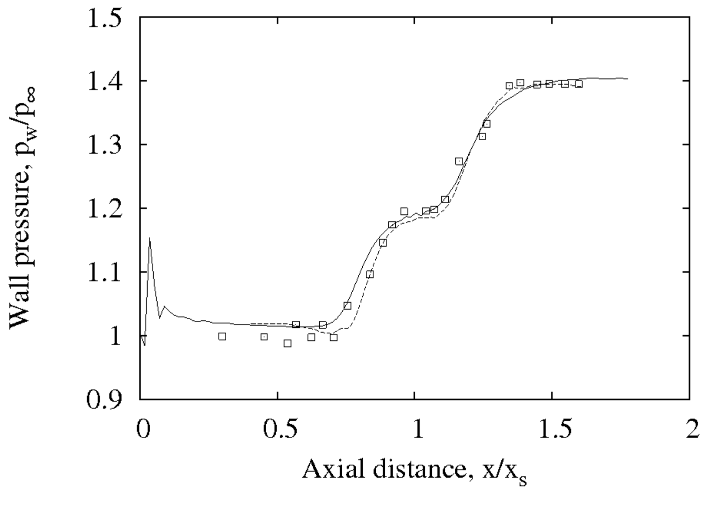


(a) Velocity profiles

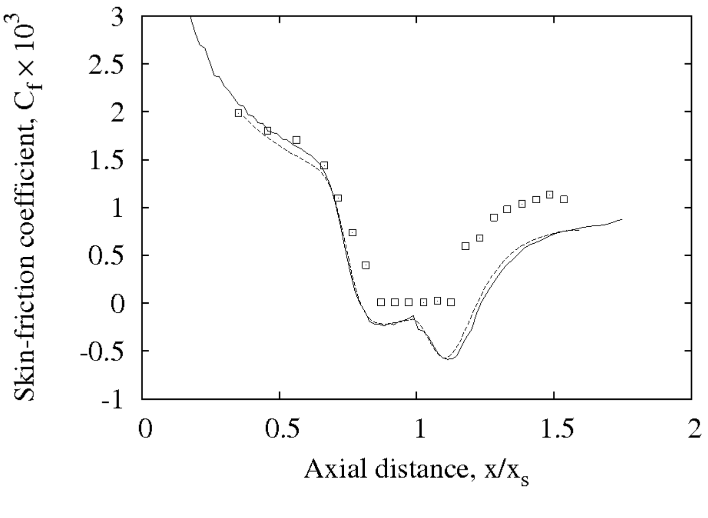


(b) Streamlines

Figure 5.28: Separation bubble in the shock-boundary layer interaction: (a) velocity profiles and (b) streamlines superimposed on temperature contours. Temperatures are expressed in Kelvin.



(a) Wall pressure coefficient



(b) Wall skin-friction coefficient

Figure 5.29: Numerical and experimental results for the shock-boundary layer interaction: □, experiment by Hakkinen *et al.* (88); —, CU5-TVD; −−, CFD by Yao *et al.* (89).

surface at a location identified by the letter R in Fig. 5.26; the external flow deviates through compression waves, which cause a second rise in wall pressure and coalesce downstream into the so-called reattachment shock.

We have considered the experimental flow conditions of Hakkinen *et al.* (88). The experiment was performed in the continuous flow supersonic wind tunnel of the Gas Turbine Laboratory of the Massachusetts Institute of Technology in 1959. The plate was equipped with static-pressure orifices, connected to mercury manometers, and a thermocouple, read by means of a potentiometer. Stagnation pressure and temperature were observed similarly, with an orifice and a thermocouple installed in the settling chamber. A small slit-mouth tube was used to measure the skin-friction by taking measurements with the tube resting on the plate. A theory for the use of small impact pressure probes as skin-friction meters states that the difference between the pressure measured by the probe resting on the surface and the undisturbed local static pressure is related to the local shear stress through a power law with exponent 5/3. Hakkinen *et al.* used this theory to calibrate the tube as a skin-friction meter, by comparing the tube readings with the measurements of an absolute floating element skin-friction meter. The instrument was not able to measure negative values of the shear stress, and so shear stress measurements are only valid before the separation and after the reattachment points.

The free stream Mach number was $M_\infty = 2$ and the Reynolds number based on free stream properties and the distance L of the shock impingement point from the plate leading edge was $Re_{\infty,L} = 2.96 \times 10^5$. The impinging shock was generated by a 3° wedge, resulting in a shock strength $p_3/p_\infty = 1.4$, where p_3 is the static pressure downstream from the SBLI. The shock impingement location was about $5cm$ from the plate leading edge. The plate was about $15cm$ long, and the interaction took place within $10cm$ from the leading edge.

In our simulation the computational domain is $(x, y) \in [-0.5cm, 8.5cm] \times [0cm, 6cm]$, and the plate leading edge is placed at $x = 0cm$. A grid independency analysis was carried out and it was found that results on a grid of 108×270 cells are converged. The grid is equally spaced in the x-direction, and exponentially stretched in y-direction with a minimum cell thickness $\Delta y_{min} = 1.7 \times 10^{-3} cm$. The supersonic inflow boundary condition was enforced on the left and top boundaries, while zero gradient was enforced on the right boundary. On the bottom boundary, the surface ahead of the plate leading edge is a slip-wall, while the plate surface is an adiabatic no-slip wall.

In Fig. 5.27 steady-state density and pressure contours are shown, and all of the features of the interaction are observed. Density and pressure distributions are similar, but while the pressure is constant across the boundary layer thickness and the separation bubble, these zones are identified by a lower density. This results from the adiabatic-wall condition: as the temperature increases from its value in the external flow to its adiabatic-wall value, while the pressure remains constant, the density decreases. The oscillation in the top-right part of both pictures results from the reflection on the top boundary of a disturbance generated at the left boundary. It is due to the fact that the impinging shock has a finite thickness in the computational domain while it has no thickness in the ghost cells, where the supersonic inflow boundary condition is applied. The pattern of this reflection is such that it does not affect the simulation of the SBLI.

In Fig. 5.28 a zoom into the shock impingement location shows details of the separated flow. Velocity profiles and streamlines clearly show the core of the separation vortex, located downstream from the impingement location, and how the bubble thins after the shock reflection.

In Fig. 5.29 the non-dimensional wall-pressure, p_w/p_∞, and the skin-friction coefficient, $C_f = 2\tau_w/\rho_\infty u_\infty^2$, are plotted as functions of the distance from the plate leading edge, normalised by the shock impingement location x_s. The present method is compared to both experimental data and the recent CFD data obtained by Yao *et al.* (89) using a fourth-order explicit-TVD FD method. The features of the pressure profile (primary pressure rise - pressure plateau - secondary pressure rise) are correctly captured, and the agreement is excellent with both experimental measurements and previous CFD data. On the skin-friction coefficient, the agreement between our CFD and Yao *et al.* is excellent. Both their CFD results and ours agree with experimental results in the region upstream from the separation but predict a wider separation bubble and a lower friction after reattachment. Similar results were obtained by Katzer (90) and Wasistho (91), and this suggests that the validity of these experimental data must be questioned. Katzer, in particular, suggested that three-dimensional effects due to the presence in the experiment of side walls could be the cause of such a discrepancy.

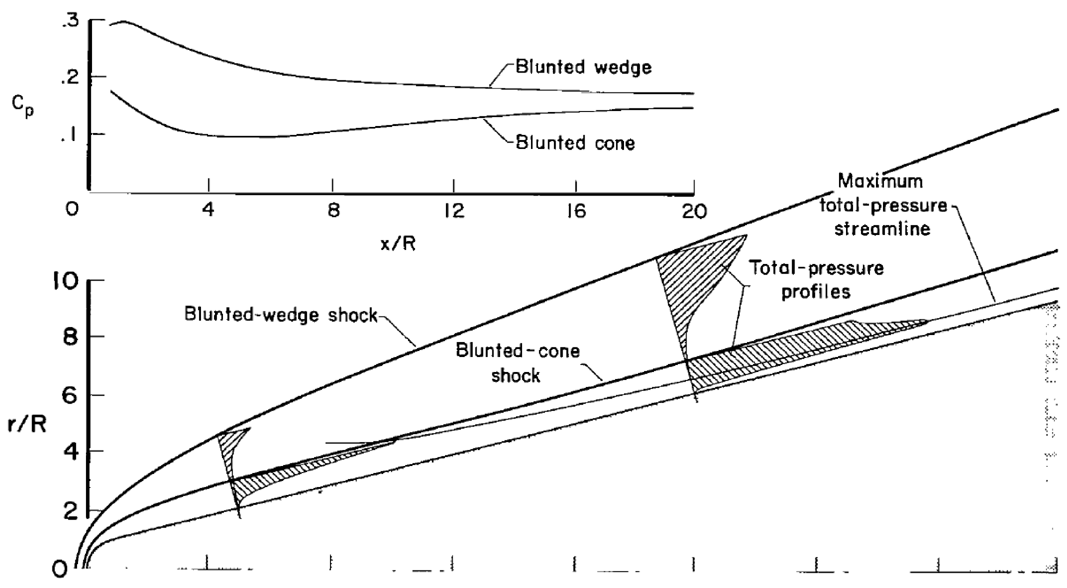


Figure 5.30: Comparison between blunted-cone and blunted-wedge hypersonic flows. Reproduced from Cleary (92).

5.4 Hypersonic blunted cone

In this section we demonstrate through a test case the capability of the current method to simulate hypersonic flows. We consider a sphere-cone geometry, common in hypersonic applications and often tested experimentally.

The hypersonic flow past a blunted-cone presents several interesting features, depicted in Fig. 5.30 that is reproduced from Cleary (92). First, the geometry is axisymmetric and so the shock wave is much closer to the body surface than in the two-dimensional case. Furthermore, the fluid passing through the strong part of the bow shock experiences a reduction in stagnation pressure and forms a low-density and high-entropy layer that, because of the conical shape, diminishes in thickness as it moves downstream. Near the nose, the flow field is similar to the one predicted by blast-wave theory (1) while, many nose radii downstream, the flow features are essentially those of a flow past a sharp cone, except for the thin entropy layer. Since the entropy layer thins moving downstream along the cone, the flow over-expands to surface pressures below the sharp cone value, and then re-compresses, with the surface pressure asymptotically approaching the sharp cone value. Finally, the thinning of the entropy layer interacts

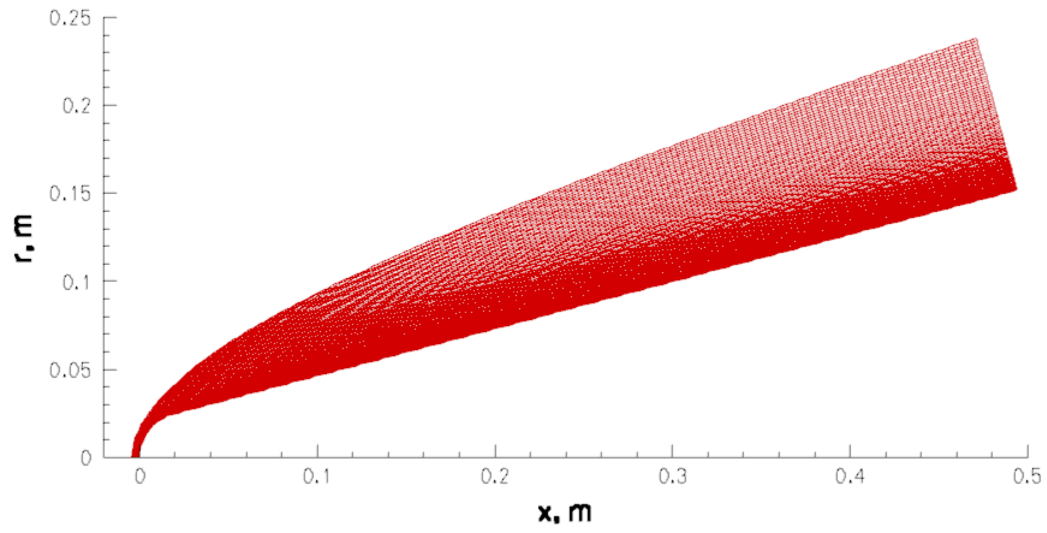


Figure 5.31: Computational grid for the hypersonic blunted cone test case.

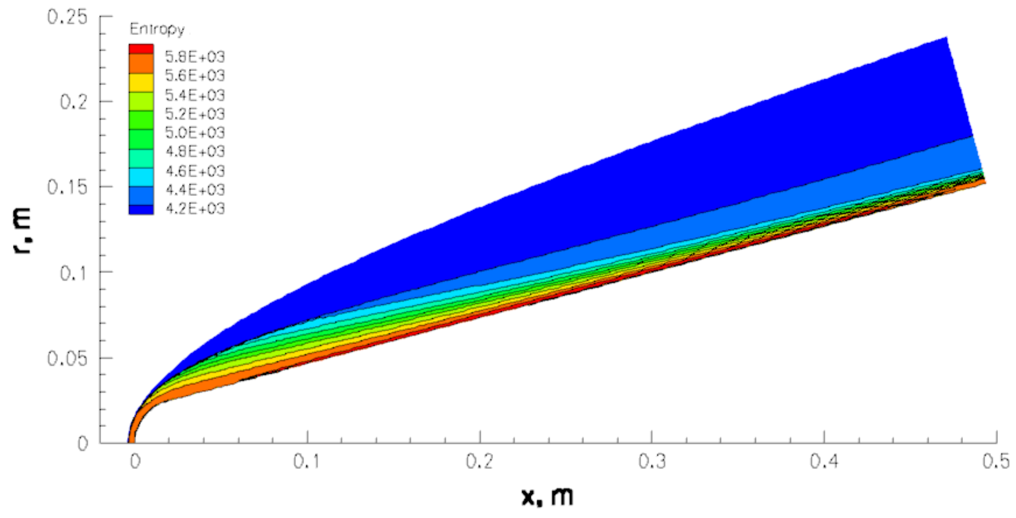
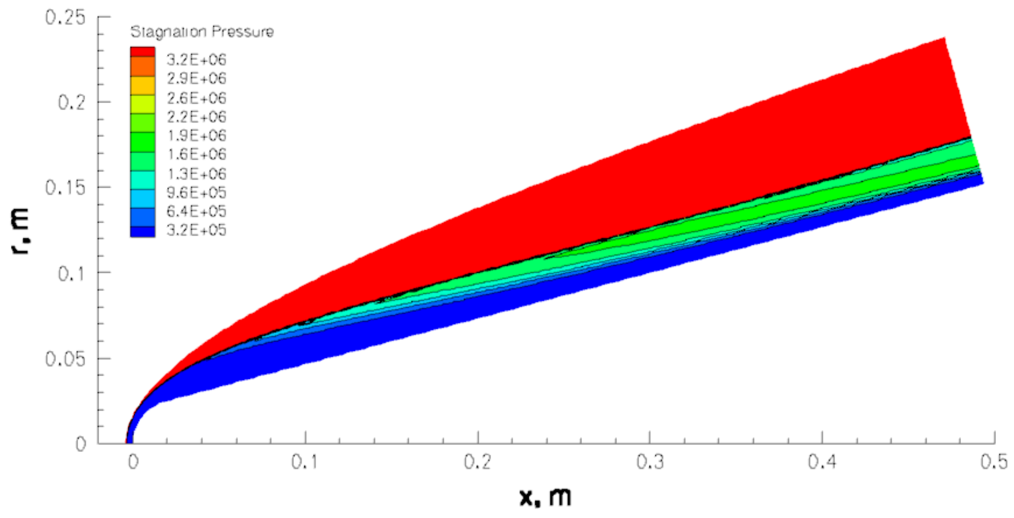
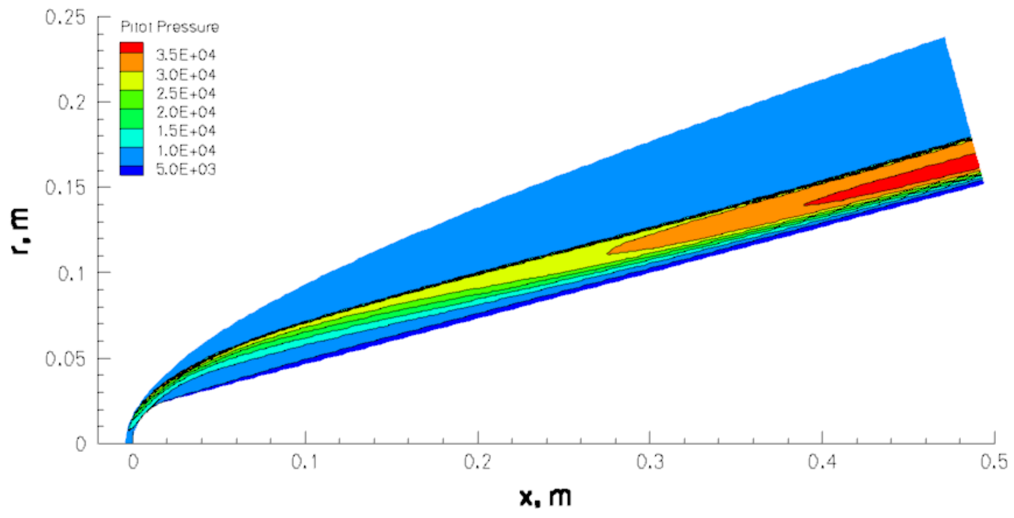


Figure 5.32: Hypersonic blunted-cone: steady-state contours of entropy, expressed in J/K.



(a) Stagnation pressure, Pa



(b) Pitot pressure, Pa

Figure 5.33: Hypersonic blunted-cone: steady-state (a) stagnation pressure and (b) Pitot pressure countour plots.

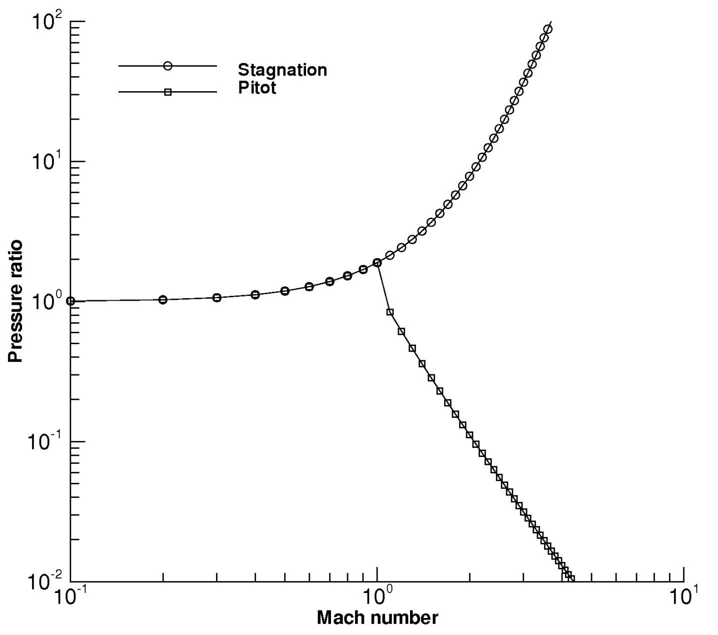


Figure 5.34: Stagnation and Pitot pressures as function of the Mach number.

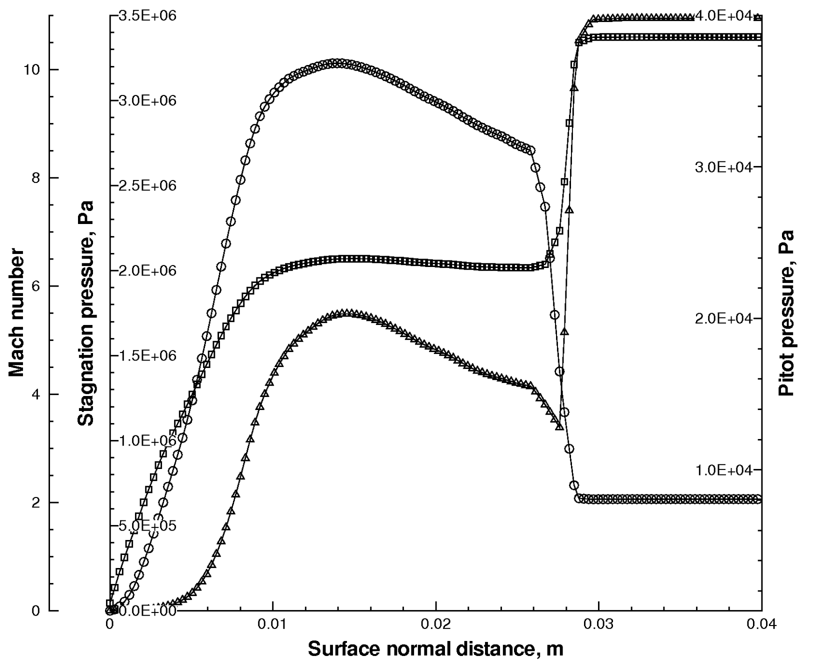
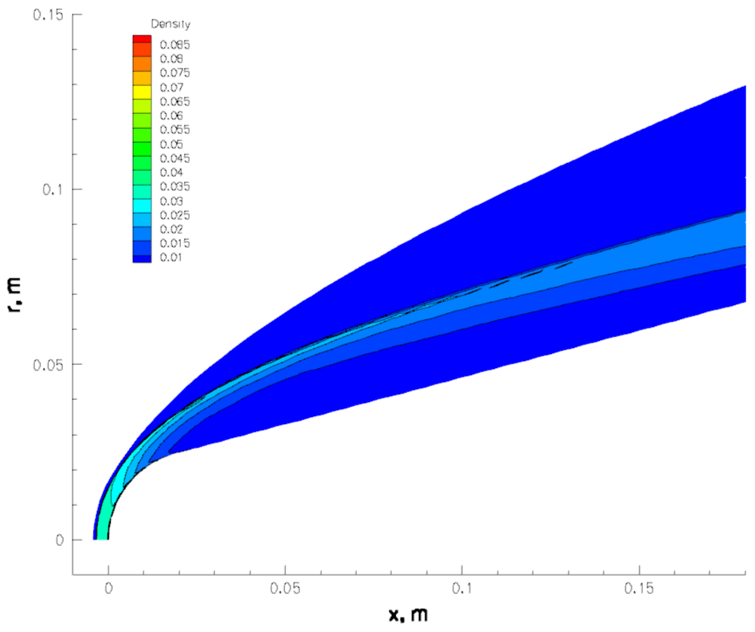
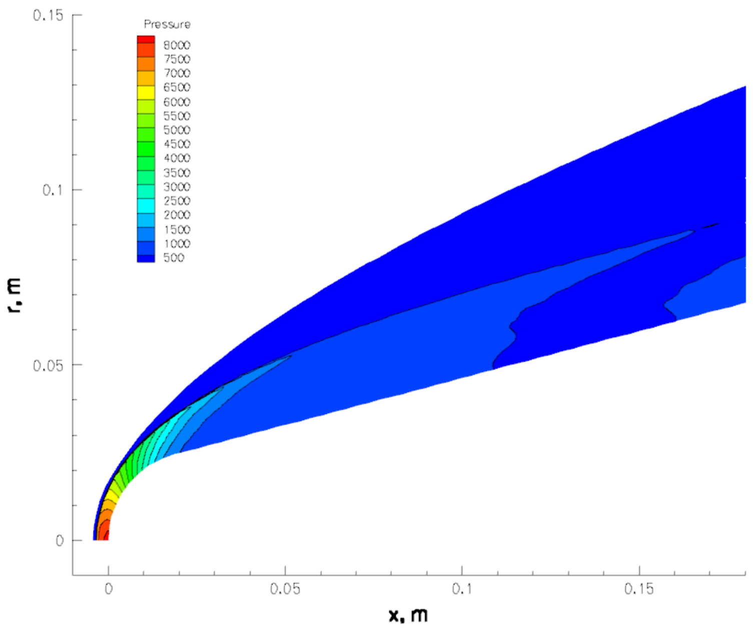


Figure 5.35: Computed flow variables at station $x/R_n = 16.69$ as a function of the distance from the cone surface: –△–, stagnation pressure, Pa; –□–, Mach number; –○–, Pitot pressure, Pa.

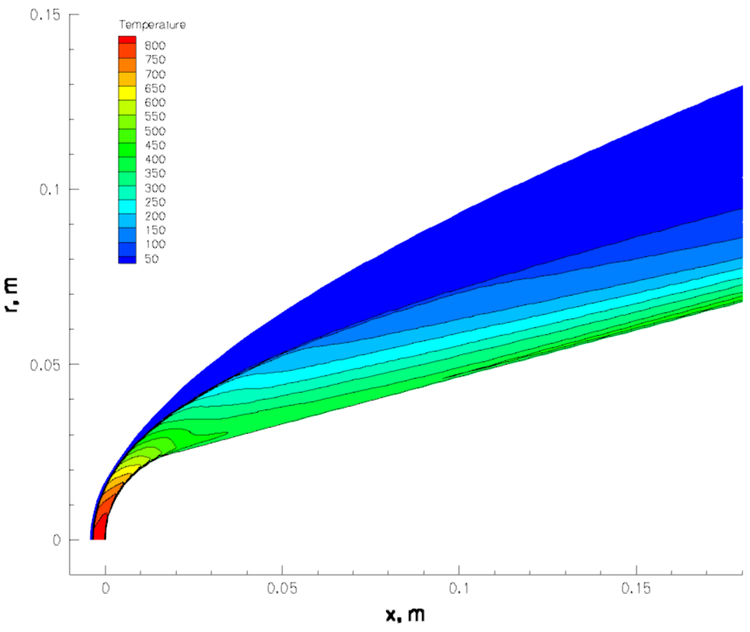


(a) Density, kg/m^3

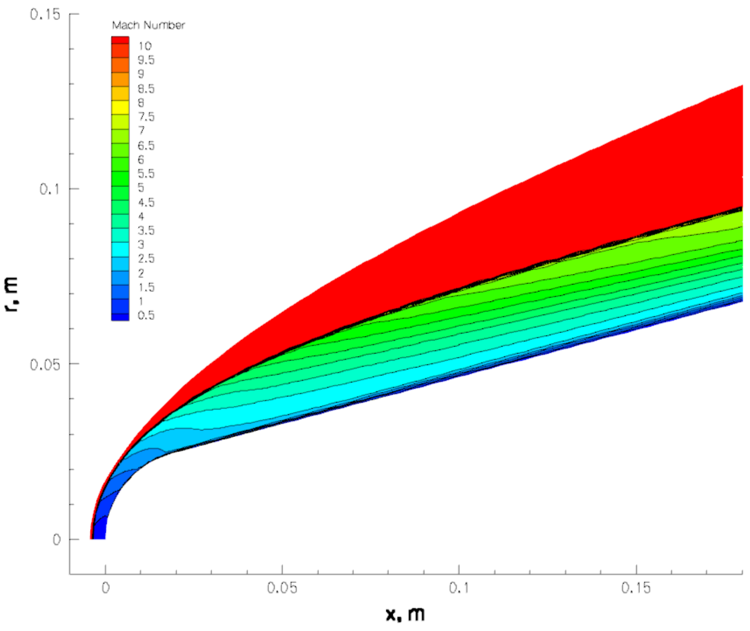


(b) Pressure, Pa

Figure 5.36: Nose of the hypersonic blunted-cone: steady-state (a) density and (b) pressure contour plots.



(a) Temperature, K



(b) Mach number

Figure 5.37: Nose of the hypersonic blunted-cone: (a) temperature and (b) Mach number contour plots.

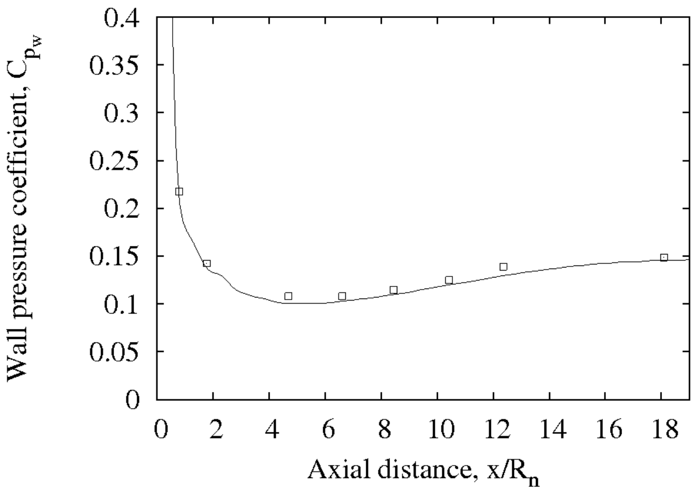
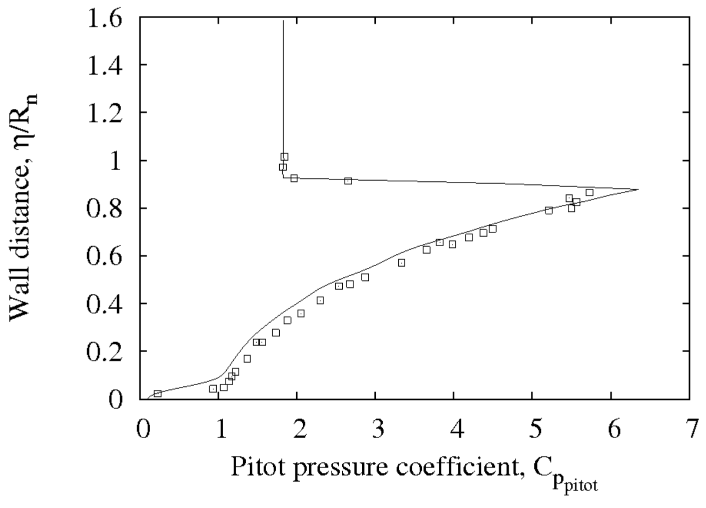


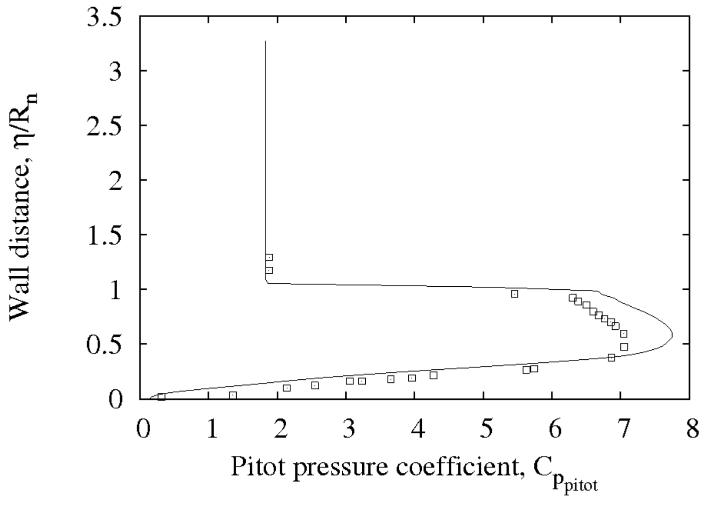
Figure 5.38: Hypersonic blunted-cone. Surface pressure coefficient: □, experiment by Cleary (92); —, present method.

with the bow shock in such a way that a high-stagnation pressure layer forms, which encloses the low-stagnation pressure layer adjacent to the surface.

Hypersonic flow conditions very often involve phenomena such as real gas effects, chemical reactions, rarefaction. We have simulated the experimental flow conditions of Cleary (92), in which the free-stream conditions were such that these phenomena could be neglected and the flow could be simply described by the non-reacting Navier-Stokes equations and the perfect gas thermodynamic model. The test was run in the Hypersonic Wind tunnel of NASA Ames in 1969. The model was a blunted-cone with 15° half-angle, a nose curvature-radius $R_n = 2.54cm$ and a base diameter $d_b = 30.48cm$. It was instrumented with orifices along the surface to measure the wall pressure, and movable pitot probes were mounted on the top at the station $x/R_n = 3.59$ and on the bottom at $x/R_n = 16.67$. The model was internally cooled. According to Cleary, surface temperatures were well below the adiabatic-wall temperature during the entire run, which was about two minutes long, but the author does not provide any value for the wall temperature. In order to evaluate the effect of varying wall temperature he made Pitot-pressure measurements both at the beginning and at the end of the run but he did not observe any significant difference. Cleary estimated the precision of the pressure coefficients to be within ±4%. In the experiment the free-stream Mach



(a) $x/R_n = 3.59$



(b) $x/R_n = 16.67$

Figure 5.39: Hypersonic blunted-cone. Pitot pressure coefficients: □, experiment by Cleary (92); —, present method.

number, Reynolds number per metre and stagnation temperature were:

$$M_\infty = 10.6, \qquad Re_{\infty,1} = 3.28 \times 10^6 m^{-1}, \qquad T_{0,\infty} = 1111K.$$

The computational domain and grid for our simulation are shown in Fig. 5.31. The cone has no angle of attack, so the axisymmetric formulation is used and only half of the flow field is computed. The boundary conditions are: slip-wall on the axis, supersonic inflow at the top boundary, zero-gradient at the right boundary, no-slip isothermal-wall at the body surface. The wall temperature is set to $300K$. The top boundary is a parabola, with the minimum and maximum distances from the body surface $\eta_{\min}/R_n = 0.18$ and $\eta_{\max}/R_n = 3.5$, respectively. A grid independency analysis was performed and results were found to be converged on a grid of 158×107 cells. The grid is uniform along the body-fitted coordinate ξ, while it was exponentially stretched in the coordinate normal to the cone surface η, with minimum cell thickness $\Delta\eta_{\min}/R_n = 4 \times 10^{-4}$. The fluid is air, with isentropic index $\gamma = 7/5$, gas constant $R_{air} = 287.06 m^2/s^2K$, and Prandtl number $Pr = 0.725$.

In Fig. 5.32 entropy contours for the computed solution are shown. An entropy layer is clearly created at the blunt nose and it thins as the flow moves downstream along the conical part of the geometry. In Fig. 5.33(a) stagnation pressure contours are shown. The low-stagnation-pressure layer on the body surface is clearly visible, as well as the high-stagnation-pressure layer enclosing it. In Fig. 5.33(b) Pitot pressure contours are shown. The Pitot pressure is the pressure measured by a Pitot probe, i.e. the local stagnation pressure if the flow is subsonic, or the local stagnation pressure downstream of a normal shock if the flow is supersonic. This can be expressed by the formula:

$$\frac{p_{pitot}}{p} = k\left(1 + \frac{\gamma-1}{2}M^2\right)^{\frac{\gamma}{\gamma-1}},$$

where:

$$k = \begin{cases} 1 & \text{if } M \le 1, \\ \left(\frac{\frac{\gamma+1}{2}M^2}{1+\frac{\gamma-1}{2}M^2}\right)^{\frac{\gamma}{\gamma-1}} \left(\frac{1}{\frac{2\gamma}{\gamma+1}M^2 - \frac{\gamma-1}{\gamma+1}}\right)^{\frac{\gamma}{\gamma-1}} & \text{if } M > 1. \end{cases}$$

We introduce this quantity because the experimental data supplied by Cleary are Pitot probe readings.

Pitot pressure contours look very different from stagnation pressure contours because, fixing the static pressure, while the stagnation pressure monotonically increases with the Mach number, the Pitot pressure drops heavily for Mach numbers between 1 and 3, and then approaches zero asymptotically. This can be seen in Fig. 5.34, where Pitot and stagnation pressure ratios (p_{pitot}/p and p_0/p) are plotted as a function of the Mach number. The most remarkable effect of this dependency on the blunted cone flow field is a very high Pitot pressure layer appearing in the conical part of the geometry. In Fig. 5.35 Mach number, stagnation pressure and Pitot pressure are shown as functions of the distance from the cone surface at $x/R_n = 16.67$. Crossing the shock, the Mach number drops from 10 to 6, causing the stagnation pressure to reduce by 2/3 and the Pitot pressure to treble. In the region between the shock and the entropy layer, the Mach number is constant, and both stagnation and Pitot pressures rise as the static pressure rises. Finally, in the entropy layer both stagnation and Pitot pressures drop, although the Pitot pressure curve is less steep for supersonic Mach numbers.

In Figs. 5.36 and 5.37 steady-state contours of density, pressure, temperature and Mach number at the nose are shown. The strong bow shock is captured sharply, no spurious oscillations are observed and the flow features in the shock layer are smooth. In Fig. 5.36(a), the highest values of density are not visible as they are located in a very thin layer enclosing the wall. Fig. 5.36(b) clearly shows the pressure diminishing below the sharp cone value on the blunt nose and increasing further downstream.

Our results are compared to the experimental measurements in Figs. 5.38 and 5.39 in terms of pressure coefficient:

$$C_p = \frac{p - p_\infty}{1/2\rho u_\infty^2}. \tag{5.13}$$

In Fig. 5.38 the wall pressure coefficient (p is the wall pressure p_w in Eq. 5.13) is plotted as function of the axial coordinate. Our CFD compares very well with the experimental data; the over-expansion below the sharp cone pressure as well as the successive re-compression are correctly captured. In Fig. 5.39, the Pitot pressure coefficient (p is the Pitot pressure p_{pitot} in Eq. 5.13) as a function of the distance from the cone surface is plotted at stations $x/R_n = 3.59$ and $x/R_n = 16.67$. At $x/R_n = 3.59$ (Fig. 5.39(a)), the Pitot pressure experiences a jump across the shock, then diminishes across shock and entropy layers, and drops more quickly in the boundary layer. All of these features are correctly captured in our simulation and the agreement with the measurements is good. At $x/R_n = 16.67$ (Fig. 5.39(b)), the Pitot pressure increases in the shock layer, and diminishes in the entropy and boundary layers. In the shock layer our CFD predicts Pitot pressure coefficients about 10% higher than the ones measured (that are accurate within $\pm 4\%$), while there is good agreement within the entropy and boundary layers. In fact, the coefficients measured in the shock layer are lower than what an inviscid calculation would suggest, and Cleary explains this discrepancy as a not-well-understood viscous effect, although no strong viscous effects are expected to take place in the shock layer.

5.5 Summary

In this chapter we have presented inviscid and viscous flow simulations performed using our compact-TVD method.
Inviscid flow simulations are performed to investigate the resolution properties of the method. Typically our compact-TVD method shows better resolution than methods of comparable accuracy that employ a WENO scheme to capture discontinuities.
Viscous flow simulations are performed to validate the code. Our numerical results are in excellent agreement with exact solutions of smooth compressible flows, namely the shock-layer and the supersonic boundary layer over a flat plate. Comparison to experimental results is provided for two simulations, namely the shock-boundary layer interaction and the hypersonic flow past a blunted cone. For the shock-boundary layer interaction, wall pressures are in good agreement with the experimental data. A discrepancy between our CFD and experimental values of the friction coefficient after reattachment is found. Howerver, our prediction agrees with independent CFD data by other authors (89, 90, 91). For the hypersonic flow past a blunted cone, our results are in good agreement with the experimental data, although a minor discrepancy is observed at some points downstream from the conical shock.

6

HPC applications of the compact-TVD method

In Chapter 4 we presented two algorithms to speed up the compact-TVD method on HPC architectures, such as distributed-memory clusters of multi-core processors (section 4.1) and GPUs (section 4.2). These algorithms are new contributions of this work, and their potential and limitations are studied in this chapter.

In section 6.1 we study the impact of the perturbation introduced on the original linear system in the multi-block NS solver, and quantify the achievable speed-ups. In section 6.2 we validate the GPU implementation of the NS solver and study its performance.

6.1 Application of the hybrid multi-task and multi-thread NS solver

The hybrid NS solver presented in section 4.1 embeds the slab/drawer partitioning (34) into a structured-block decomposition algorithm. The multi-block algorithm, namely the CU5-MB method, substitutes boundary closures with explicit interface-boundary formulas at block-interface boundaries. This procedure introduces a perturbation on the linear system to compute the numerical flux function. In this section we now present five test cases to study how this perturbation impacts the computed results. For each case we compute single- and multi-block solutions and compare them. The simulations have been carried out using the CU5-TVD-MB code.

In section 6.1.1 we present two one-dimensional Riemann problems. They are the most challenging Riemann problems among those suggested by Toro (93) to test high-order methods for gas dynamics.

In section 6.1.2 we simulate supersonic flow past a cone-cylinder geometry. This test was used by Sengupta *et al.* (33) to assess the accuracy of their multi-block compact method. The sharp leading edge, the expansion corner between the cone and the cylinder, and the heavy partitioning (each block comprises only 9 internal nodes in the axial direction) challenge our multi-block numerical method.

In section 6.1.3 we simulate the interaction between a shock wave and a low-density jet. This test was used by Don and Quillen (94) to assess the accuracy of spectral and ENO schemes. The shock-jet interaction results in a very complex flow structure, and the improved resolution properties of the compact formula over the explict one are expected to manifest in the simulated flow field. Furthermore, the shock-jet interaction results in waves continuously crossing block-interface boundaries and interacting. For these reasons, this is the right test to identify any loss in resolution or failure in wave-propagation of the multi-block method.

Finally, in section 6.1.4 we use the shock-jet interaction case to study the parallel performance of the multi-block method, isolating its distributed- and shared-memory components.

6.1.1 One-dimensional Riemann problems

We consider the set of four one-dimensional Riemann problems (RP) designed by Toro (93) to test high-order methods for gas dynamics. These tests present different challenges, and generally a numerical method fails on one of them, at least. Two constant states, $P_L = (\rho_L, u_L, p_L)^T$ and $P_R = (\rho_R, u_R, p_R)^T$, are separated by a discontinuity at $x = 0.5$. The states are listed in Table 6.1 for all tests. Exact and numerical

RP	ρ_L	u_L	p_L	ρ_R	u_R	p_R
1	1	0.75	1.0	0.125	0.0	0.1
2	1	-2.0	0.4	1.0	2.0	0.4
3	1	0.0	1000.0	1.0	0.0	0.01
4	5.99924	19.5975	460.894	5.99242	-6.19633	46.0950

Table 6.1: Initial condition for the Riemann problems (RP) designed by Toro (93).

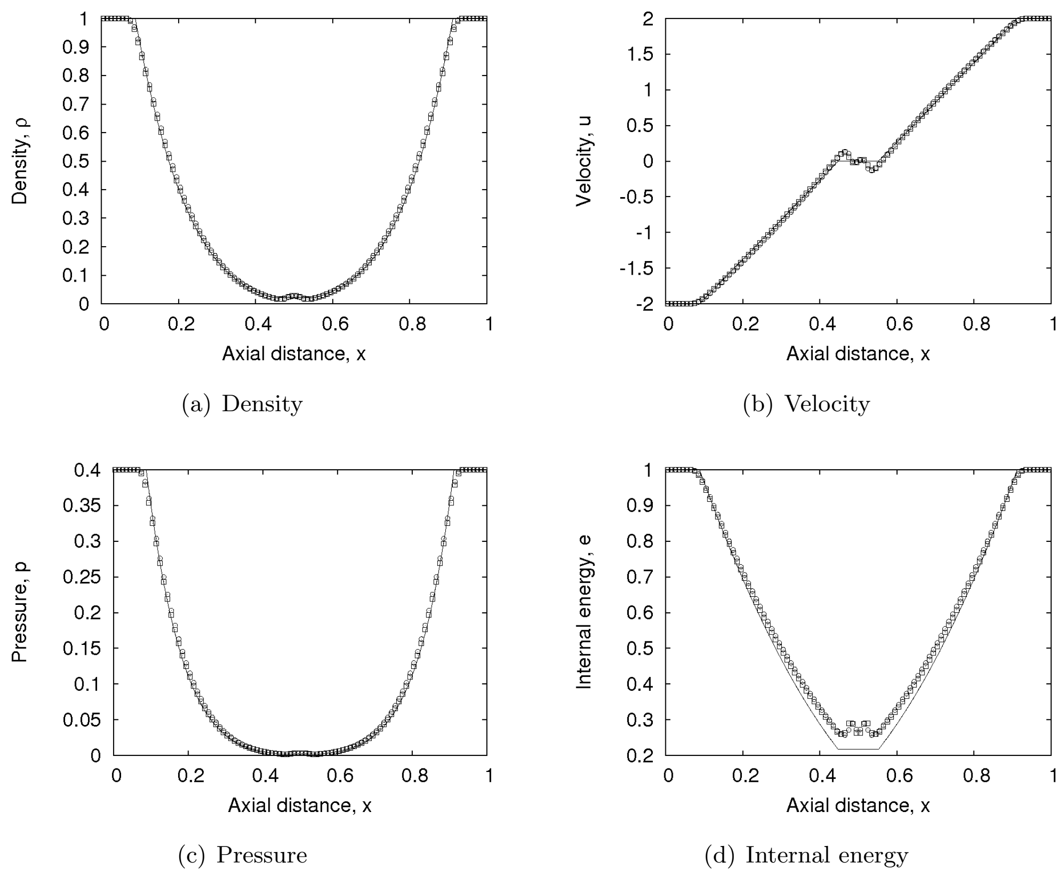
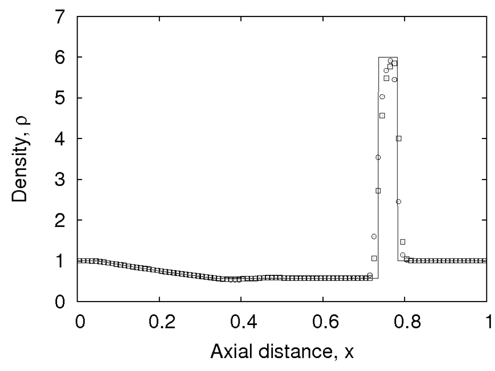


Figure 6.1: Results for RP_2 at time $t = 0.15$: ◯, multi-block solution; □, single-block solution; —, exact solution.

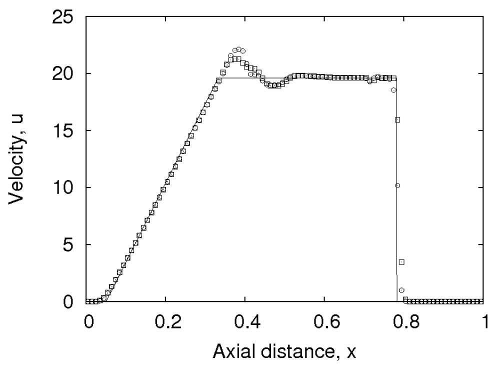
solutions are found in the spatial domain $0 \leq x \leq 1$. Numerical solutions are computed on a uniform grid of 100 cells using both single- and multi-block methods. For the multi-block simulation, the domain is partitioned into four equally sized blocks, each one comprising 25 internal nodes and 3 interface-boundary nodes per block interface. The two solutions are compared against each other and against the solution computed by an exact Riemann solver.

RP_1 and RP_4 are solved successfully and the results are not shown here: single- and multi-block solutions are identical, and both follow perfectly the exact solution. As with the Lax shock-tube presented in section 5.1.1, shocks are captured in two points and contacts in three.

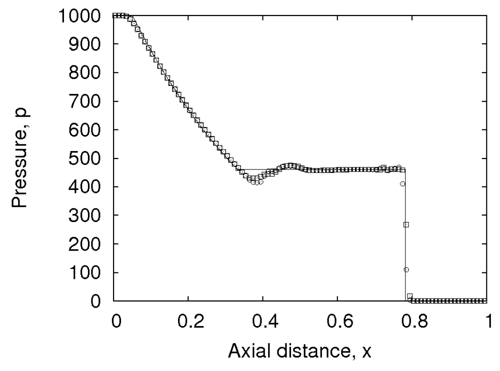
The results for RP_2 and RP_3 are shown in Figs. 6.1 and 6.2, respectively. These



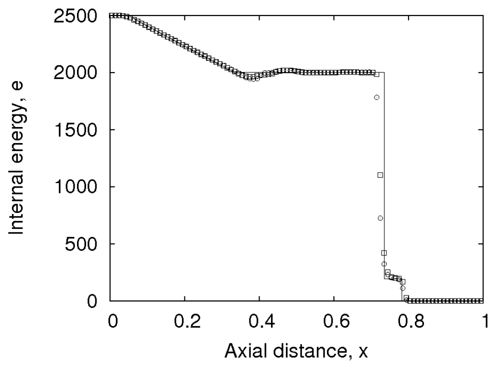
(a) Density



(b) Velocity



(c) Pressure



(d) Internal energy

Figure 6.2: Results for RP_3 at time $t = 0.15$: ◯, multi-block solution; □, single-block solution; —, exact solution.

two tests are more challenging and the compact-TVD method shows its limits.

The exact solution of RP_2 consists of two symmetric rarefaction waves and a contact wave of zero speed. The region between the rarefaction waves is close to vacuum, which makes this problem a suitable test for assessing the performance of numerical methods for low-density flows. The stable Courant number is $CFL = 0.1$ for this test, while $CFL = 0.4$ is stable for the other tests. Our compact-TVD method fails to accurately predict the near-vacuum condition. However, this failure is common to many numerical methods and ours performs better than many others considered by Toro (93). The TVD limiter fails to damp spurious oscillations at the tail of the rarefaction wave: oscillations are evident in the internal energy profile, mainly due to the oscillation in the density, and are quite noticeable in the velocity profile. The difference between the multi-block and single-block solution is visible, although very small. The over-prediction of the internal energy as well as the amplitude of the spurious oscillation in the near-vacuum conditions is not significantly affected by the domain partitioning.

The exact solution of RP_3 consists of a right-travelling shock wave, a contact surface and a left-running rarefaction wave. This structure is similar to that in the Lax test, but the shock is stronger and the region between the shock and the contact is thinner. The challenge in this test case is resolving the region between the contact discontinuity and the shock wave. The compact-TVD method is able to accurately predict the density peak, although the uniform density region between the contact discontinuity and the shock wave is not captured. Spurious oscillations are observed at the tail of the expansion wave, as in Test 2, and visible in pressure, velocity and internal energy profiles. Single- and multi-block solutions differ slightly in this oscillation region and in the region across the discontinuities. The region across the discontinuities is about 8 nodes wide, comparable to the numerical thickness of the discontinuities (about 3 nodes each). Therefore, the difference between single- and multi-block solution in that region is irrelevant, as the only meaningful quantities (width of the region and density peak) are consistent between the two.

6.1.2 Supersonic flow past a cone-cylinder configuration

In this section we examine the effect of the multi-block partitioning on the simulation of wall-bounded flows. We consider axisymmetric supersonic flow past a cone-cylinder configuration at zero angle of attack. This test case was used by Sengupta *et al.* (33) to validate their multi-block compact method. More precisely, the authors showed that, unlike Gaitonde's multi-block compact method (32), theirs does not distort smooth flow features.

The problem is dimensionless: lengths are normalised with respect to the cylinder diameter, D_{cyl}, density and temperature with respect to their respective free-stream values, ρ_∞ and T_∞, velocity components with respect to the free-stream speed of sound, a_∞, and the pressure with respect to $\rho_\infty a_\infty^2$. The free-stream Mach and Reynolds numbers are, respectively,

$$M_\infty = \frac{u_\infty}{a_\infty} = 4, \qquad Re_\infty = \frac{\rho_\infty u_\infty D_{cyl}}{\mu_\infty} = 4.48 \times 10^6.$$

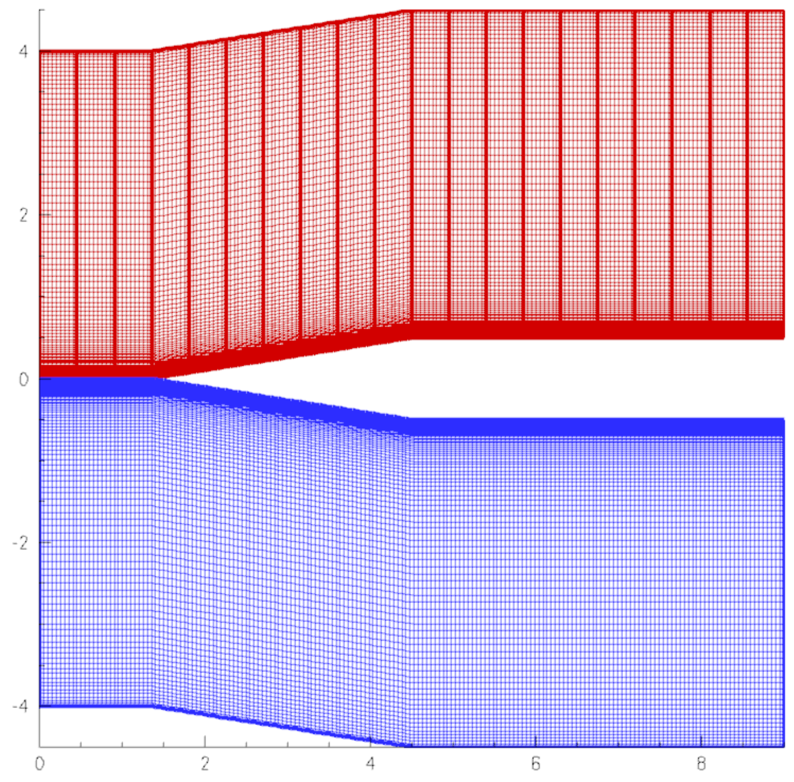
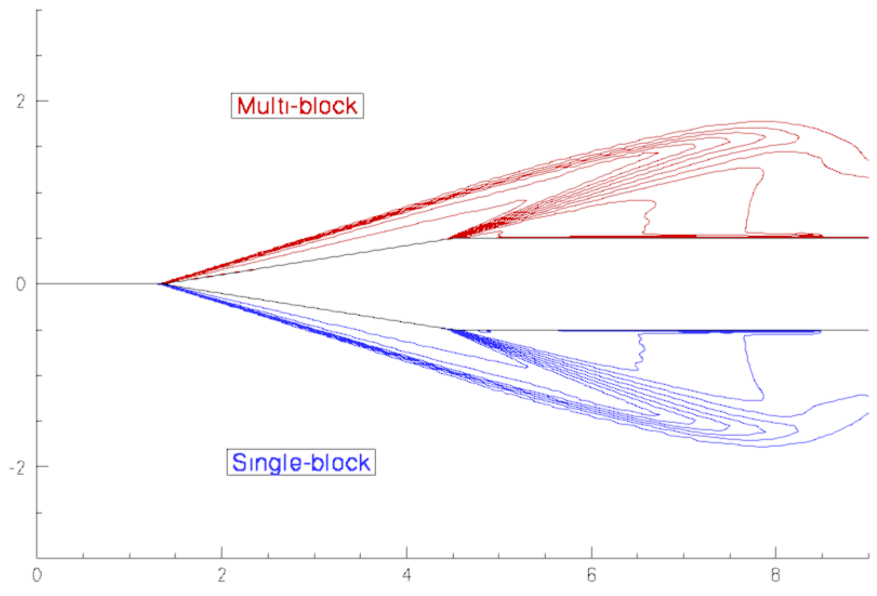
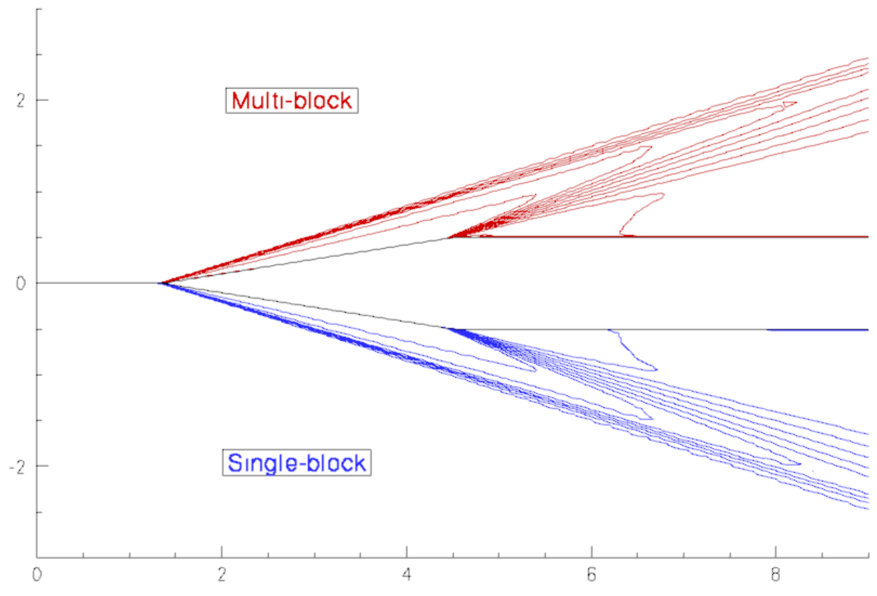


Figure 6.3: Computational grids for the supersonic flow past a cone-cylinder geometry. The flow is simulated using both a multi-block mesh, shown in the upper part, and a single-block mesh, shown in the lower part.

The cone has a semi-angle $\delta_c = 9.46°$ and a base diameter $D_{cyl} = 1$. The computational grid is shown in Fig. 6.3: it consists of 180×112 cells, and is uniform in the axial direction, while exponentially stretched in the radial direction, with minimum cell thickness $\Delta y_w = 4 \times 10^{-4}$. Both a single- and a multi-block grid are considered. The multi-block grid is partitioned along the axial direction only, and consists of 20 blocks, each one comprising 9×112 internal nodes. Such a heavy partitioning, with only a few nodes per block in the axial direction, aims to amplify any difference in the behaviour of single- and multi-block solvers. The supersonic inflow condition is imposed at left and top boundaries, while the supersonic outflow condition is imposed at the right boundary. A slip-wall condition is imposed at the axis, while the cone-cylinder surface is modelled as a no-slip adiabatic wall. The simulation is started impulsively, i.e. the free-stream

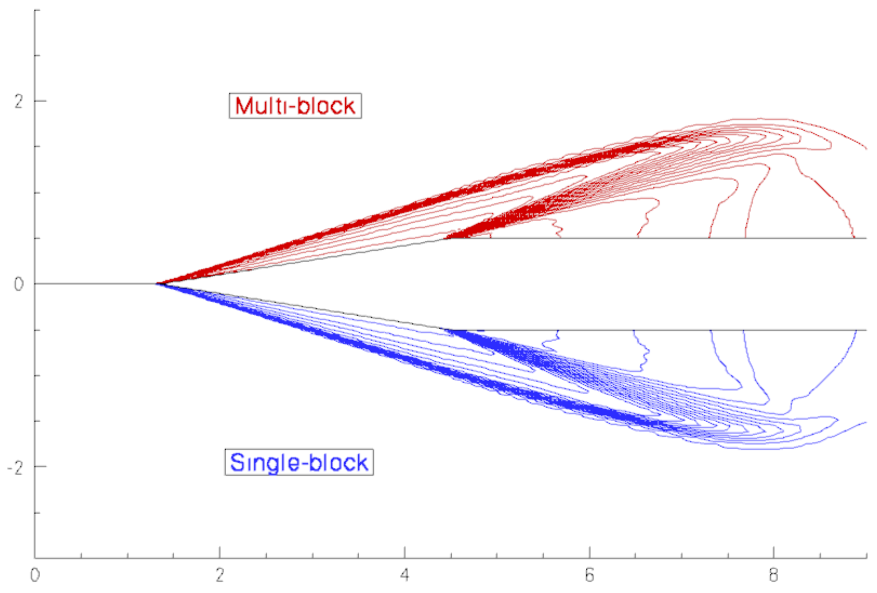


(a) Density at time $t = 1$

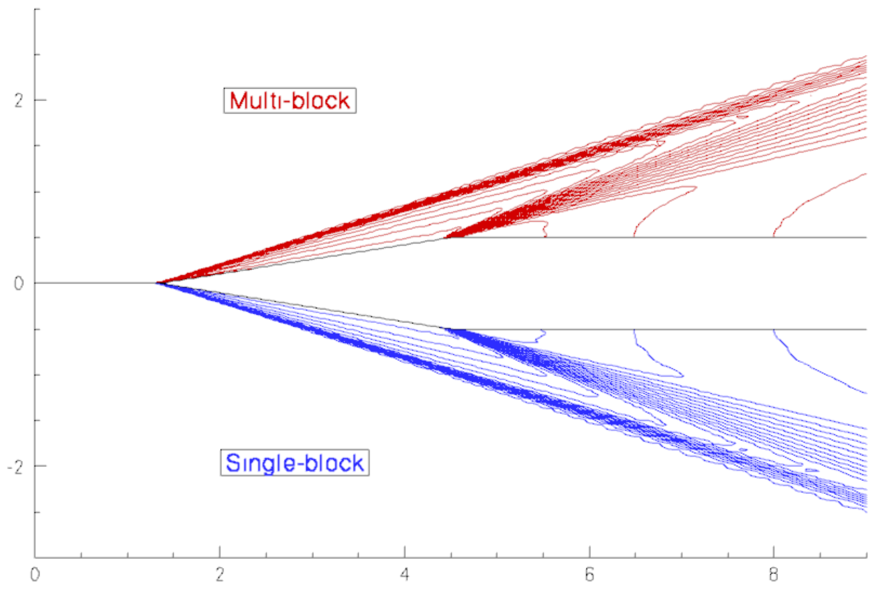


(b) Density at time $t = 8$

Figure 6.4: Mach 4 flow past a cone-cylinder geometry: density contours at indicated times, 20 equally-spaced levels between 0.1428 and 1.428. Comparison between multi-block (upper part) and single-block (lower part) solutions.

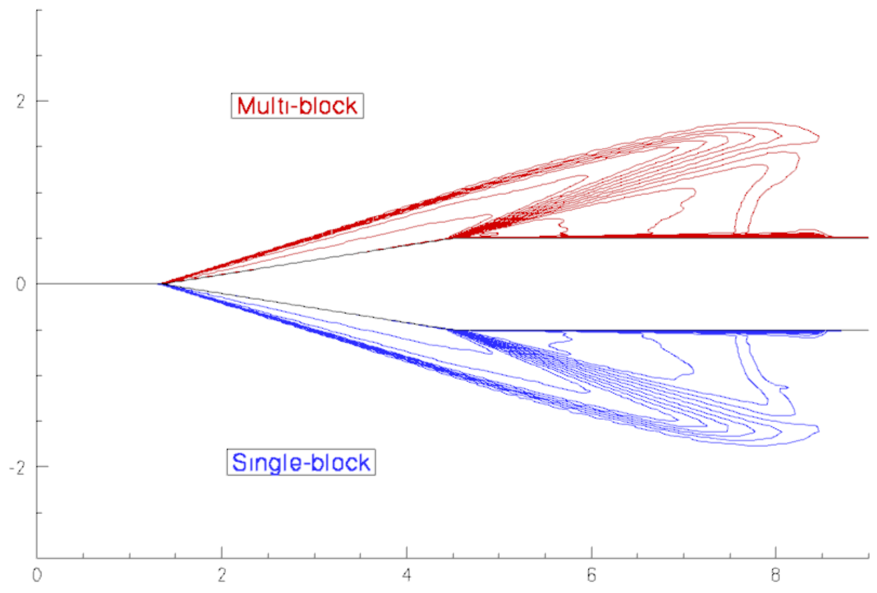


(a) Pressure at time $t = 1$

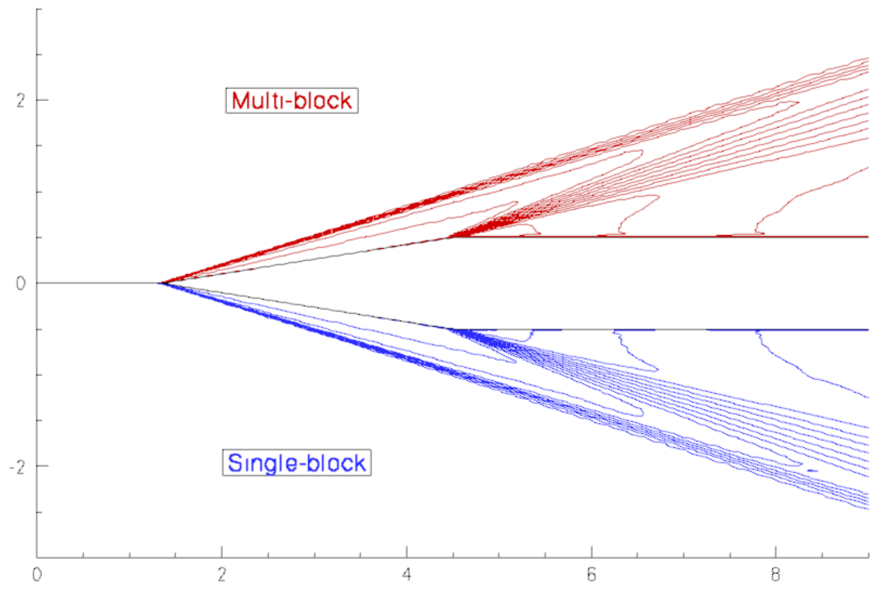


(b) Pressure at time $t = 8$

Figure 6.5: Mach 4 flow past a cone-cylinder geometry: pressure contours at indicated times, 24 equally-spaced levels between 0.98 and 2.66. Comparison between multi-block (upper part) and single-block (lower part) solutions.

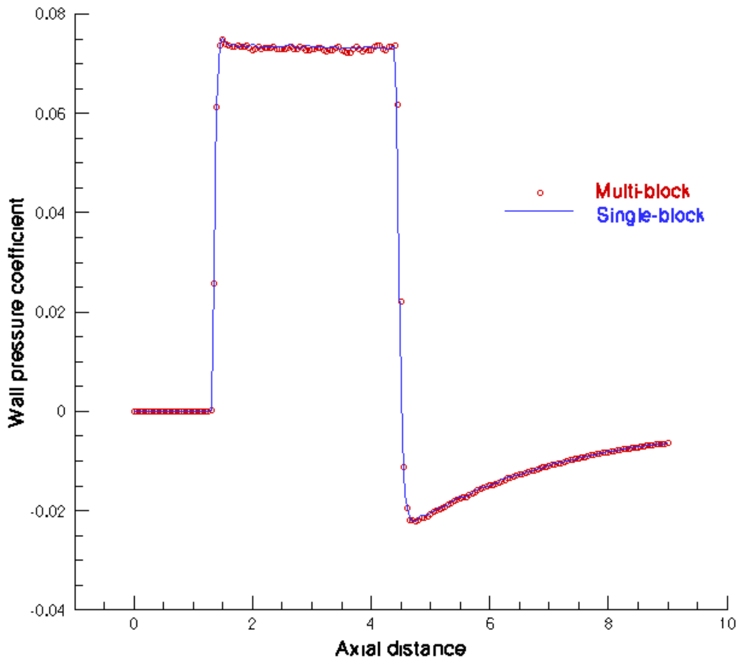


(a) Mach number at time $t = 1$

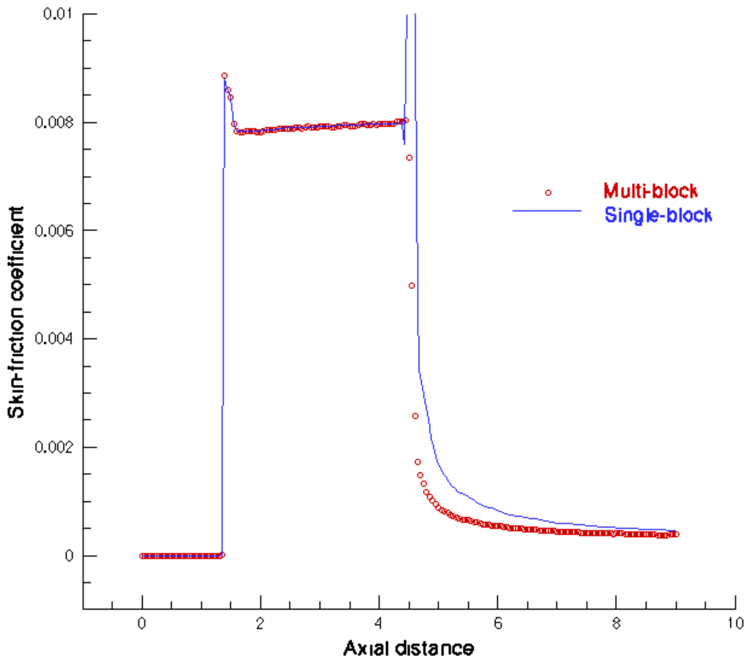


(b) Mach number at time $t = 8$

Figure 6.6: Mach 4 flow past a cone-cylinder geometry: Mach number contours at indicated times, 96 equally-spaced levels between 0.04 and 4.6. Comparison between multi-block (upper part) and single-block (lower part) solutions.

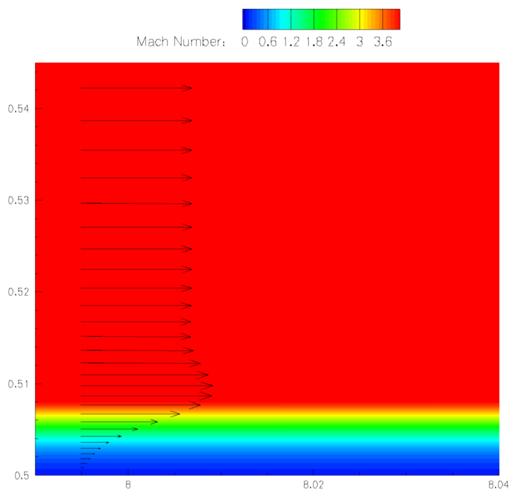


(a) Wall-pressure coefficient, $C_{p,w}$

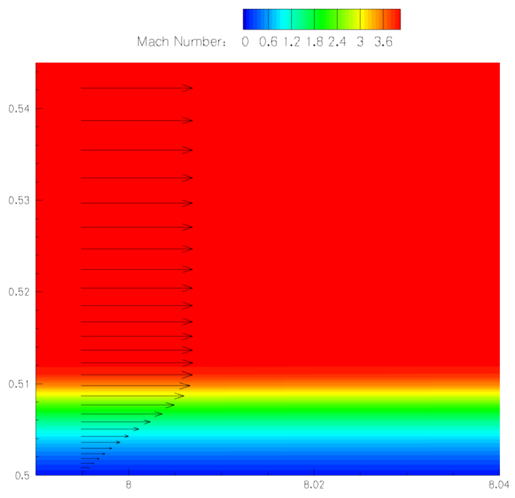


(b) Skin-friction coefficient, C_f

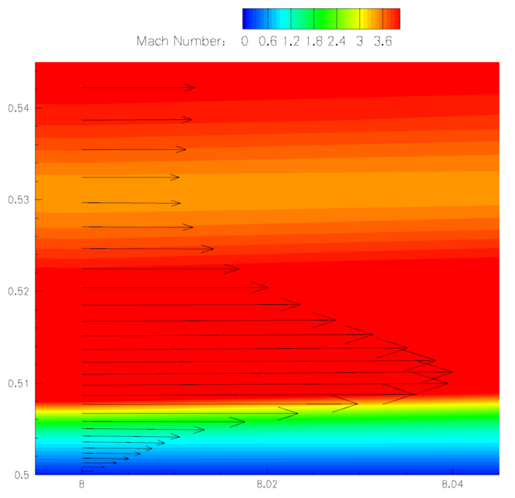
Figure 6.7: Wall coefficients for Mach 4 flow past a cone-cylinder geometry: comparison between multi- and single-block solutions.



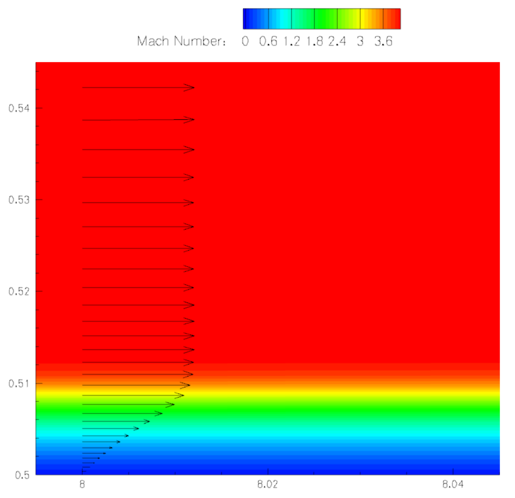
(a) Single-block, global characteristic decomposition



(b) Single-block, local characteristic decomposition



(c) Multi-block, global characteristic decomposition



(d) Multi-block, local characteristic decomposition

Figure 6.8: Effect of characteristic decomposition and domain partitioning on the simulation of the supersonic flow past a cone-cylinder configuration. Velocity profiles and Mach number contours are shown at a station on the cylinder surface.

conditions:

$$\rho_\infty = 1, \qquad u_\infty = 4, \qquad v_\infty = 0, \qquad p_\infty = 5/7,$$

are imposed everywhere in the flow-field at time $t = 0$, and the time-evolution is computed up to $t = 8$.

In Figs. 6.4–6.6 the time-evolutions computed on single- and multi-block grids are compared in terms of density, pressure and Mach number. The solution is shown at times $t = 1$ and $t = 8$, in order to compare single- and multi-block results both in the transient and at the steady state. The steady flow has a rather simple structure: there is a conical shock attached to the cone leading-edge and an expansion fan at the corner between the cone and the cylinder. The flow over-expands at this corner and re-compresses along the cylinder. The agreement between our single- and multi-block solutions is good both in the transient and at the steady state. However, Figs. 6.4–6.6 only provide information on the behaviour of the external flow. Information on how the boundary layer is captured is contained in Figs. 6.7 and 6.8. Fig. 6.7 shows the wall coefficients as functions of the axial coordinate. While the wall-pressure profiles match perfectly (Fig. 6.7(a)), the single-block method computes a consistently higher friction at the expansion corner (Fig. 6.7(b)). Single- and multi-block skin-friction coefficients agree up to the expansion corner; at the corner the single-block solution contains a non-physical peak, while the multi-block solution smoothly evolves into a cylindrical flow; agreement between the two solutions is slowly recovered downstream. This behaviour confirms the difficulty the compact-TVD method has in capturing expanding flows, as shown by the Riemann problems in sections 5.1.1 and 6.1.1. An interesting finding of this study is that, for viscous corner-flows, the block-partitioning eases this problem. This can be explained as follows: in the multi-block solution a block-interface boundary is located at the corner, where an explicit formula is used; this formula is more dissipative than the compact formula used at internal nodes, and so stabilises the computed solution.

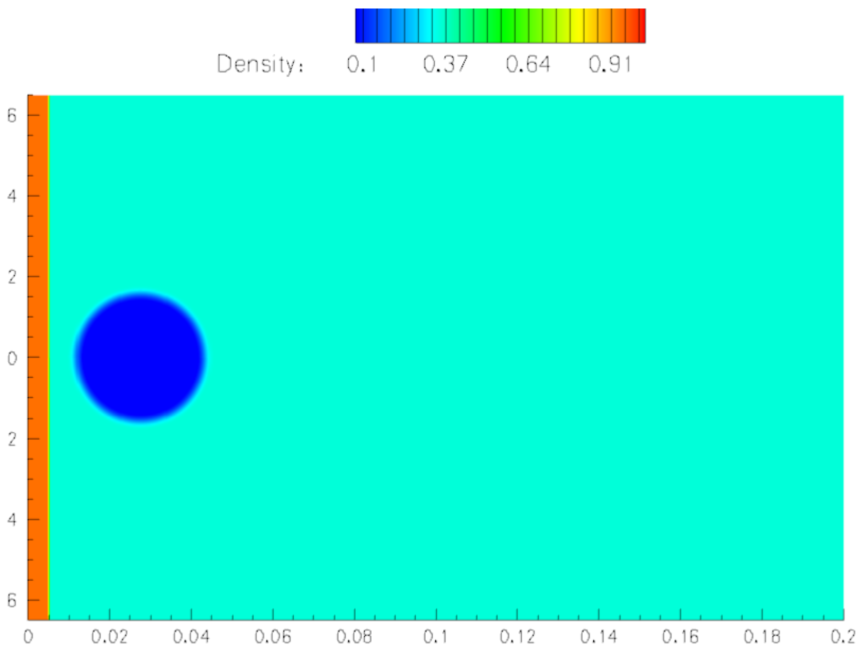
Fig. 6.8 shows Mach number contours and velocity profiles in the boundary layer at a late station along the cylinder. At this station the error introduced at the corner in the single-block solution has vanished, and so single-block (Fig. 6.8(b)) and multi-block (Fig. 6.8(d)) solutions clearly agree. Figs. 6.8(a) and 6.8(c) show results computed **without** the modification we introduce on the characteristic decomposition, described in section 2.3.3. Without our modification, the single-block solver cannot be started impulsively: negative values of pressure and density at the corner are computed for any arbitrarily small time step. We ran the first 100 time steps at first-order accuracy and used the computed solution to initialise the high-order simulation. Furthermore, the steady state solution contains an overshoot at the border of the boundary layer, clearly visible in Fig. 6.8(a). We explain this phenomenon as follows. The existence of the characteristic variables for the Euler equations is related to the conservation of entropy. Across the boundary layer, entropy is not conserved and therefore the characteristic equations should not be used instead of the Euler equations. Tu and Yuan's limiting strategy relies on the use of the characteristic equations along a grid line and fails to suppress some spurious oscillations. These spurious oscillations are greatly amplified by the block-partitioning, as shown in Fig. 6.8(c), and dramatically deteriorate the accuracy of the solution. On the other hand, in our limiting strategy the characteristic decomposition is computed at the spatial location where the flux is limited and used for all nodes involved in the limiting. Our single- and multi-block solutions do not contain any spurious oscillation at the border of the boundary layer (Figs. 6.8(b) and 6.8(d)).

Finally, the main outcomes of the analysis conducted in this section are:

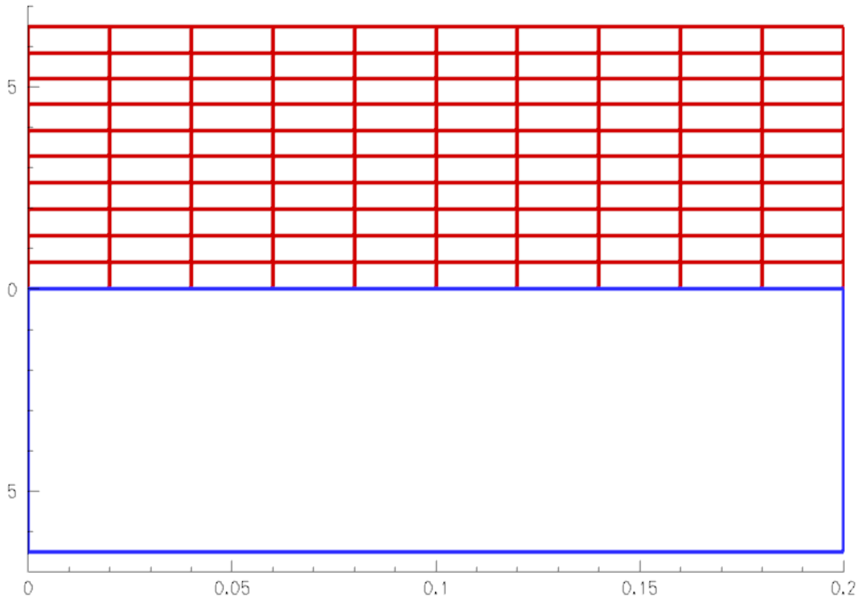
- Our modification to the characteristic treatment greatly improves the accuracy in simulating wall-bounded flows, and is absolutely essential to retain the accuracy on multi-block grids.
- Viscous corner flows pose a serious challenge to the compact-TVD method, even if our modified characteristic treatment is used. Block-partitioning seems to improve the accuracy in simulating this type of flow, because the formulas used at block-interface boundaries make the method more dissipative.

6.1.3 Shock-jet interaction

In this section we present the simulation of a planar shock wave propagating in air and interacting with a cylindrical jet of lighter fluid. This test was proposed by Don and Quillen (94) and represents a technique used to enhance the mixing between the reactants in combustion engines. The authors used the test to investigate the ability of FD ENO and spectral schemes to simulate such mixing. They ran inviscid multi-species simulations, where the cylindrical jet was made of hydrogen. We use exactly the same set-up but we do not model the presence of multiple species. Since the flow is inviscid,

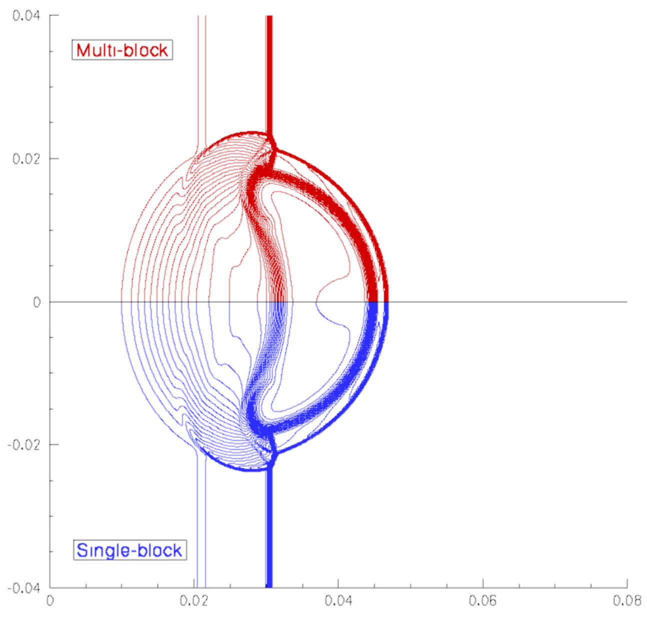


(a) Density contours at time $t = 0$.

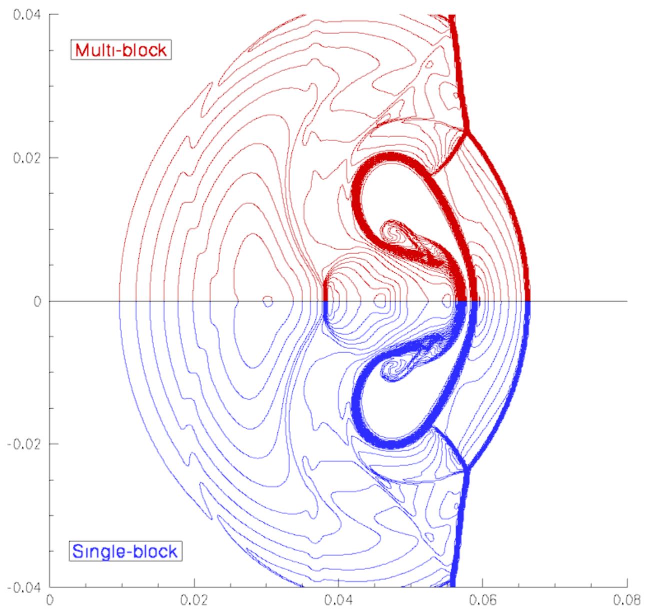


(b) Mesh partitioning.

Figure 6.9: Initial density contours and mesh partitioning for the shock-jet interaction. The case is simulated using both a multi-block (Fig. 6.9(b), upper part) and a single-block (Fig. 6.9(b),lower part) mesh. Only the block boundaries are marked as the mesh is too fine to be shown. The scale of the axes is in metres while the density is in kg/m^3.

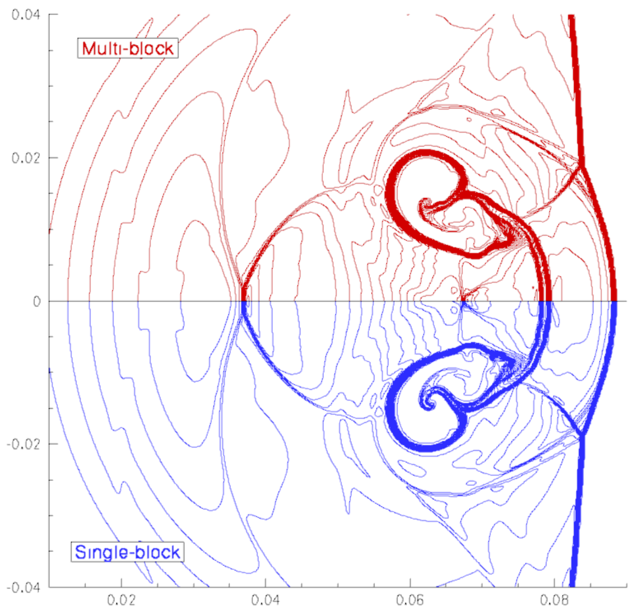


(a) Density at time $t = 20$ μs

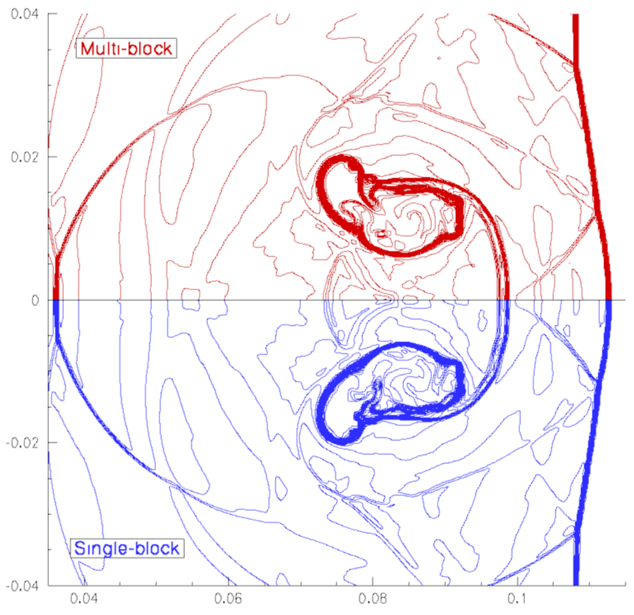


(b) Density at time $t = 40$ μs

Figure 6.10: Shock-jet interaction: density contours at indicated times, 40 equally spaced levels between 0.05 kg/m^3 and 1.2 kg/m^3. Comparison between multi-block (upper part) and single-block (lower part) solutions. The scale of the axes is in metres.



(a) Density at time $t = 60\ \mu s$



(b) Density at time $t = 80\ \mu s$

Figure 6.11: Shock-jet interaction: density contours at indicated times, 40 equally spaced levels between 0.05 kg/m^3 and 1.2 kg/m^3. Comparison between multi-block (upper part) and single-block (lower part) solutions. The scale of the axes is in metres.

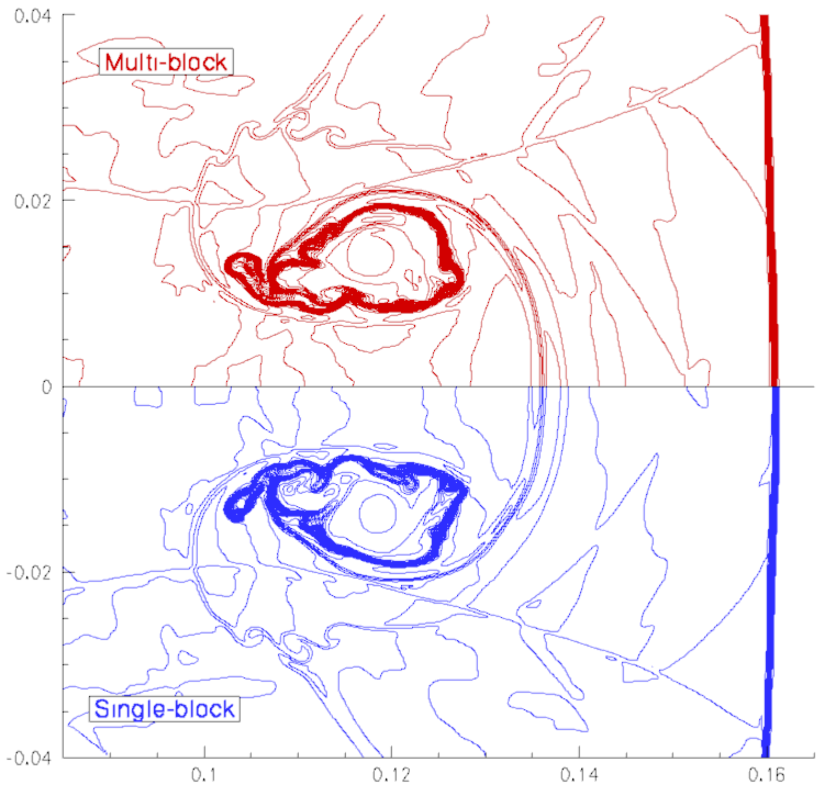
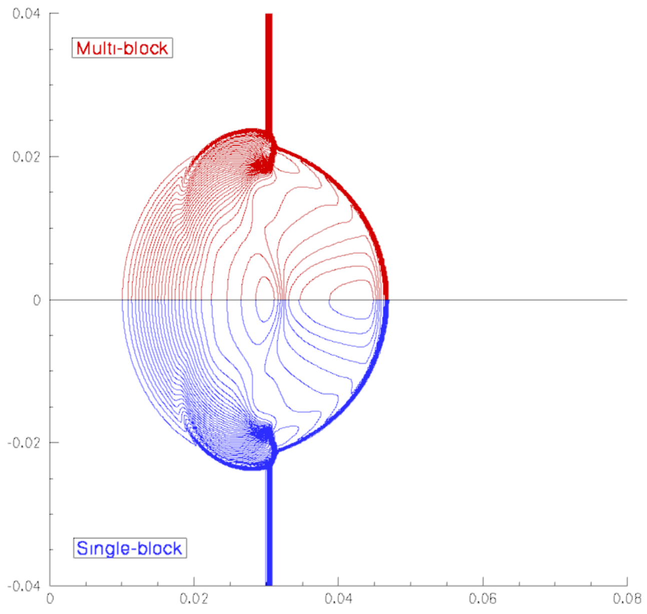


Figure 6.12: Shock-jet interaction: density contours at time $t = 120\ \mu s$, 40 equally spaced levels between 0.05 kg/m^3 and 1.2 kg/m^3. Comparison between multi-block (upper part) and single-block (lower part) solutions. The scale of the axes is in metres.

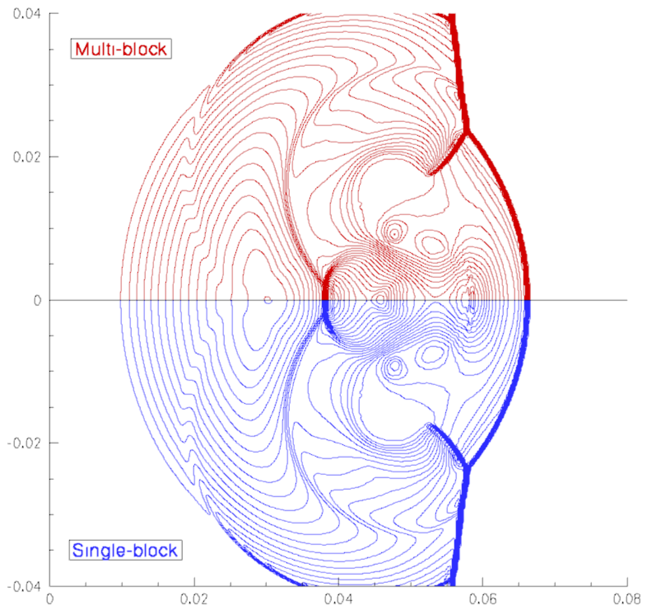
there is no species diffusion; furthermore, hydrogen and air are both diatomic gases, so the isentropic index is $\gamma = 7/5$ for both. Hence, modelling multiple species has no impact on the structure of the simulated flow. In the following we describe the test set-up as in Don and Quillen, bearing in mind that in our simulations the hydrogen jet is simply a zone of less dense fluid.

Initial density contours for this test case at time $t = 0$ are shown in Fig. 6.9(a). The planar shock is located at $x_s = 0.05$ cm at time $t = 0$ and is moving downstream at Mach $M_s = 2$. The hydrogen jet has its centre at $(x_{H_2}, y_{H_2}) = (2.5\text{ cm}, 0\text{ cm})$ and a radius $r_{H_2} = 2$ cm. Initially it has a diffused boundary with the surrounding air, i.e. the conserved variables at the interface are modified according to the factor:

$$s(r) = \exp\left[-\alpha\left(\frac{r}{r_{H_2}}\right)^{\delta}\right],$$

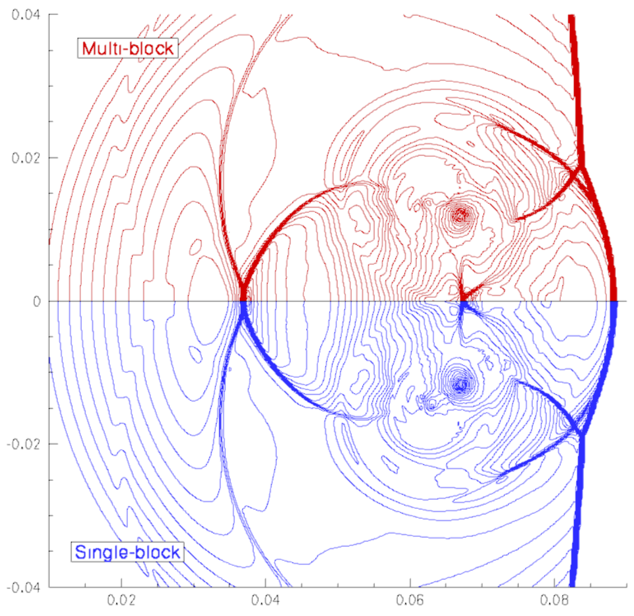


(a) Pressure at time $t = 20\ \mu s$

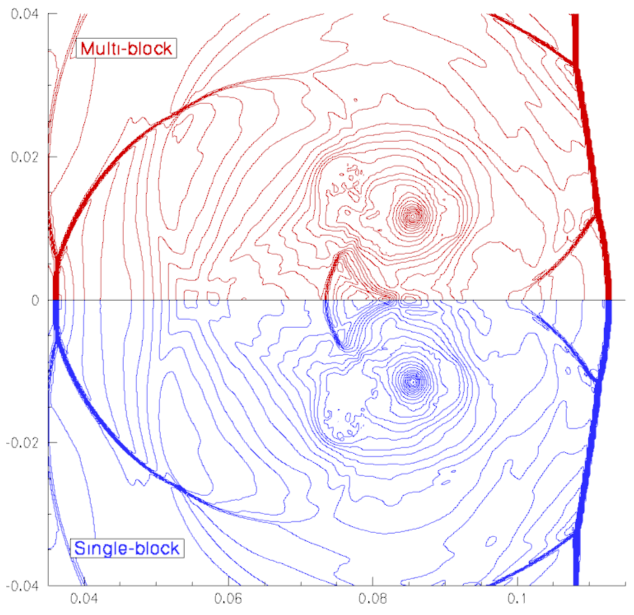


(b) Pressure at time $t = 40\ \mu s$

Figure 6.13: Shock-jet interaction: pressure contours at indicated times, 60 equally spaced levels between 110 kPa and 700 kPa. Comparison between multi-block (upper part) and single-block (lower part) solutions. The scale of the axes is in metres.



(a) Pressure at time $t = 60\ \mu s$



(b) Pressure at time $t = 80\ \mu s$

Figure 6.14: Shock-jet interaction: pressure contours at indicated times, 60 equally spaced levels between 110 kPa and 700 kPa. Comparison between multi-block (upper part) and single-block (lower part) solutions. The scale of the axes is in metres.

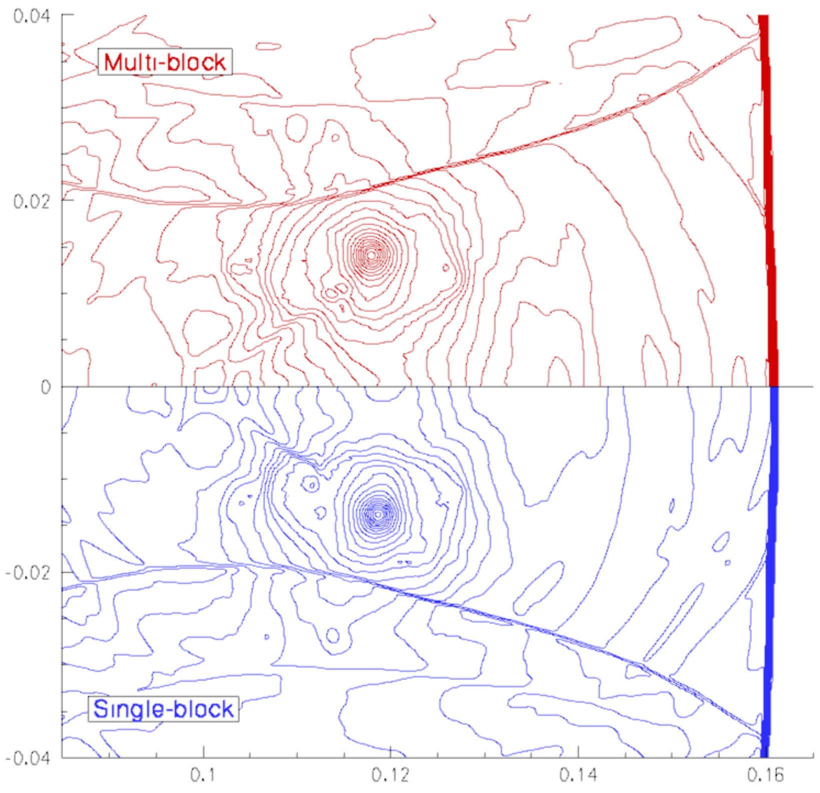
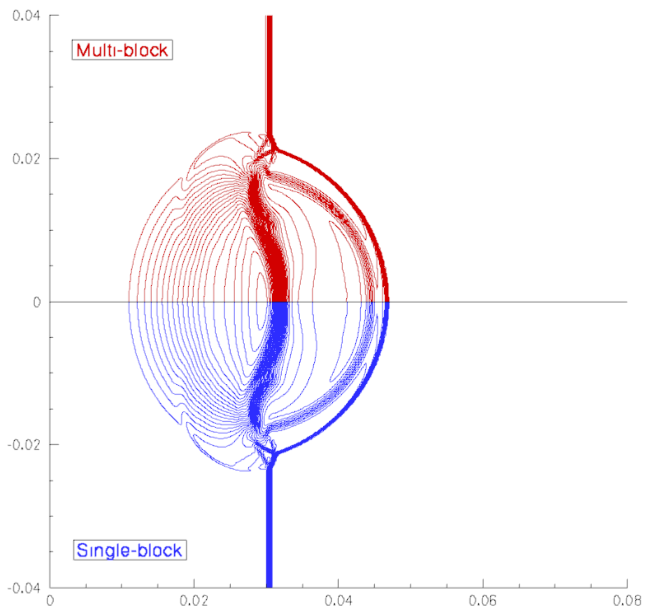
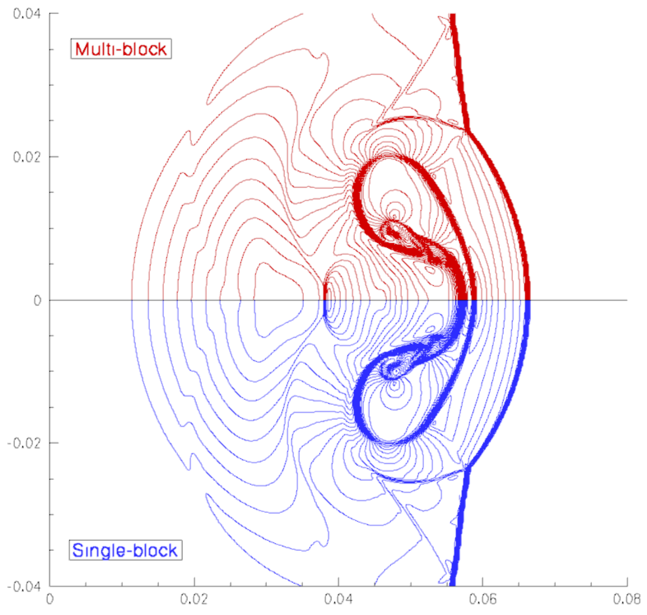


Figure 6.15: Shock-jet interaction: pressure contours at time $t = 120\ \mu s$, 60 equally spaced levels between 110 kPa and 700 kPa. Comparison between multi-block (upper part) and single-block (lower part) solutions. The scale of the axes is in metres.

where r is the radial distance from the centre of the jet, $\alpha = -\ln \epsilon$, with ϵ being the machine zero, and $\delta = 16$. The undisturbed region ahead of the shock has pressure $p_\infty = 1$ atm and temperature $T_\infty = 1000$ K. The density is determined by the perfect gas equation with gas constant $R_{air} = 286.9$ J/kg K. The jet is in equilibrium with the surrounding air, so the hydrogen-to-air density ratio is determined by the the ratio of the molecular weights $\rho_{H_2}/\rho_{air} = MW_{H_2}/MW_{air} = 4.003/28.97$. The flow peoperties behind the shock wave are determined via the normal shock wave relations, given the shock Mach number $M_s = 2$. The case is symmetric around the x-axis, so the computational domain comprises just half of the jet and is $(x, y) \in [0\text{ cm}, 20\text{ cm}] \times [0\text{ cm}, 6.5\text{ cm}]$. The computational grid consists of 752×376 equally spaced cells. The simulation is run both on a single- and on a multi-block grid. The multi-block grid is partitioned in both directions and consists of 10×10 blocks, as shown in Fig. 6.9(b).

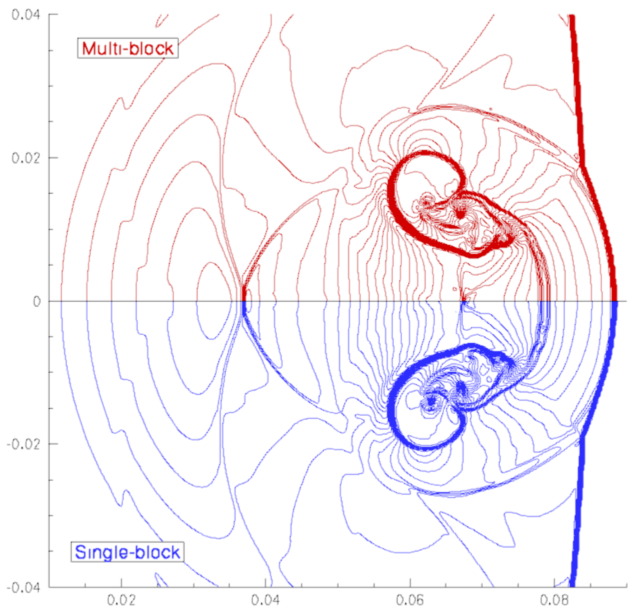


(a) Mach number at time $t = 20\ \mu s$

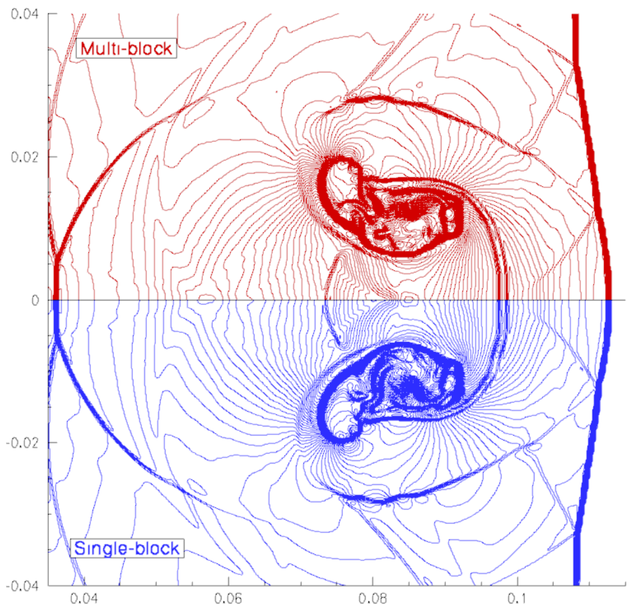


(b) Mach number at time $t = 40\ \mu s$

Figure 6.16: Shock-jet interaction: Mach number contours at indicated times, 40 equally spaced levels between 0.05 and 1.9. Comparison between multi-block (upper part) and single-block (lower part) solutions. The scale of the axes is in metres.



(a) Mach number at time $t = 60$ μs



(b) Mach number at time $t = 80$ μs

Figure 6.17: Shock-jet interaction: Mach number contours at indicated times, 40 equally spaced levels between 0.05 and 1.9. Comparison between multi-block (upper part) and single-block (lower part) solutions. The scale of the axes is in metres.

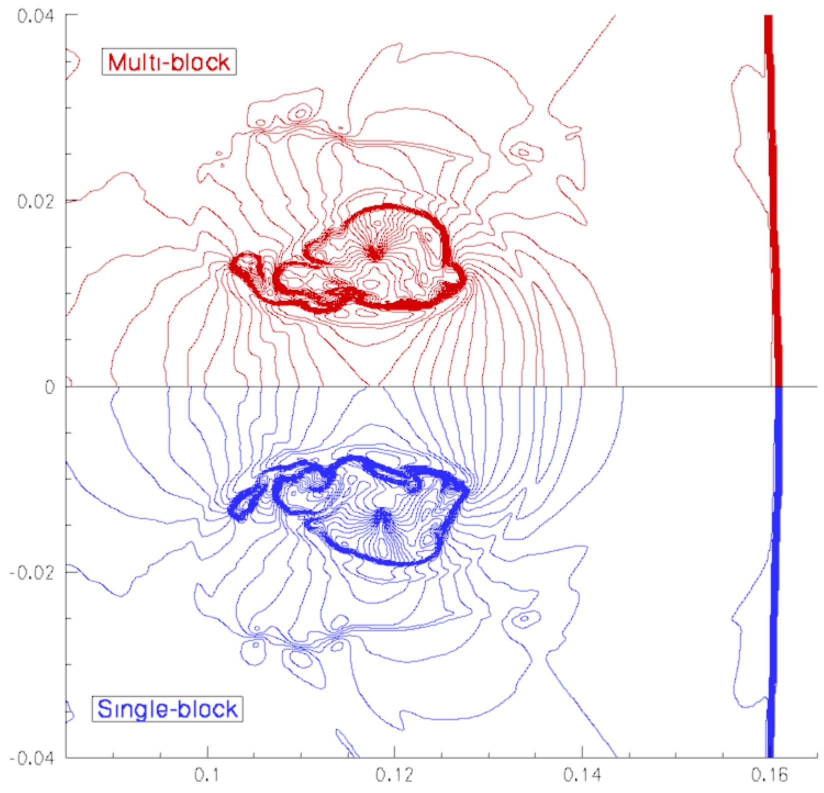


Figure 6.18: Shock-jet interaction: Mach number contours at time $t = 120\ \mu s$, 40 equally spaced levels between 0.05 and 1.9. Comparison between multi-block (upper part) and single-block (lower part) solutions. The scale of the axes is in metres.

Upper and lower boundaries are modelled as slip-walls, the left boundary as a subsonic inflow and the right boundary as a subsonic outflow.

Figs. 6.10–6.18 show the interaction from 20 μs to 120 μs in terms of density, pressure and Mach number. In these figures, the upper part is the multi-block solution, and the bottom part is obtained mirroring the single-block solution around the symmetry plane. The shock passes through the hydrogen jet and compresses it. The jet is a region with different speed of sound ($a_{H_2}/a_{air} = \sqrt{\rho_{air}/\rho_{H_2}}$) and so the shock is diffracted. A double Mach-reflection forms at the top of the jet (Fig. 6.10(a)). The hydrogen jet is convected downstream and at time $t = 40\ \mu$s (Figs. 6.10(b)) a narrow jet of air begins to penetrate it and rolls up forming a vortex. At time $t = 60\ \mu s$ (Fig. 6.11(a)) a second jet of air penetrates the hydrogen jet. The *mushroom*-type shape of this air jet reveals the nature of this mixing, related to the Rayleigh-Taylor (RT) instability, discussed in section 5.1.3. In this case the instability is triggered by the sudden acceleration of the hydrogen-air interface due to the propagating shock. The hydrogen jet is broken into two contra-rotating jets (Fig. 6.11(b)). The pressure gradient (Fig. 6.14(b)) associated with the rotation of these two jets favours the formation of more RT fingers. At time $t = 120$ μs (Fig. 6.12) the flow has a very complex structure. The jet is broken in two parts and its interface is deformed by many RT fingers; a contact rolls up (Kelvin-Helmoltz instability) and a shock, originated at the beginning of the interaction and reflected at the domain boundary, is reaching the jet.

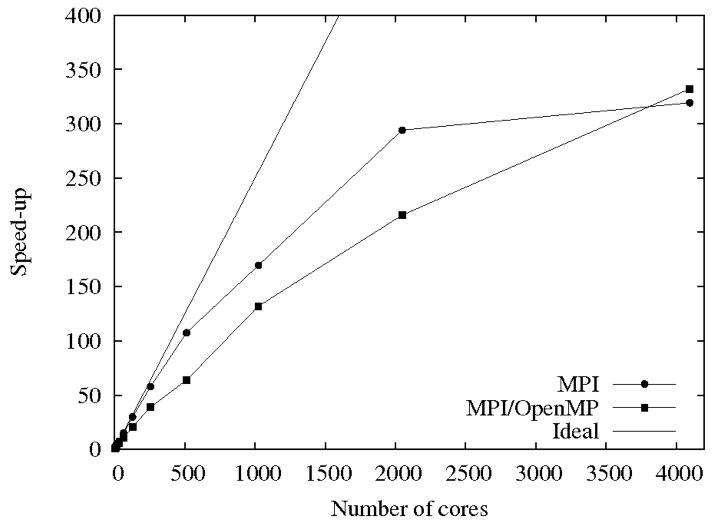
Despite the complexity of the flow structure, the agreement between single- and multi-block solutions is excellent. Up to time $t = 60$ μs the solutions are virtually identical. At time $t = 80$ μs some minor differences start to appear: the single-block solver predicts one long RT finger entering the jet, while the multi-block solver predicts two shorter fingers (Figs. 6.11(b), 6.17(b)). At time $t = 120$ μs the differences are more evident, though still minor: the single-block solver predicts one more finger entering the jet, and all fingers are generally longer than in the multi-block solution (Figs. 6.12, 6.18); the shocklet close to the symmetry plane is not sharply captured in the multi-block solution (Fig. 6.15).

Finally, we can conclude that in this case the computed results are not significantly affected by the block-partitioning.

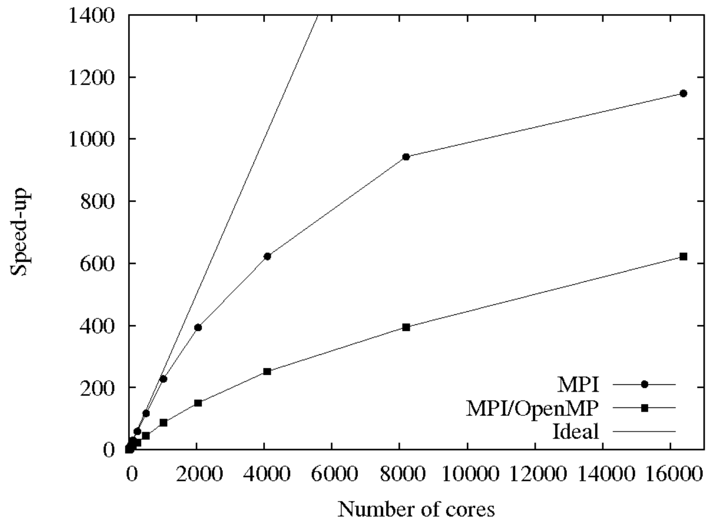
6.1.4 Parallel performance

We have made an assessment of the parallel performance of the multi-block compact-TVD method for the shock-jet interaction problem. The simulations have been run on the Cray XT4 facility of the UK National Supercomputing Service, HECToR (`www.hector.ac.uk`). The XT4 system comprises 1,416 compute blades, each of which has 4 quad-core processor sockets. This amounts to a total of 22,656 cores, each of which acts as a single CPU. The processor is an AMD 2.3 GHz Opteron. The point-to-point bandwidth is 2.17 GB/s and the latency between two processors is around 6 μs.

Strong scalability tests have been run for different problem sizes. The strong scalability represents the ability to speed-up a certain computation by using more processors. Hence, in a strong scalability test a fixed work-load is shared by an increasing number of processors, and the variation in the compute time is tracked. For a CFD application, the work-load is represented by the grid size. A coarse grid of 752×376 ($\approx 3 \times 10^5$)



(a) 3×10^5 cells



(b) 10^6 cells

Figure 6.19: Strong scalability tests. Speed-ups for two computational grids of 3×10^5 and 10^6 cells are shown.

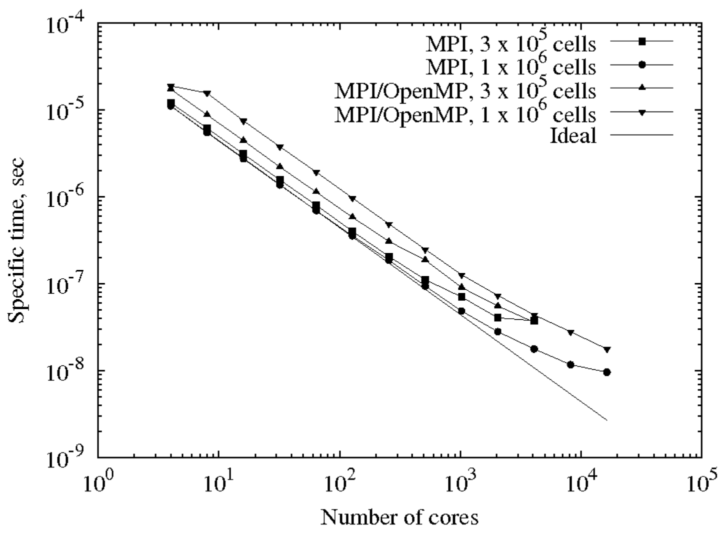


Figure 6.20: Strong scalability tests. Specific times for two computational grids of 3×10^5 abd 10^6 cells are shown.

cells and a fine grid of 1504×752 ($\approx 10^6$) cells have been considered. The simulations have been run on an increasing number of cores, starting with 4 and doubling the number at each stage. For a fixed number of cores and grid resolution, each simulation has been run first in pure MPI mode and then in hybrid MPI/OpenMP mode. In pure MPI mode, the domain was partitioned into as many blocks as cores. In hybrid MPI/OpenMP mode, the domain was partitioned into as many blocks as quad-core processors, and each block was processed in parallel by the four cores belonging to the processor, each one assigned with an OpenMP thread.

Fig. 6.19 shows the results of the strong scalability tests in terms of *speed-up*, defined as the ratio between the time taken to run the simulation on a certain number of cores and the time taken to run the same simulation on one quad-core processor:

$$\text{Speed-up}(n) = \frac{\text{Time taken on 4 cores}}{\text{Time taken on } n \text{ cores}}.$$

The speed-up is a dimensionless parameter and ideally varies linearly with the number of cores. Therefore, it represents an immediate way of looking at scalability properties. On the other hand, the speed-up usually improves as the work-load increases (as explained later in this section), and so it is misleading to compare speed-up curves for different problem sizes. For this reason, in Fig. 6.20 we show the same results in terms of *specific*

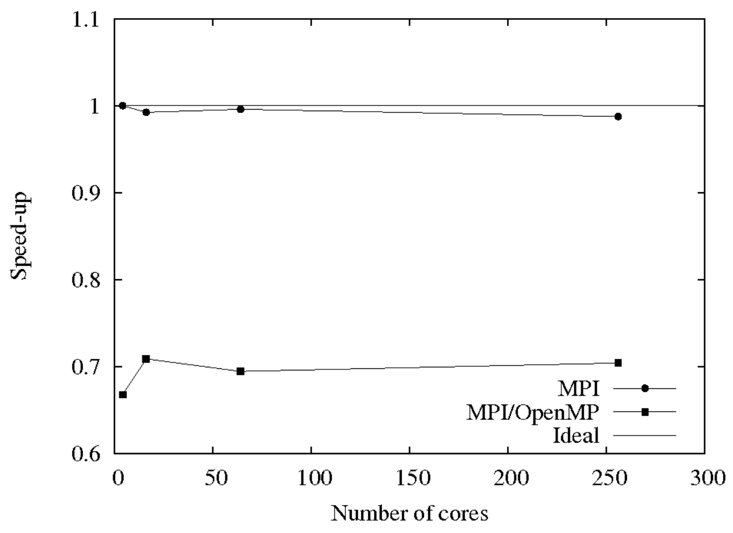


Figure 6.21: Weak scalability test. The shock-jet interaction is simulated keeping a constant load of about 3×10^5 cells per quad-core processor.

time, defined as the time taken to simulate one time step, divided by the number of cells:

$$\text{Specific time} = \frac{\text{Time to compute one time step}}{\text{Number of cells}}.$$

The specific time allows direct comparison between the scalability for different problem sizes, and also gives information on the serial performance of the code. Speed-ups and specific times are averaged over 100 time steps and do not include the time spent in input/output (I/O) operations.

A parallel efficiency can be defined as the ratio of the speed-up to the ideal speed-up, or, equivalently, as the ratio of the specific time to the ideal specific time. In pure MPI mode, efficiencies of about 90% are achieved on 256 cores for the coarse grid and on 1024 cores for the fine grid, which is an excellent result. As the number of cores increases the work-load per core decreases, the time spent in core-to-core communication becomes comparable to the time each core spends in computation and eventually begins to dominate it. This causes the speed-up curve to reach a maximum: for higher core counts the communication time is higher than the computing time and so the runtime increases. For the coarse grid this happens at about 4000 cores (Fig. 6.19(a)), while for the fine grid the code is still scaling at 16000 cores (Fig. 6.19(b)). The compute time increases about linearly with the problem size (as all cells are involved in the

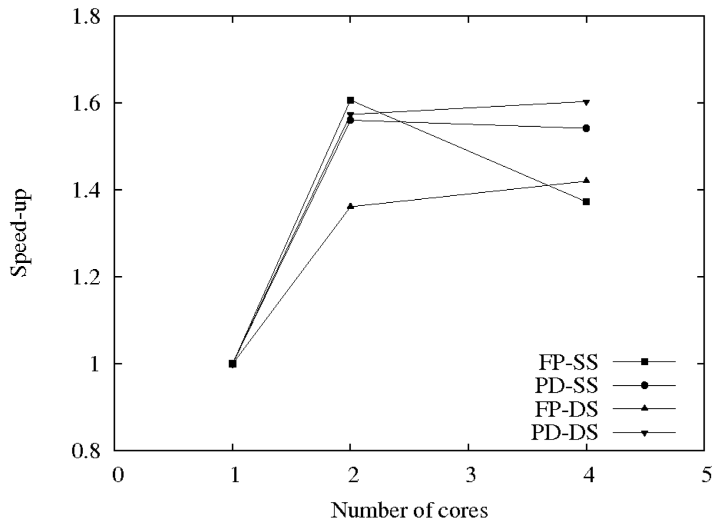


Figure 6.22: Scalability test for the OpenMP implementation of the slab decomposition algorithm. The shock-jet interaction is simulated over 3×10^5 cells using a quad-core processor.

computation), while the communication time increases as the square root of the problem size in 2D (as boundary cells only are involved in the communication); therefore, the larger the problem size the higher the core count for which the communication time becomes comparable to the compute time.

Fig. 6.21 shows the results of a *weak scalability* test in terms of speed-up. The weak scalability represents the ability to tackle larger problems by sharing the higher work-load between more processors. Therefore, in a weak scalability test the work-load increases with the core count, in such a way that the work-load per core remains constant. Ideally, larger problems are solved in the same time, if the work-load per core is kept constant, and so the ideal speed-up is equal to 1. Hence, in a weak scalability test there is no difference between speed-up and parallel efficiency.

We have considered a fixed work-load of about 3×10^5 cells per quad-core processor and used up to 256 cores (about 2×10^7 cells). We could not consider higher core counts because our code has a serial I/O and the memory requirements for problems larger than 2×10^7 cells exceed the memory available to a single processor. In pure MPI mode, the parallel efficiency is about 98% on 256 cores and the speed-up curve is close to horizontal.

In all scalability tests, the parallel efficiency in hybrid MPI/OpenMP mode is about 70% of that in pure MPI mode if the same number of cores is employed. This difference in performance results from two concurrent factors:

1. **The communication between processors.**
   In hybrid mode the communication is more efficient, as the cores do not compete for the same interconnector and each processor is assigned one structured-block only. This results in fewer bigger messages to be sent, and hence higher inter-processor bandwidth.

2. **The scalability over one quad-core processor.**
   The distributed-memory algorithm requires intra-processor communication, unlike the shared-memory one. This is clearly a disadvantage, but it is mitigated by the architecture-aware MPI implementation, able to perform intra-processor communication faster than inter-processor communication. On the other hand, the scalability of the shared-memory kernel is affected by several factors:

   (a) The memory access properties of the processor (NUMA or SMP). This issue has been discussed in section 4.1.3 and its effects are analysed later in this section.

   (b) The algorithm is parallel only in the most computationally expensive part: the computation of the flux divergence. Slave threads lie idle during some serial computations performed by the master thread and during communication.

   (c) Thread synchronization is required for each processing dimension, while in the distributed-memory algorithm, processes are synchronized just once for all dimensions.

The advantage of the hybrid mode in communication is evident at very high core counts. In the strong scalability tests (Figs. 6.19 and 6.20), at high core counts the communication time in pure MPI mode dominates the computing time, and better efficiency is achieved in hybrid mode because less communication is required. This is clearly shown for the coarse mesh (Fig. 6.19(a)), while for the fine mesh it would likely happen at a core count higher than the maximum number of cores available, as can be argued from Fig. 6.20. In the weak scalability test (Fig. 6.21), besides the lower

Shared-memory algorithm options	Specific time, μs		
	1 thread	2 threads	4 threads
PD-SS	22.6	14.5	14.7
PD-DS	22.6	14.4	14.1
FP-SS	22.7	14.1	16.5
FP-DS	22.7	16.7	16.0

Table 6.2: Specific runtimes of multi-thread CPU NS solver. The shock-jet interaction is simulated over 3×10^5 cells using a quad-core processor.

efficiency, the speed-up curve is close to horizontal, denoting efficient inter-processor communication.

In the following, we study the scalability of the shared-memory algorithm only. We simulate the shock-jet interaction on the coarse grid using one quad-core processor and an increasing number of threads. Different code configurations are considered.

In PD (Parallel Divergence) configuration, the flux divergence is computed in parallel; in FP (Full Parallel) configuration, boundary condition enforcement, variables initialisation and variables update are also performed in parallel.

Furthermore, since the OpenMP parallelism is based on loop unrolling, we consider different scheduling strategies, namely *static* and *dynamic*. In static scheduling, the slabs are equally divided between the available threads. In dynamic scheduling, the slabs are assigned to the threads as they become available. There is an overhead associated with dynamic scheduling, as the master thread checks at runtime which threads are available. We use the acronyms SS and DS referring to static and dynamic scheduling, respectively.

Specific runtimes for the shared-memory kernel are shown in Table 6.2, and corresponding speed-ups plotted in Fig. 6.22. Passing from one to two cores there is about 60% gain in speed for FP-SS, PD-SS and PD-DS configurations, but only 37% for the FP-DS configuration. In this configuration, the overhead of dynamic scheduling overwhelms the little speed-up (note the difference between FP-SS and PD-SS speed-ups for two cores) gained from parallelising the light tasks. Passing from two to four cores, the runtime increases using static scheduling, while there is a small improvement using dynamic scheduling. This clearly reveals how the memory access properties of the processor dramatically affect the performance.

The AMD Opteron has a Non-Uniform Memory Access (NUMA) architecture. As explained in section 4.1.3, in NUMA processors memory sharing is implemented at software level, but the memory is physically distributed (Fig. 4.4). All cores try to access the same memory bank and compete for the same memory controller. Hence, they are forced to lie idle while waiting to fetch data from the main memory, and cannot work effectively in parallel. Furthermore, slave cores access the memory of the master core with lower bandwith, and the overall runtime is higher than for two cores. This is particularly evident for the FP-SS configuration, where the light tasks are dramatically slowed down by the reduced memory bandwidth. The dynamic scheduler assigns more tasks to the fast cores, i.e. those with higher memory bandwidth, and so it is able to slighly improve the runtime using four cores.

Poor scalability on NUMA processors is an issue common to global and semi-global methods (95). For these methods data locality is difficult to enforce because either the solution at one node depends on all nodes within the domain (global methods), or its domain of dependency changes with the processing direction (semi-global methods). The performance of the shared-memory kernel can be improved by explicitly moving data between the cores (95). We have not implemented this modification for two reasons:

- Moving data between cores essentially means performing a distributed-memory transpose, as in the distributed-memory slab/drawer decomposition (34). Since the algorithm becomes as complex as in a distributed-memory environment, and most MPI implementations are architecture-aware, the use of a shared-memory programming model is not justified as it does not significantly affect the performance (74).
- If an SMP processor is used, the agorithm scales excellently without distributed-memory transposition, as shown in the results presented in section 6.2.3.

6.2 Application of the GPU NS solver

In this section we present validation results and performance assessment of our GPU Navier-Stokes solver, CUDA-CU5-TVD-NS.

Many standard tests, such as those presented in Chapter 5, have been performed to validate CUDA-CU5-TVD. Computed results are analogous to those shown in Chapter 5, and so are not repeated here. In this section we present two new simulations, with a dual purpose: first, we add new information on the capability of the numerical method; second, we show that the reduced precision (in CUDA-CU5-TVD-NS floating-point operations are performed in single-precision, while double-precision is employed in CU5-TVD-NSK) does not impact the results either in time-accurate simulations (section 6.2.1) nor in steady-state calculations (section 6.2.2).

The first test is the time-accurate simulation of vortex pairing in a compressible mixing layer (96). For this test, presented in section 6.2.1, we validate our results against numerical data obtained by Sandham and Yee (96).

The second test case is hypersonic flow over a compression ramp. In this simulation the shock-boundary layer interaction results in a large region of separated flow, whose extent is strongly affected by the finite span of the ramp, as shown by Rudy *et al.* (97). In section 6.2.2 our results are validated against experimental data (98), showing CUDA-CU5-TVD-NS's capability to simulate 3D hypersonic flows.

Finally, in section 6.2.3 we assess the performance gain of the GPU implementation over a multi-thread CPU implementation.

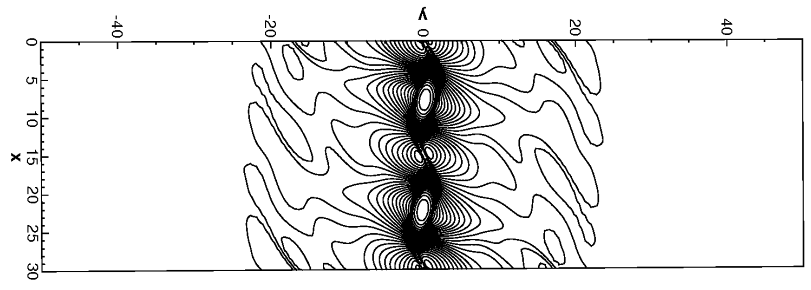
6.2.1 Vortex pairing in a compressible mixing layer

The simulation of vortex pairing in a mixing layer is a classic test case for high-order numerical methods. Two viscous streams at different velocities flow parallel to each other; since the streams are viscous, a layer forms between the two streams, where the fluid dynamic variables smoothly pass from one free-stream condition to the other. Such a region is called a mixing layer and is subject to the Kelvin-Helmoltz instability.

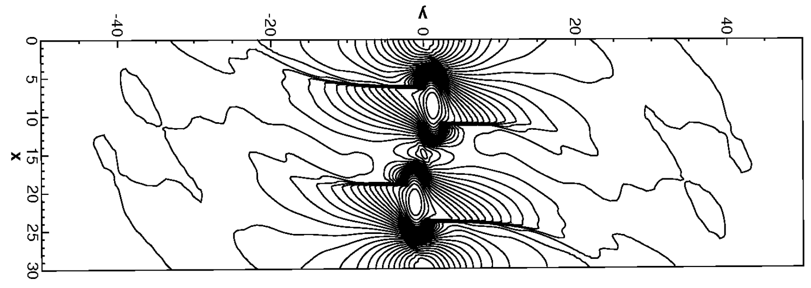
Numerically, the mixing layer evolution is studied either spatially or temporally. In a spatial mixing layer, a suitably big domain encloses the two streams and the instability is studied as it develops both in time and space along the domain. In a temporal mixing layer, the problem is periodic along the direction parallel to the streams, and the development of the instability is studied in time only.

We simulate a temporal mixing layer with *convective Mach number*:

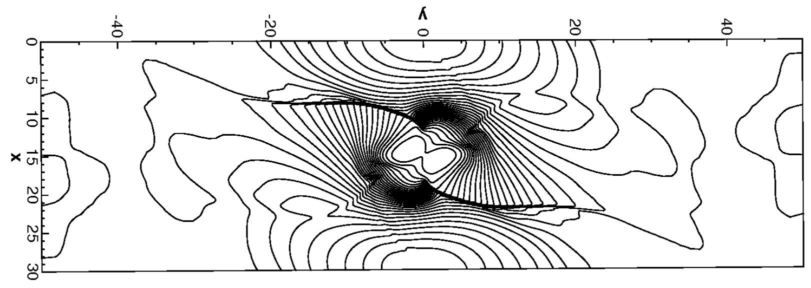
$$M_c = \frac{U_1 - U_2}{a_1 + a_2} = 0.8,$$



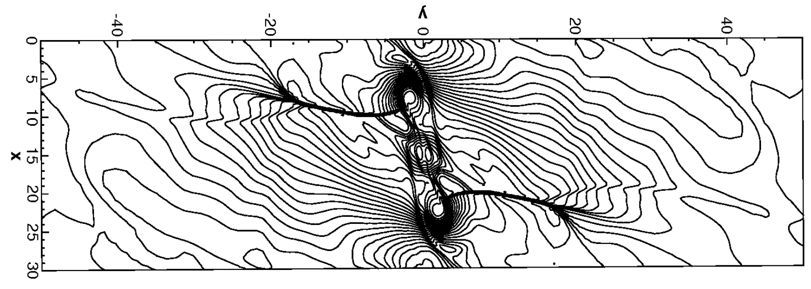
(a) $t = 40$



(b) $t = 80$

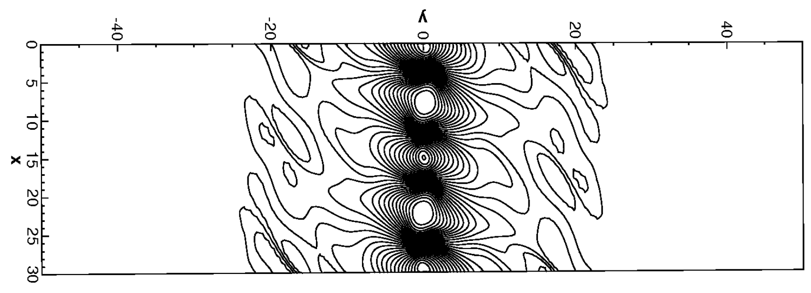


(c) $t = 120$

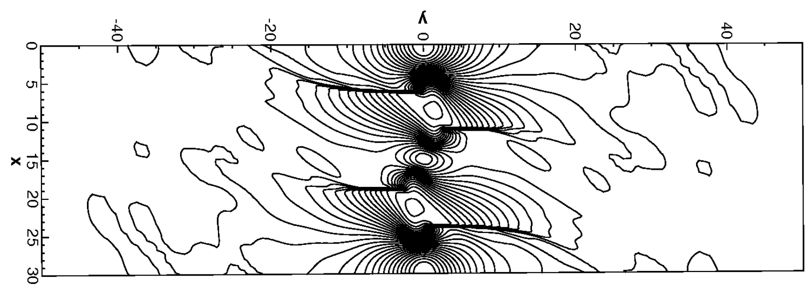


(d) $t = 160$

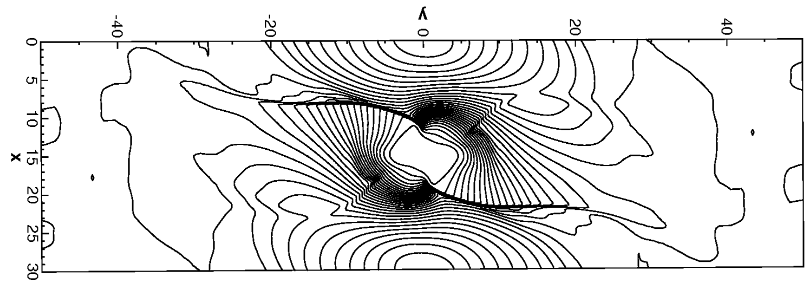
Figure 6.23: Compressible mixing layer, density contours at indicated times: 30 equally spaced levels between 0.35 and 1.35.



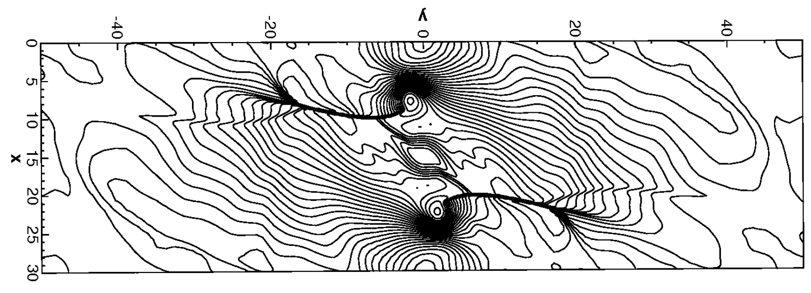
(a) $t = 40$



(b) $t = 80$

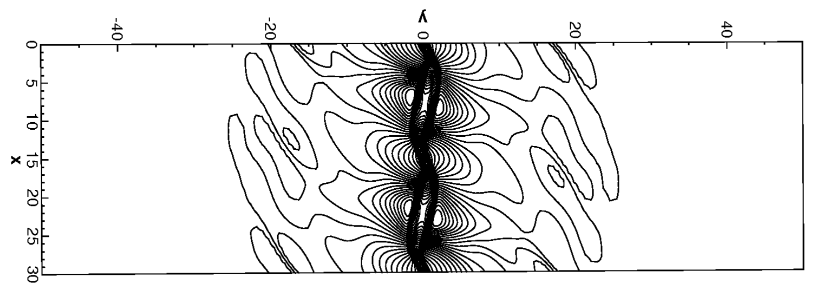


(c) $t = 120$

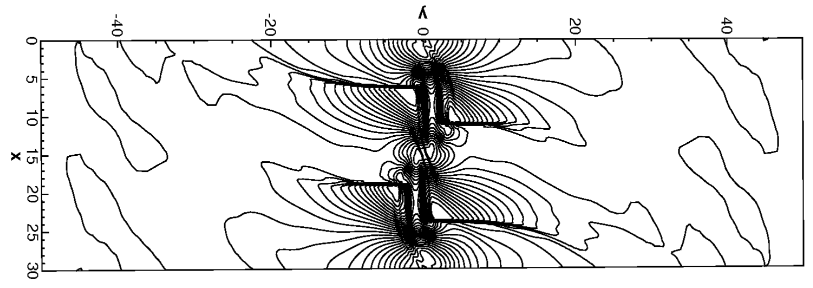


(d) $t = 160$

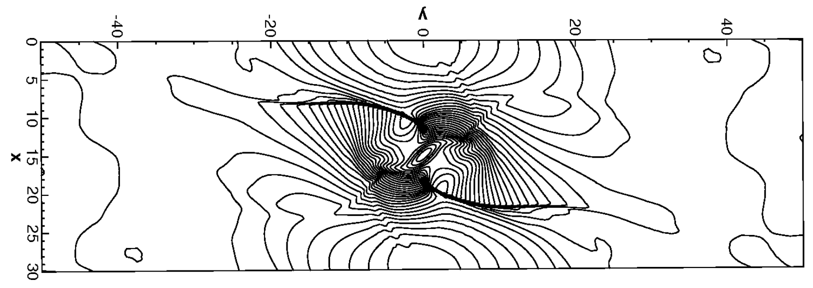
Figure 6.24: Compressible mixing layer, pressure contours at indicated times: 30 equally spaced levels between 0.08 and 0.4.



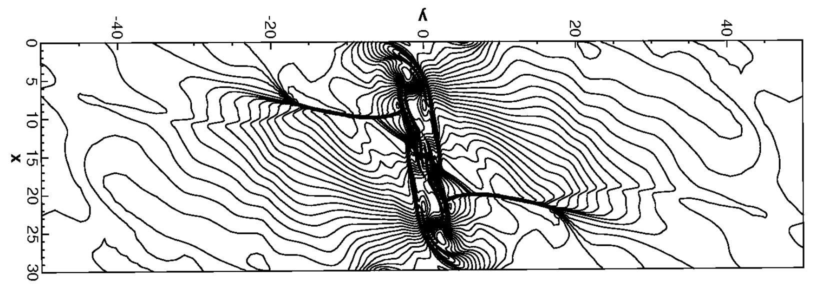
(a) $t = 40$



(b) $t = 80$

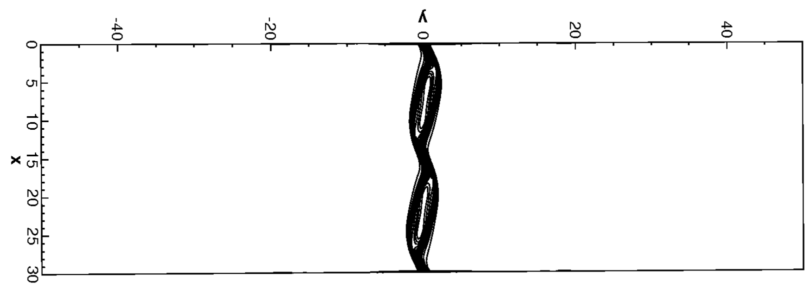


(c) $t = 120$

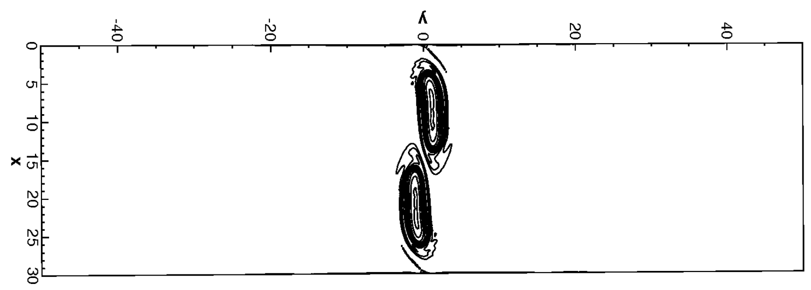


(d) $t = 160$

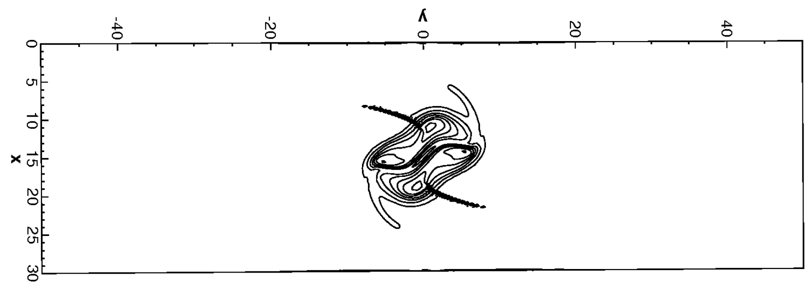
Figure 6.25: Compressible mixing layer, temperature contours at indicated times: 30 equally psaced levels between 0.72 and 1.18.



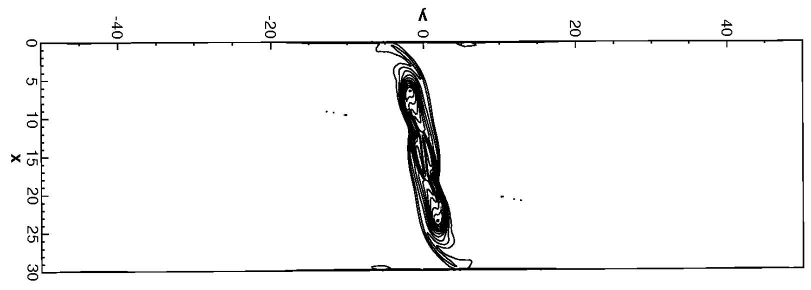
(a) $t = 40$



(b) $t = 80$

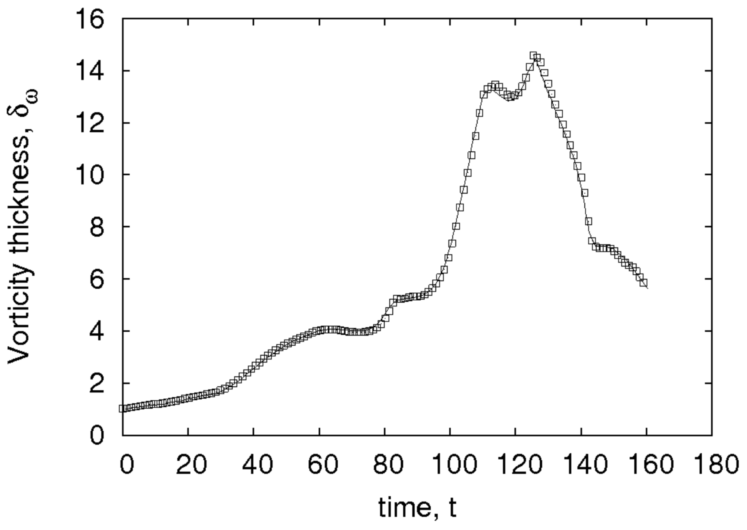


(c) $t = 120$

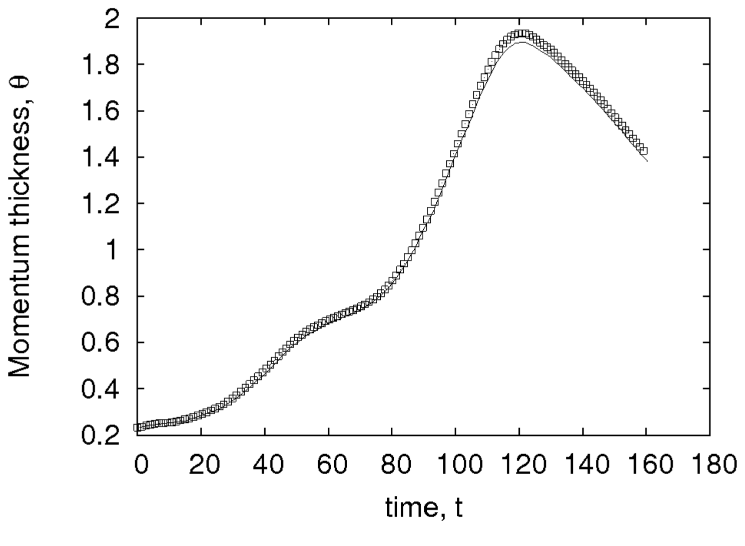


(d) $t = 160$

Figure 6.26: Compressible mixing layer, vorticity contours at indicated times: 55 equally spaced levels between 0.05 and 2.75.



(a) Vorticity thickness, δ_ω



(b) Momentum thickness, θ

Figure 6.27: Vorticity (a) and momentum (b) thickness time-evolutions: —, ACM44 method (96); □, CUDA-CU5-TVD-NS.

where subscripts 1 and 2 denote upper and lower streams, respectively, U the velocity and a the speed of sound. The set-up is the same as in Sandham and Yee (96). The velocity components are normalised with respect to the velocity difference between upper and lower stream, $V_{ref} = U_1 - U_2$, and $U_2 = -U_1$. The base velocity field is:

$$u(x, y) = 0.5 \tanh(2y),$$

for which $u(x, y \to \pm\infty) = \pm 0.5$. Lengths are normalised with respect to the *vorticity thickness*:

$$\delta_\omega = \frac{U_1 - U_2}{\max\left|\frac{d}{dy}\left(\frac{\bar{\rho u}}{\bar{\rho}}\right)\right|}, \tag{6.1}$$

computed at time $t = 0$. In Eq. 6.1 an over-bar denotes averaging over the x-direction:

$$\bar{f}(y) = \frac{1}{L_x}\int_0^{L_x} f(x, y)dx,$$

where either $f = \rho u$ or $f = \rho$, and $L_x = 30$ is the domain size in the x-direction. Temperature and density are normalised with respect to their free-stream values, $T_1 = T_2$ and $\rho_1 = \rho_2$, respectively, and the pressure with respect to $\rho_1(U_1 - U_2)^2$. The pressure is uniform throughout the flow field, while the temperature varies so that the total enthalpy remains constant:

$$T(x, y) = T_1 + \frac{1}{2c_p}\left(u(x, y)^2 - u_1^2\right).$$

The reference temperature is $T_{ref} = 300\ K$, and the reference Mach and Reynolds numbers are, respectively:

$$M = \frac{U_1 - U_2}{a_1} = 1.6, \qquad Re = \frac{\rho_1(U_1 - U_2)\delta_\omega(t = 0)}{\mu_1} = 1000.$$

The instability is triggered by adding a perturbation:

$$\begin{aligned} u'(x, y) &= -\frac{2y}{b}\exp\left(-\frac{y^2}{b}\right)\frac{L_x}{2\pi}\sum_{k=1}^{2}\frac{a_k}{k}\sin\left(\frac{2\pi kx}{L_x} + \phi_k\right), \\ v'(x, y) &= \exp\left(-\frac{y^2}{b}\right)\sum_{k=1}^{2} a_k\cos\left(\frac{2\pi kx}{L_x} + \phi_k\right), \end{aligned}$$

to the base velocity field. The perturbation is divergence-free and has parameters:

$$a_1 = 0.01, \quad a_2 = 0.05, \quad \phi_1 = \phi_2 = -\pi/2, \quad b = 10.$$

The domain is $(x, y) \in [0, L_x] \times [-L_y/2, L_y/2]$, with $L_x = 30$ and $L_y = 100$. The grid consists of 192×192 nodes, equally spaced in the x-direction and stretched in the y-direction. The stretching is obtained mapping a grid, uniform in the transformed coordinate $-1 \leq \eta \leq 1$, onto the physical coordinate y, according to:

$$y = \frac{L_y}{2} \frac{\sinh(b_y \eta)}{\sinh b_y},$$

with stretching factor $b_y = 3.4$. The problem is periodic in the x-direction, while y-constant boundaries are modelled as slip-walls. The evolution is computed up to time $t = 160$, and the results are shown in Figs. 6.23–6.26 in terms of density, pressure, temperature and vorticity. Two vortices form (Figs. 6.23(a), 6.24(a), 6.25(a), 6.26(a)) and start to rotate clockwise causing the formation of shocklets in the two transonic streams (Figs. 6.23(b), 6.24(b), 6.25(b), 6.26(b)). During the rotation the vortices come close to each other and nearly merge (Figs. 6.23(c), 6.24(c), 6.25(c), 6.26(c)). Finally, while completing a rotation of 360°, they also move apart (Figs. 6.23(d), 6.24(d), 6.25(d), 6.26(d)).

The computation of the pairing process is very sensitive to the dissipation properties of the numerical method. Therefore, the compressibility poses a significant challenge as numerical diffusion is necessary to capture the shocks (2). Our computed evolution, shown in Figs. 6.23–6.26, agrees with the one computed by Sandham and Yee (96) using the ACM44 method (fourth-order explicit central FD method + artificial compressibility + second-order TVD filter). A quantitative comparison is shown in Fig. 6.27. Two measures of the shear-layer thickness, namely the vorticity thickness, Eq. 6.1, and the *momentum thickness*:

$$\theta = \int_{-L_y/2}^{L_y/2} \bar{\rho}(U_1 - \bar{u})(\bar{u} - U_2) dy,$$

are shown. Both thicknesses experience an initial growth followed by a short flat region. Such behaviour corresponds to growth and saturation of the instability responsible for the formation of the two vortices. A second growth in the thicknesses is observed, and is associated with the growth of the instability responsible for the pairing motion. Finally, viscous effects cause the vortex motion to end, and the shear-layer to thin.

Our results are in excellent agreement with those of Sandham and Yee. The evolution of the vorticity thickness is followed in every detail, while we compute a slightly higher peak in the momentum thickness. We have also computed the evolution using

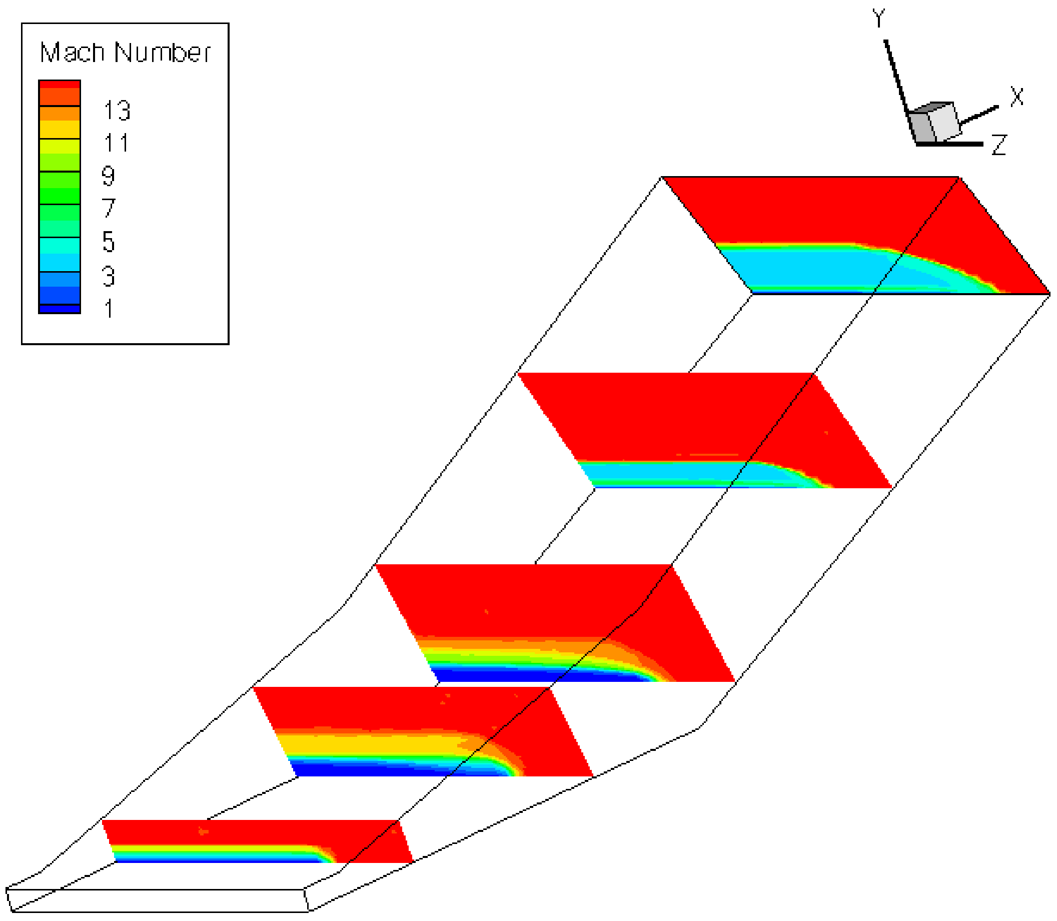


Figure 6.28: Hypersonic flow over a compression ramp: Mach number contour plots at $y - z$ planes.

the double-precison CPU code; the results are virtually identical to those in single-precision and so are not shown.

6.2.2 Hypersonic compression ramp

The hypersonic compression ramp is a classic example of a viscous/inviscid interaction, like the shock/boundary layer interaction over a flat plate discussed in section 5.3. A stream-wise positive pressure gradient forms in the boundary layer, caused by the impingement of an oblique shock for the flat-plate case, and by a sudden inclination of the wall for the compression-ramp. The physics of the flow is the same as that described

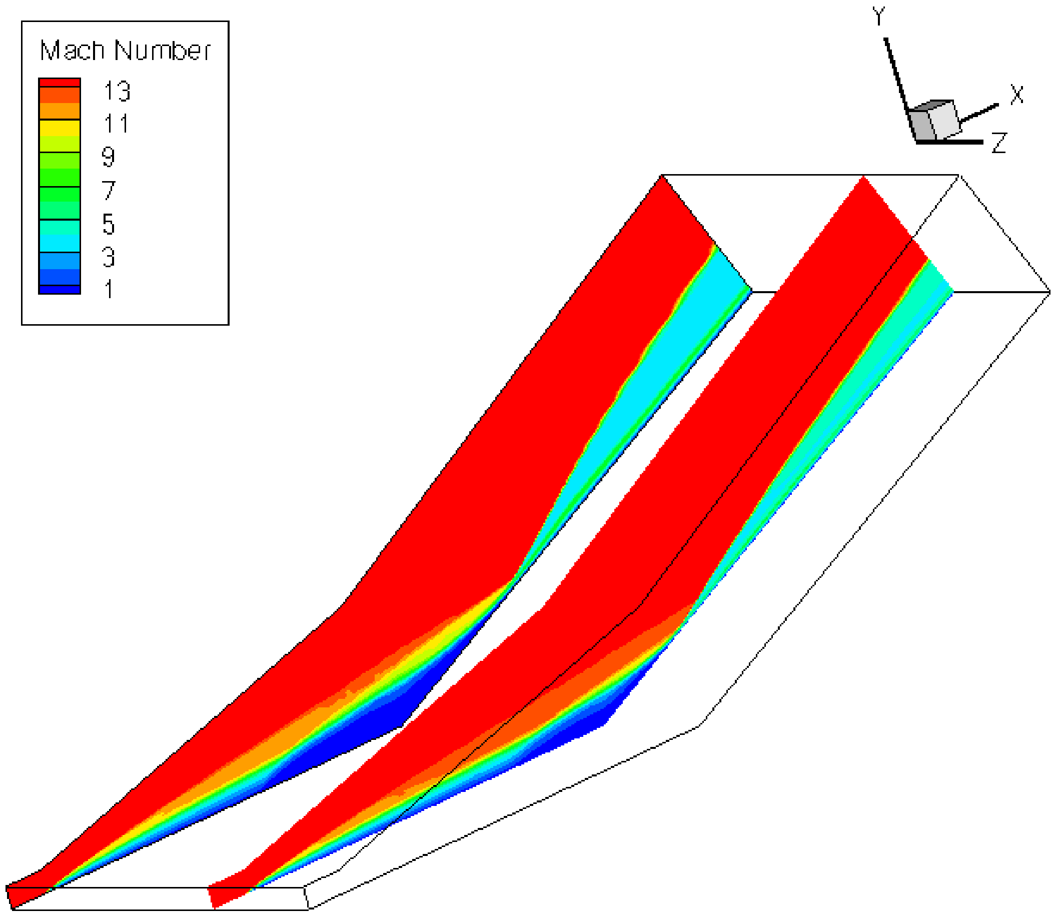


Figure 6.29: Hypersonic flow over a compression ramp: Mach number contour plots at two $x - y$ planes. The left plane is the ramp symmetry plane, while the right one is at the edge of the ramp.

in section 5.3: the high pressure propagates upstream through the subsonic part of the boundary layer causing flow separation and reattachment.

We have simulated the experimental flow conditions of Holden and Moselle (98). The ramp consists of a flat plate of length $L = 0.4389$ m, followed by a wedge of the same length with 24° semi-angle. The ramp is 0.6096 m wide and has no side walls. The free-stream Mach and Reynolds numbers are:

$$M_\infty = \frac{V_\infty}{a_\infty} = 14.1, \qquad Re = \frac{\rho_\infty V_\infty L}{\mu_\infty} = 103700.$$

The free-stream temperature is $T_\infty = 88.88$ K. The model is internally cooled and the

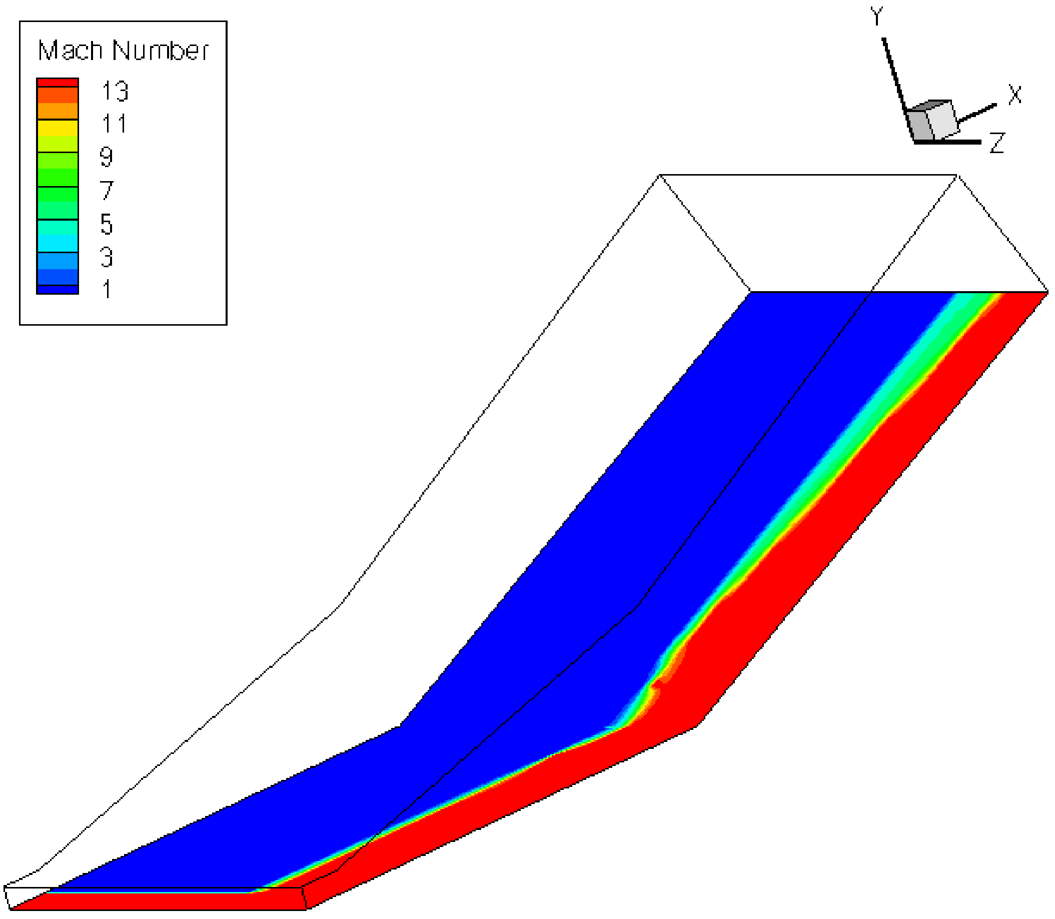


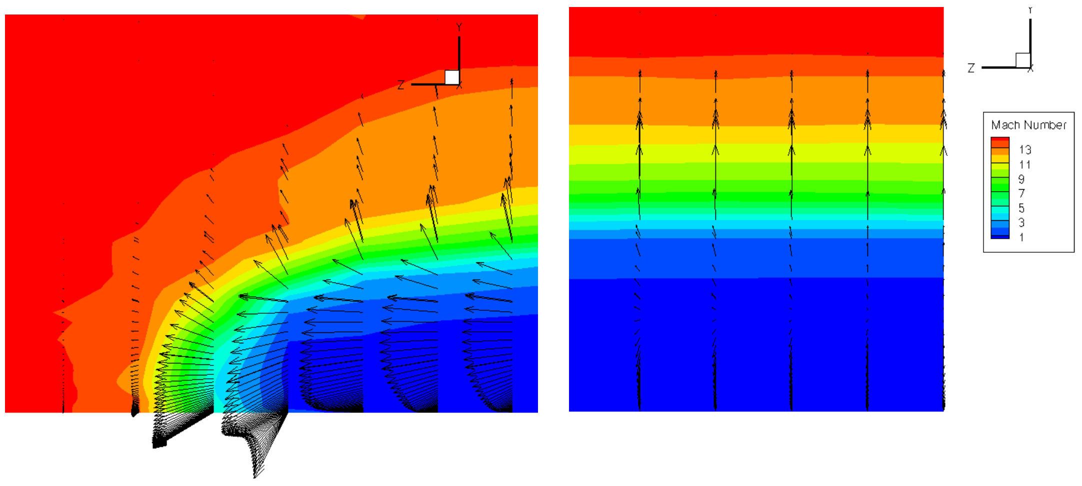
Figure 6.30: Hypersonic flow over a compression ramp: Mach number contour plots at the ramp surface plane.

wall temperature is $T_{wall} = 297.2$ K. Experimental values of the pressure coefficient, friction coefficient and Stanton number at the ramp center-line are available. The Stanton number is:

$$St = \frac{q_w}{\rho_\infty V_\infty (H_\infty - H_w)},$$

where q_w is the wall heat flux, and H_∞ and H_w are the free-stream and wall total enthalpy, respectively.

Rudy *et al.* (97) showed that the short width of the ramp and the absence of side walls in this experiment yielded a 3D flow structure: 2D flow simulations typically predict a longer separation region and lower peaks of wall pressure, friction and heat



(a) Border of the ramp (b) Symmetry plane

Figure 6.31: Hypersonic flow over a compression ramp: cross flow at the corner of the ramp.

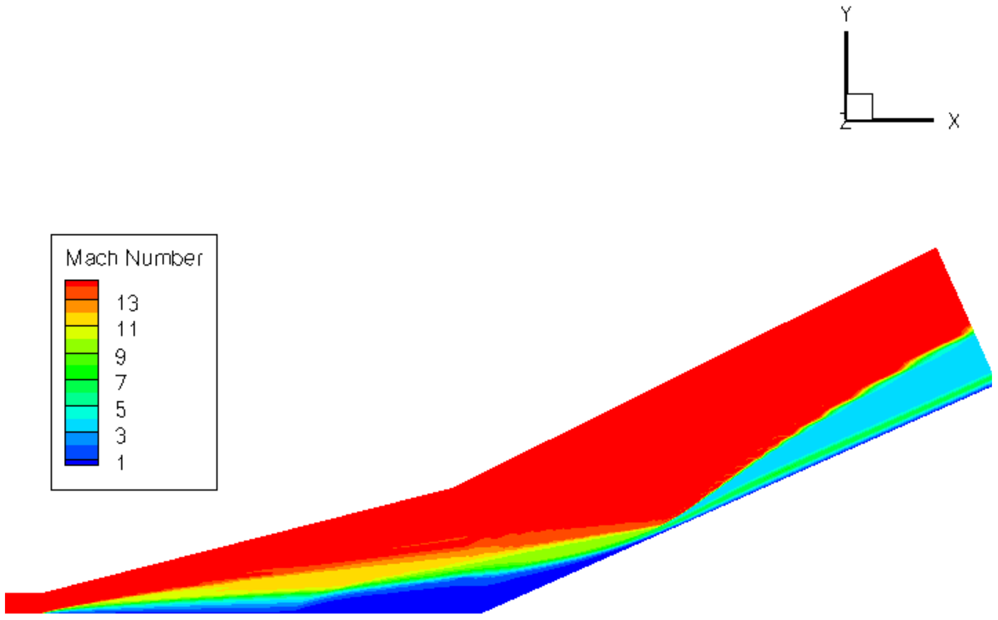
flux. On the other hand, 3D simulations are in good agreement with the experimental data.

We run a 3D simulation employing a grid of $198 \times 64 \times 32$ nodes. The grid is finer in the x–direction at the leading edge and at the corner. In the y–direction it is geometrically stretched with $\Delta y_w = 6 \times 10^{-5}$. The grid is uniform in the z–direction.

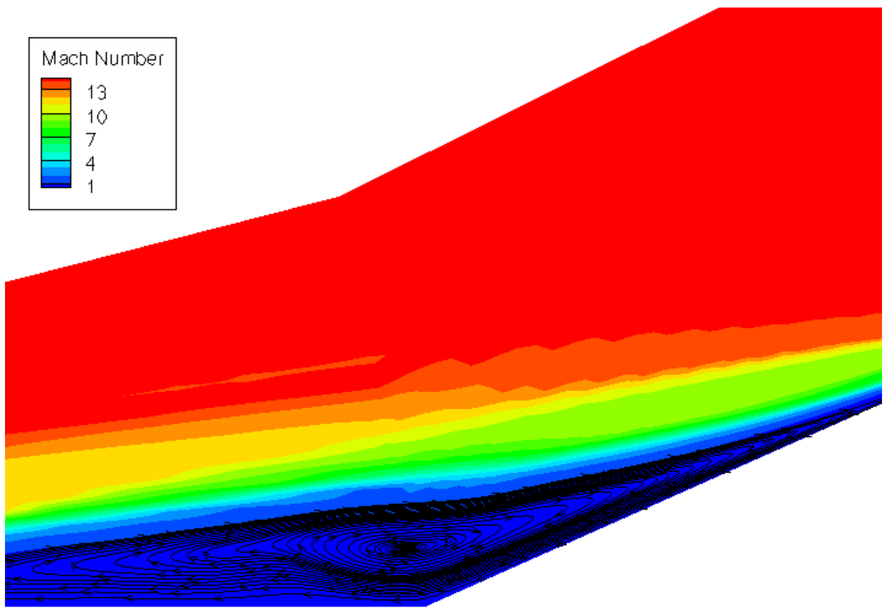
We simulate half of the geometry and enforce the slip-wall boundary condition at the boundary $z = 0$. The ramp surface is modelled as an isothermal no-slip wall and it is 22 nodes wide in the z–direction. The surface upstream from the plate leading-edge is modelled as a slip-wall. Farfield boundary conditions are enforced at all other boundaries.

Steady-state Mach number contours of the simulated 3D flow are shown in Figs. 6.29–6.33. In Fig. 6.29 $x - y$ slices of the flow field at the symmetry plane and at the ramp border are shown. It is evident that the separation region, identified by the low Mach numbers, shrinks from the symmetry plane along the span of the ramp. Figs. 6.28 and 6.30 show that border effects increase as the slope of the ramp changes.

Fig. 6.31 shows the cross flow at the ramp corner: the cross flow is driven by the span-wise pressure gradient in the shock layer and goes from the symmetry plane towards the border of the ramp. Fig. 6.32 shows the Mach number and the streamlines

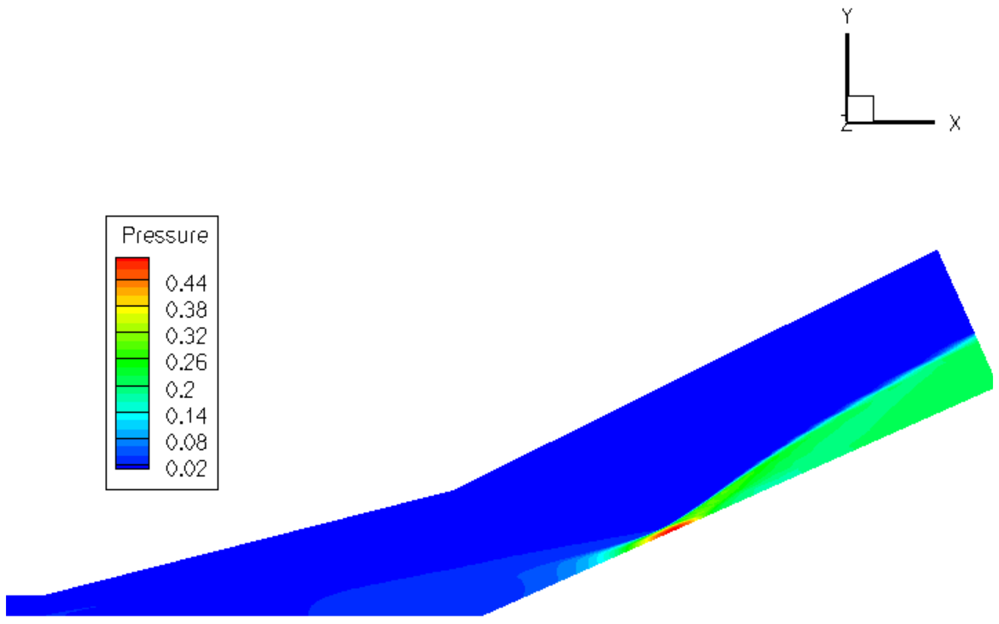


(a) Mach number

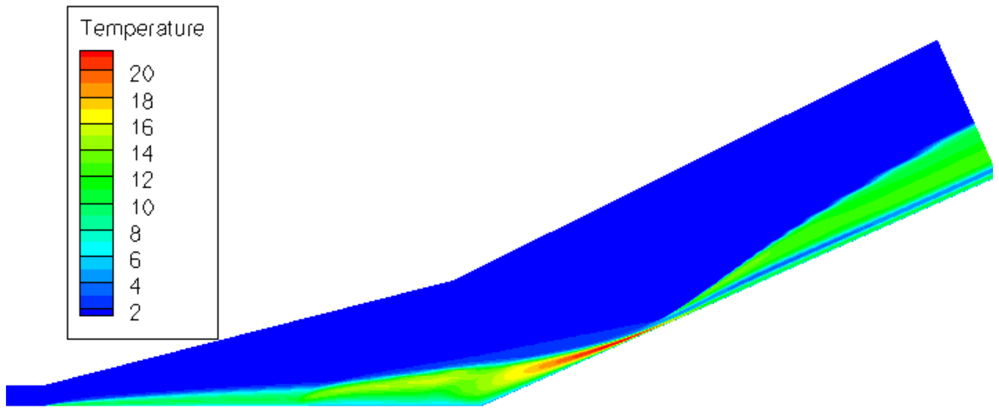


(b) Streamlines

Figure 6.32: Hypersonic flow over a compression ramp: contour plots and streamlines at the symmetry plane.

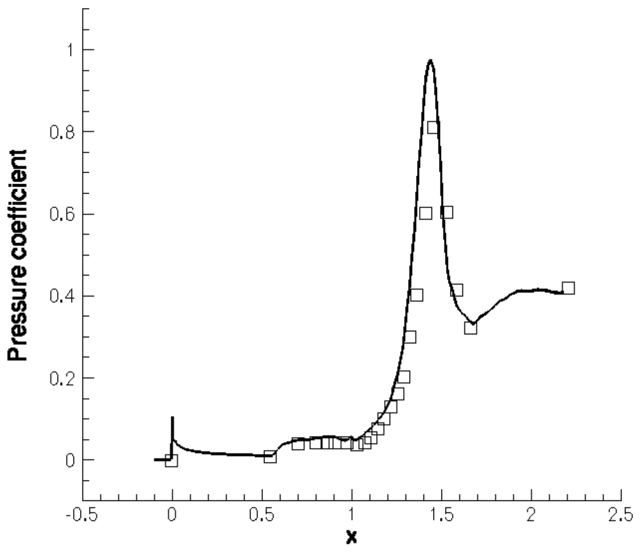


(a) Pressure

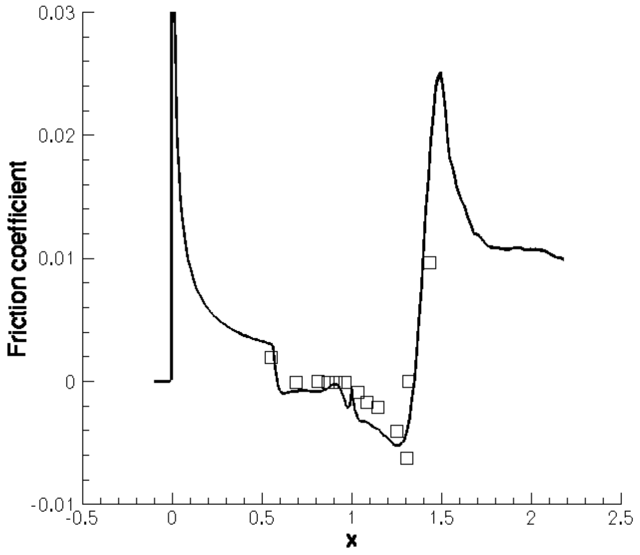


(b) Temperature

Figure 6.33: Hypersonic flow over a compression ramp: contour plots at the symmetry plane.



(a) Pressure coefficient



(b) Friction coefficient

Figure 6.34: Hypersonic flow over a compression ramp: wall coefficients; –, CUDA-CU5-TVD-NS; □, experiment (98).

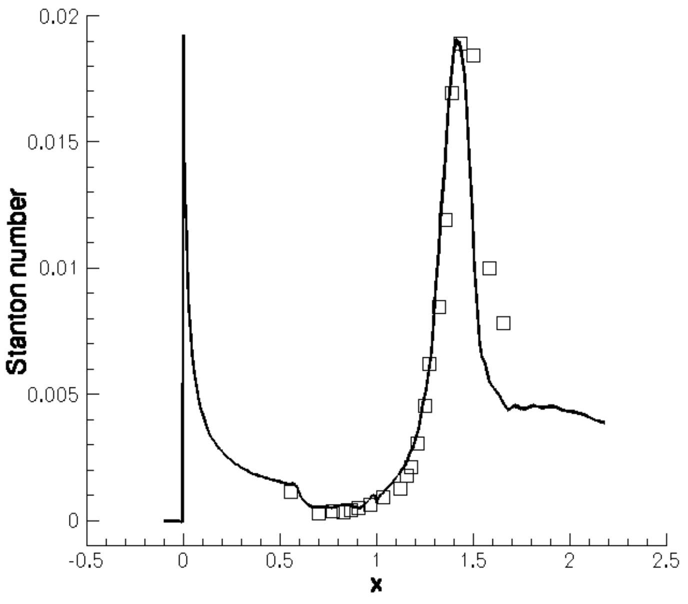


Figure 6.35: Hypersonic flow over a compression ramp: Stanton number; – CUDA-CU5-TVD-NS; □, experiment (98).

in the separation region at the symmetry plane. Pressure and temperature contours at the symmetry plane are shown in Fig. 6.33.

Finally, the computed wall coefficients are compared to the experimental data in Figs. 6.34 and 6.35. The extent of the separation region is correctly predicted. The overall agreement is good, although our code slightly over-predicts the pressure peak and under-predicts the heat flux after reattachment.

6.2.3 Performance of the GPU implementation

In this section we assess the performance of our GPU-accelerated Navier-Stokes solver, namely CUDA-CU5-TVD-NS. Since CU5-TVD-NSK and CUDA-CU5-TVD-NS are quite different in terms of implementation, a performance comparison between the two codes would be rather misleading. Therefore, we have ported CUDA-CU5-TVD-NS to CPU, and we refer to this implementation as OMP-CU5-TVD-NS. This also gives us the opportunity to extend the performance analysis of the method in a shared-memory environment, as presented in section 6.1.4. The structured programming adopted in

CUDA-CU5-TVD-NS made the porting easy. The only differences between the two codes are:

1. Data arrays are allocated in the CPU memory instead of the GPU memory.
2. CUBLAS calls are substitued by BLAS calls.
3. The library OMP-CFD is used in lieu of CUDA-CFD.

The OMP-CFD library is a multi-thread CPU version of CUDA-CFD: all procedures but `cudaTranspose()` have been ported to CPU using OpenMP and analogous partitioning strategies. Data transposition is performed using the function `mkl_somatcopy()` from the Intel Math Kernel Library (MKL) (`http://software.intel.com/en-us/articles/intel-mkl`). Also, all BLAS calls link to the MKL multi-thread implemetation of the appropriate procedure.

Our tests were performed on the Distributed Computing (DisCo) system CSEHT at STFC Daresbury Laboratory (`http://www.cse.scitech.ac.uk/disco`). The DisCo cluster hosts 8 Nehalem servers, each one consisting of an Intel Xeon quad-core 2.8 GHz and an Nvidia Tesla S1070 graphic card. Each Tesla comprises four GPUs, each one with its own memory. Neither CUDA-CU5-TVD-NS nor OMP-CU5-TVD-NS possess distributed-memory capabilities, and so we only use one quad-core CPU and one GPU.

We first analyse the performance of each procedure, so that the performance of the code can be properly interpreted. The procedures are tested on three dummy grids: G1, consisting of $32 \times 32 \times 32$ ($\approx 33 \times 10^3$) nodes; G2, consisting of $64 \times 64 \times 64$ ($\approx 262 \times 10^3$) nodes; G3, consisting of $128 \times 128 \times 128$ ($\approx 2 \times 10^6$) nodes. Table 6.3 gives the *specific runtimes*,

$$\text{Procedure specific runtime} = \frac{\text{Procedure runtime}}{\text{Number of calls} \times \text{Grid size}},$$

on a single CPU core. The specific times shown are averaged over 100 calls to each procedure.

The most time-consuming task is the characteristic TVD filter (procedure `ompTVD()`), taking about seven times as long as the computation of the kinetic split flux (procedure `ompSplitFlux()`) and the compact reconstruction (procedure `sgemm()`). Specific times are about constant with the grid size for all procedures except `ompNSCBC()` and `sgemm()`.

Procedure	Specific time, $10^{-4}\mu$s		
	33×10^3 nodes	262×10^3 nodes	2×10^6 nodes
`ompTVD()`	47.6	49.2	49.5
`ompSplitFlux()`	7.17	7.24	7.33
`sgemm()`	2.53	4.36	7.80
`ompNSCBC()`	5.44	2.54	1.32
`ompODErhs()`	1.47	1.57	1.69
`ompCons2prim()`	1.14	1.24	1.28
`ompSdmv()`	0.358	0.360	0.369
`ompTranspose()`	0.333	0.282	0.310
`ompViscosity()`	0.106	0.105	0.146

Table 6.3: Specific runtimes on a single CPU core for different procedures and different grid size.

The time taken to enforce the boundary conditions grows as a 2/3 power law of the grid size, because boundary nodes only are involved, and so the specific time decreases with the grid size.

On the other hand, the cost of a matrix-matrix product grows super-linearly with the matrix size, and so the specific runtime of `sgemm()` increases with the grid size. Assuming that the OMP-CU5-TVD-NS specific runtime is about equal to the sum of the specific runtimes of the three most time-consuming procedures, the use of the matrix-matrix multiplication brings up the specific time by about 10% when going from G1 to G3. Therefore, the use of a tri-diagonal solver in lieu of the matrix-matrix multiplication is estimated to reduce the runtime by 10% at most on grids of a few million nodes. In practice, the impact of the compact reconstruction on the total runtime is much smaller, because the matrix transposition has a large impact on the runtime. Although the matrix transpose is inexpensive with respect to other tasks, it is performed many more times and, eventually, affects significantly the runtime, as shown later in this section.

In Figs. 6.36–6.40 procedure *speed-ups* are shown, where:

$$\text{Speed-up} = \frac{\text{Runtime on one CPU core}}{\text{Runtime}}.$$

The CPU code runs on an Intel Xeon E5560. This processor has Hyper-Threading (HT) capability, allowing each core to run up to two threads in parallel. The HT technology enables overlapping between different operations of different threads; if, for example, a core is assigned threads 0 and 1, it runs additions for thread 0 in parallel with multiplications for thread 1. Although this is not equivalent to running two threads on two separate cores, some codes do show improved performance. Therefore, we show CPU speed-ups on up to eight threads, although the eight threads run, in fact, on four cores.

Fig. 6.36(a) shows speed-up for the most time-consuming task, the characteristic-based TVD filter. The CPU implementation scales linearly with the number of threads, achieving a speed-up of 7.9 on four cores using HT. This is an outstanding result and testifies to the high quality of the implementation and of the processor. The speed-up is weakly sensitive to the grid size, ranging from 7.1 to 7.9 on eight threads as bigger problems are considered. Despite the great perfomance of the CPU implementation, the GPU outperforms it by far, running more than 50 times faster on the finest grid. This datum is particularly important, as `cudaTVD()` has 6% multi-processor occupancy only (see Table 4.1). It demonstrates that the high arithmetic intensity of the task minimises the impact of the occupancy. The speed-up is weakly sensitive to the grid size, ranging from 45.7 to 50.7 as bigger problems are considered.

Analogous results are found for the computation of the kinetic split flux (Fig. 6.36(b)). The CPU implementation scales linearly with the number of threads, but the GPU implementation shows even better performance than in the previous case, achieving a speed-up of 68.5 on the finest grid. Indeed, this is due to the higher multi-processor occupancy achieved by the `cudaSplitFlux()` procedure (Table 4.1).

Speed-up results for the compact reconstruction show a more complex behaviour (Fig. 6.37(a)). The MKL implementation of `sgemm()` scales super-linearly on grids G1 and G2 up to four threads, and linearly on the finest grid G3. HT delivers a super-linear speed-up on the intermediate grid G2 and improves the speed-up on the fine grid G3, but it worsens it on the coarse grid G1. This irregular behaviour is likely related to the cash-memory usage. Expected speed-ups are between 6 and 7 on four cores with HT enabled. The behaviour of `cublasSgemm()` is more regular. The speed-up is about 21.5 on intermediate and fine grids, while only 8.4 on the coarse one, that apparently does not supply enough computation to use effectively all GPU cores.

The NSCBC procedure shows super-linear speed-up on the CPU, and is weakly sensitive to the grid size, with speed-ups ranging from 7.9 to 9.1 on eight threads

(Fig 6.37(b)). The GPU speed-up is very sensitive to the grid size and varies between 21.9 and 37.5. The reason for such sensitivity is that boundary nodes only are involved in the computation, and therefore even the intermediate grid G2 is not enough to use effectively all of the GPU cores. Speed-ups are still quite high (especially considering the low occupancy) because of the high arithmetic intensity.

Procedures implementing "light" tasks show similar behaviour. Such tasks are: computation of the ODE right hand side (RHS) from the numerical flux function (Fig. 6.38(a)), conversion of the conserved variables into primitive (Fig. 6.38(b)), component-by-component vector multiplication (Fig. 6.39(a)). The CPU speed-up is close to linear with the number of threads. Speed-ups on the finest grid G3 are somewhat worse than those on the other two grids, most likely because of an unfavourable memory path. GPU speed-ups are good, ranging from 8.8 to 17.3, but not better because the tasks have low arithmetic intensity.

The computation of the viscosity is a light task but it performs well (Fig. 6.39(b)). The CPU implementation does not show the loss of speed-up on the finest grid, but scales almost linearly with the number of threads for all three grid sizes. The GPU speed-up is much higher, ranging from 45.7 to 50.7 as the problem size increases. This can likely be attributed to a higher arithmetic intensity.

The matrix transpsition is the weak point of both CPU and GPU implementations (Fig. 6.40). Apparently, the MKL procedure `mkl_somatcopy()` is not multi-thread, as no speed-up is seen as the number of threads is increased. The GPU implementation does not perform particularly well either, as speed-ups range from 4.3 to 12.3 depending on the grid size. The multi-processor occupancy is rather high, 75%, therefore the low speed-ups as well as the high sensitivity to the problem size are due to the low arithmetic intensity. Matrix transposition is, from the point of view of the arithmetic intensity, equivalent to a copy operation: each number is fetched from a memory location and copied somewhere else.

Finally, Fig. 6.41 shows the overall performance of the Navier-Stokes solvers. The hypersonic compression ramp is simulated on a "coarse" grid consisting of $198 \times 64 \times 32$ ($\approx 4 \times 10^5$) nodes, and on a fine grid consisting of $396 \times 128 \times 64$ ($\approx 3.1 \times 10^6$) nodes. Specific times for the NS solvers averaged over 100 time steps are shown in Table 6.4.

Going from one to two threads, OMP-CU5-TVD-NS runs from 1.5 to 1.8 times faster, depending on the grid size. This datum is very similar to the 1.6 speed-up ob-

Number of grid nodes	Specific time, μs				
	CPU				GPU
	1 thread	2 threads	4 threads	8 threads	
3.9×10^5	16.7	11.4	6.4	4.1	0.7
3.1×10^6	23.2	12.7	7.4	5.2	0.8

Table 6.4: Specific runtimes of multi-thread CPU and GPU NS solvers for two grid sizes.

tained running CU5-TVD-NSK on two cores of a quad-core AMD Opteron (Fig. 6.22). On the other hand, running OMP-CU5-TVD-NS on the Intel Xeon, we obtain a speed-up between 4.1 and 4.5 when using four cores with HT enabled, versus 1.6 obtained running CU5-TVD-NSK on four cores of the AMD Opteron. This datum supports the conclusion that the poor speed-up seen on the Opteron is due to its NUMA architecture, as the Intel Xeon SMP architecture allows the code to scale beyond two cores.

Although OMP-CU5-TVD-NS scales very well, its speed-up is less impressive than the 8× speed-up seen for most procedures. The reason for the performance gap lies in the matrix transposition. This operation is not parallel and is performed for many variables; it therefore has a non-negligible effect on the total runtime, that becomes more and more important as the number of threads increases.

The GPU code is between 24.3 and 29.3 times faster than the CPU code running serially. These speed-ups are intermediate between the very high values (40–70) achieved by tasks with high arithmetic intensity and the lower values (10–20) achieved by those with low arithmetic intensity. As for the CPU implementation, the performance is strongly bounded by the performance of the matrix transposition, and the effect on the GPU is even greater: the most time-consuming tasks for the CPU are also those that experience the greater speed-up on the GPU, and so the transposition time represents for the GPU a bigger percentage of the total runtime.

Data transposition is the price paid by semi-global methods. It is essential for the code to follow the best memory pattern, which is a requirement for all procedures to be accelerated. Therefore, further speed-up of the code is unlikely to be achieved by looking at the matrix transpose procedure, which is already highly optimised. More likely, further speed-up could be obtained by trying to increase the ocupancy of procedures with high arithmetic intensity and low occupancy, such as `cudaTVD()` and `cudaNSCBC()`. This could be achieved by breaking the procedures into sub-procedures with slighly lower arithmetic intensity but also less demanding memory requirements and so higher occupancy.

6.3 Summary

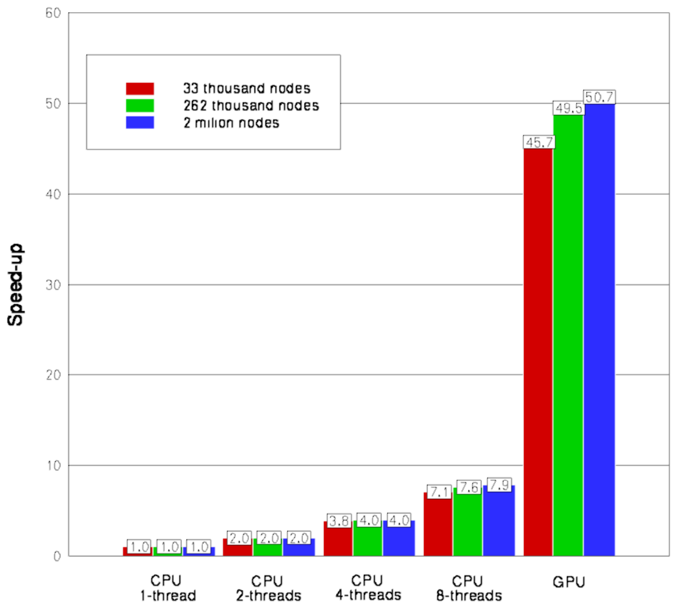
Novel contributions presented in this chapter are: a study of the effect of the domain decomposition on the computed solution for our multi-block algorithm, and a study of its parallel performance; the validation of our GPU code and a study of its performance. Our findings can be summarised as follows:

- For a flow as complex as a shock-jet interaction, single- and multi-block solutions have only a few minor differences. Interesting results are found for the supersonic flow past a cone-cylinder: the multi-block partitioning seems to cure a problem shown by the single-block method in capturing the boundary layer at the expansion corner between conical and cylindrical parts of the geometry. This test also allows us to show that our modification to the characteristic decomposition described in section 2.3.3 is essential to simulate this corner flow.

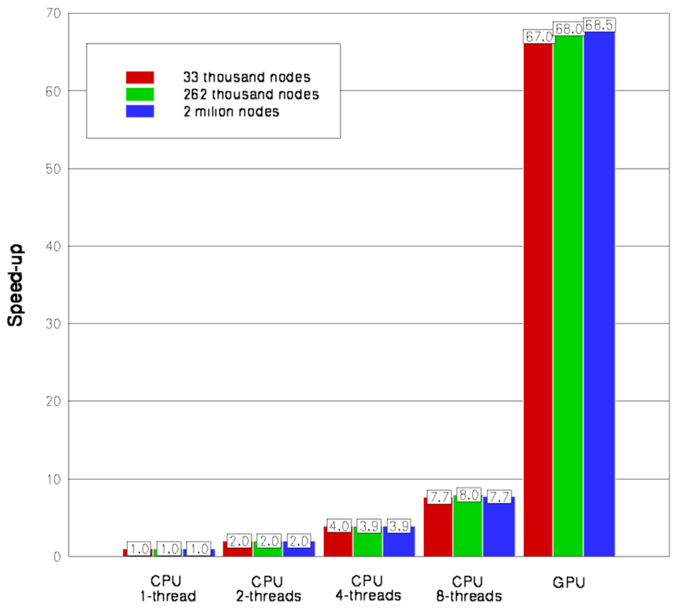
- Our multi-block algorithm is suitable for massively parallel computing: the parallel efficiency is about 90% when running a simulation of one million cells using one thousand CPU cores. We also highlight the performance issue of compact methods running on NUMA processors: our compact-TVD method does not scale beyond two cores on a NUMA processor if the shared-memory slab decomposition is used.

- Our GPU code is validated against two tests cases: more have been performed but their results, similar to those shown in Chapter 5 for the CPU code, are not shown. Despite single-precision being used in the floating-point calculations, the GPU code successfully simulates vortex pairing in a compressible mixing layer and 3D hypersonic flow over a compression ramp.

- Our task-dependent domain decomposition is shown to be very effective: the GPU speed-up is about 29× with respect to one CPU core when running a simulation of 3 million nodes. Our multi-thread CPU code employing the same decomposition achieves 4× speed-up when running on four cores of an SMP processor. We have

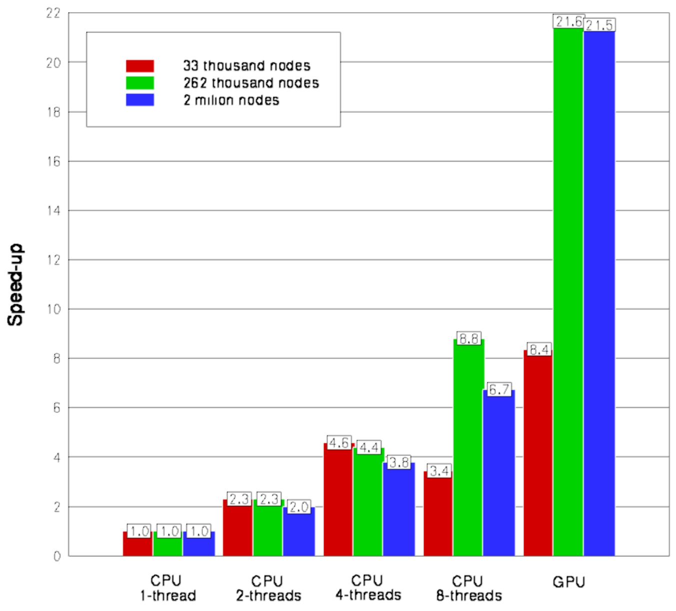


(a) Characteristic-based TVD filter

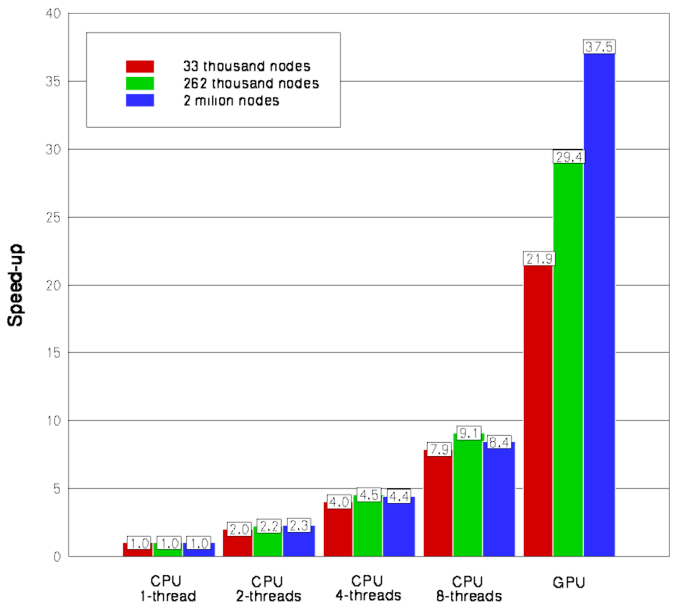


(b) Kinetic Flux Vector Splitting

Figure 6.36: Performance of the multi-thread CPU and GPU implemetations of the indicated tasks for different mesh sizes.

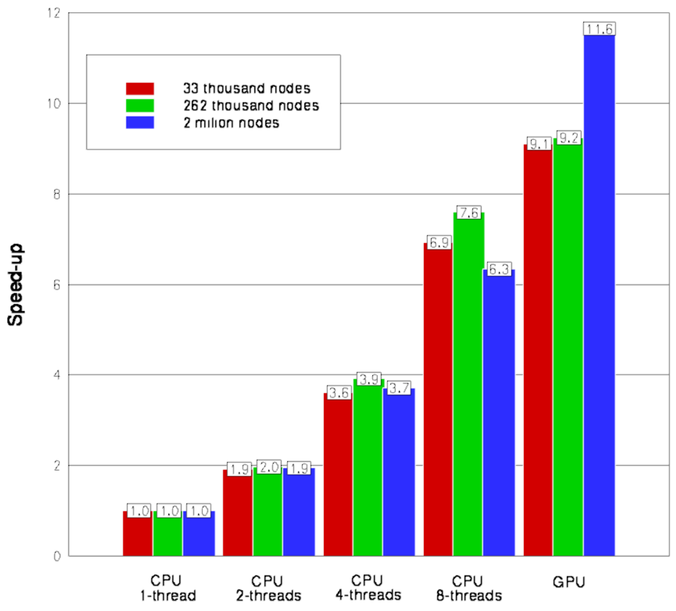


(a) Compact reconstruction, MKL sgemm() vs. cublasSgemm()

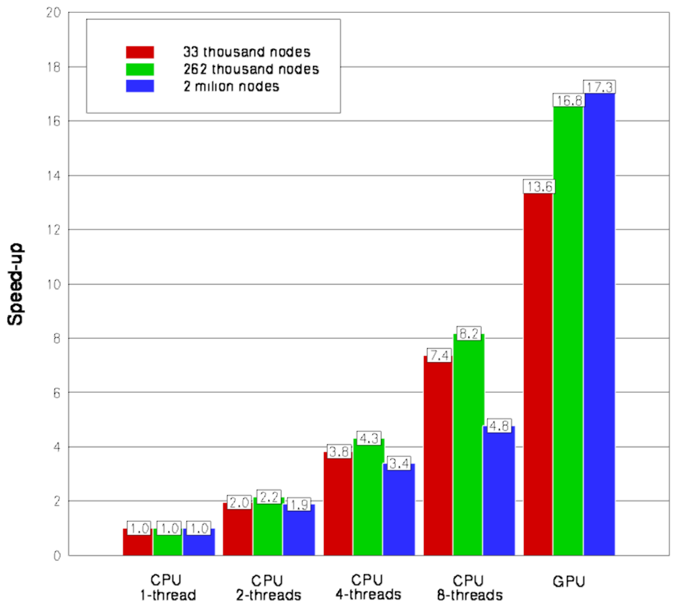


(b) Navier-Stokes Characteristic Boundary Conditions enforcement

Figure 6.37: Performance of the multi-threaded CPU and GPU implemetations of the indicated tasks for different mesh sizes.

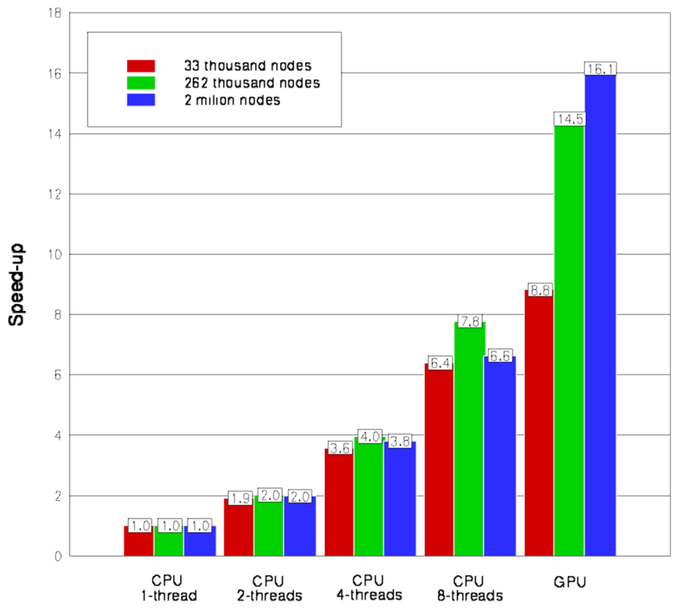


(a) ODE RHS computation from numerical flux function

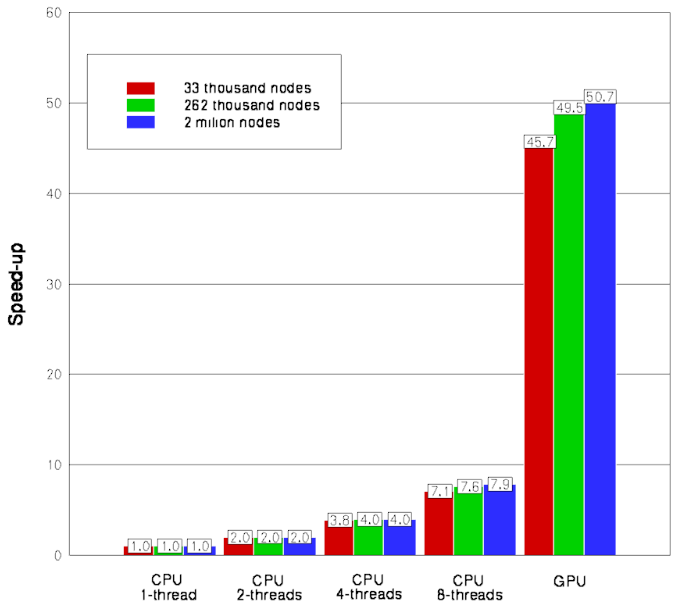


(b) Coversion of conserved variables into primitive

Figure 6.38: Performance of the multi-threaded CPU and GPU implemetations of the indicated tasks for different mesh sizes.



(a) Component-by-component vector multiplication



(b) Viscosity computation

Figure 6.39: Performance of the multi-threaded CPU and GPU implemetations of the indicated tasks for different mesh sizes.

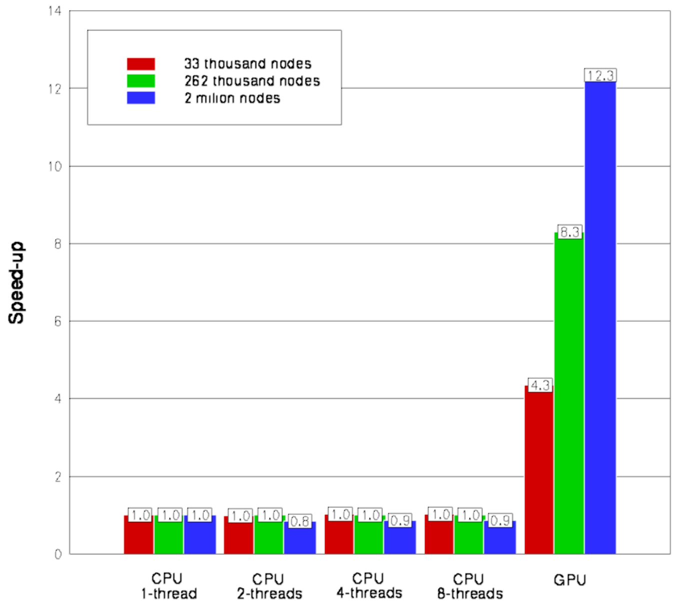


Figure 6.40: Performance of the multi-threaded CPU and GPU implemetations of the matrix transpose for different mesh sizes.

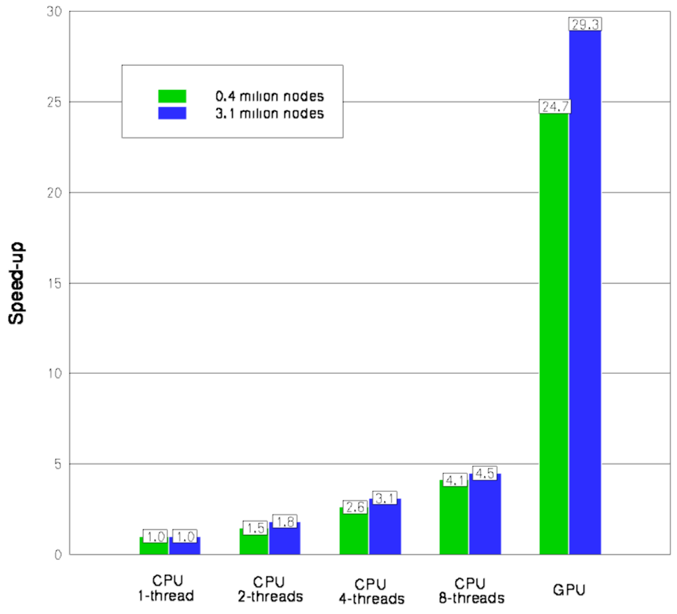


Figure 6.41: Performance of the multi-thread CPU and GPU Navier-Stokes solvers for different mesh sizes.

studied how each elementary task contributes to the overall speed-up and found that the matrix transposition imposes a cap on the performance: it has no speed-up on the CPU because its implementation is not multi-thread, while it has a speed-up of only about 10× on the GPU because of its low arithmetic intensity.

7

Conclusions

In this thesis we have presented the development of a high-order high-resolution numerical method for hypersonic flow simulation in the framework of high performance computing. Our major findings are summarised in this chapter.

7.1 Simulation of inviscid flows

In Chapter 2 we presented the compact-TVD method for the Euler equations that we used as a starting point for our study. The method proposed by Tu and Yuan (25) was chosen because of its high-accuracy and good resolution properties, its conservative formulation, stability and relatively low computational cost. We used the high-accuracy Kinetic FVS, unlike Tu and Yuan who use the classic Steger-Warming FVS.

Inviscid flow simulations were performed to investigate the resolution properties of the method, and results shown in Chapter 5. We found that our compact-TVD method shows better resolution than methods of comparable accuracy that employ a WENO scheme to capture discontinuities.

7.2 Simulation of viscous flows

In Chapter 3 we presented a Navier-Stokes solver based on the compact-TVD method presented in Chapter 2. This solver employs a splitting for the Navier-Stokes flux.

Our main contribution is having identified the possibility to combine the Kinetic FVS for the NS flux, devised by Chou and Baganoff (56), with the compact-TVD method. Combining the two techniques has several advantages:

- the computational cost of the compact scheme, associated with the solution of linear systems to compute derivatives, is reduced because fewer linear systems are solved;
- the loss of accuracy associated with the TVD filter is mitigated, because the filter is applied to the NS flux, which is smoother than the inviscid flux.

In Chapter 5 the method was validated against steady viscous flows. The explicit time discretisation makes the method less ideal for steady state calculations, as the tight time-step restriction for stability leads to long computing times. On the other hand, the method is intended to be applied mainly for time-accurate unsteady simulations. If steady state calculations are to be performed in the future, an implicit time discretisation will have to be implemented.

Our numerical results were in excellent agreement with exact solutions of smooth compressible flows, namely the shock-layer and the supersonic boundary layer over a flat plate. Comparison to experimental results was provided for two simulations, namely the shock-boundary layer interaction and the hypersonic flow past a blunted cone. For the shock-boundary layer interaction, wall pressures were in excellent agreement with the experimental data. A discrepancy between our CFD and experimental values of the friction coefficient after reattachment was found. However, our prediction agreed with independent CFD data by other authors (89, 90, 91). For the hypersonic flow past a blunted cone, our CFD was in good agreement with the experimental data, although a minor discrepancy was observed at some points downstream from the conical shock.

In section 6.1.2 we highlighted the difficulty the method has in capturing viscous supersonic flows around expansion corners. The test case was the supersonic flow past a cone-cylinder geometry, and the expansion corner was the one between the conical and cylindrical parts of the geometry. The method predicted a high-amplitude unphysical oscillation in the boundary layer profiles all along the geometry. We tackled this problem by modifying characteristic treatment in the TVD limiting step, as detailed in section 2.3.3. This eased the problem although did not completely solve it, as the friction coefficient was still overpredicted at the corner.

7.3 Parallel application of the Navier-Stokes solver

In Chapter 4 we presented a novel algorithm to parallelise our compact-TVD method. The algorithm is based on structured-block partitioning and improves previous algorithms that employ the same approach (32, 33, 35). Our algorithm is fully conservative, does not distort smooth flow features and introduces a minimal perturbation on the original linear system to compute derivatives.

In Chapter 6 we studied the effect of this perturbation on the computed solution. For a flow as complex as a shock-jet interaction single- and multi-block solutions have only a few minor differences. Interesting results were found for the supersonic flow past a cone-cylinder: the multi-block partitioning seemed to cure the problem shown by the single-block method in capturing the boundary layer at the expansion corner between the conical and cylindrical parts of the geometry.

Our multi-block algorithm was proven to be suitable for massively parallel computing: we studied its parallel performance and found that its parallel efficiency is about 90% when running a simulation of one million cells using one thousand CPU cores.

In Chapter 4 we also presented a strategy to parallelise our compact-TVD method on a cluster of multi-core processors. We proposed combining our multi-block parallel algorithm with the slab/drawer decomposition algorithm (34): the small communication cost of the multi-block algorithm made it ideal to divide the work between processors; the slab/drawer decomposition was used to divide the work between the cores of a processor. In Chapter 6 we studied the performance of the shared-memory kernel and found that the shared-memory slab decomposition algorithm did not scale beyond two cores. We identified the reason for such a poor scaling in the NUMA architecture of the processor used: in section 6.2.3 we showed that a code implementing a similar method and partitioning strategy scaled linearly up to four cores on an SMP processor.

7.4 GPU application of the Navier-Stokes solver

In Chapter 4 we presented a strategy to parallelise our compact-TVD method in the complex computing environment of a GPU. Our strategy consisted of two steps:

- algorithm break-down into elementary tasks;
- adoption of a task-dependent domain partitioning.

Breaking the algorithm into elementary tasks allows us to meet the tight memory constraints of the GPU. The runtime of each task was minimised by choosing the appropriate partitioning strategy.

In Chapter 6 our GPU code was validated against two tests cases: more have been performed but their results, similar to those shown in Chapter 5 for the CPU code, were not shown. Despite single-precision being used in the floating-point calculations, the GPU code successfully simulated vortex pairing in a compressible mixing layer and 3D hypersonic flow over a compression ramp.

In Chapter 6 we also presented a performance study of our GPU code. Our task-dependent partitioning strategy was shown to be very effective: the speed-up was about $29\times$ with respect to one CPU core when running a simulation of 3 million nodes. We studied how each elementary task contributed to the overall speed-up and found that the matrix transposition imposes a cap on the performance: it has a speed-up of only about $10\times$ on the GPU because of its low arithmetic intensity.

7.5 Suggestions for future work

Our GPU code, CUDA-CU5-TVD-NS, shows excellent performance when running on a single GPU. In the future the code could undergo two major developments:

- Development of CPU-GPU work-sharing capability: part of the computation could be performed by the host CPU in parallel with the GPU. Tasks with low arithmetic intensity could perhaps be offloaded on the CPU: our experiments show that GPU runtimes for these tasks are only marginally better than quad-core CPU runtimes (see for example Fig. 6.38(a)). The development of this capability requires careful thinking: data transfer between device and host memory is slow, and, if not overlapped with computation, can outrule the benefit of work-sharing between GPU and CPU.

- Development of distributed-memory capability: the work-load could be shared between multiple GPUs using our multi-block algorithm, CU5-MB, for example.

The code could be applied in the fields of compressible LES, DNS and aeracoustics, where the use of a compact scheme is expected to provide an advantage over codes using explicit finite difference schemes.

References

[1] S. K. Lele. **Compact finite difference schemes with spectral-like resolution**. *Journal of Computational Physics*, **103**:16–42, 1992. vi, 5, 6, 7, 8, 29, 47, 59, 122

[2] S. Laizet, E. Lamballais, and J.C. Vassilicos. **A numerical strategy to combine high-order schemes, complex geometry and parallel computing for high resolution DNS of fractal generated turbulence**. *Computers & Fluids*, **39**(3):471 – 484, 2010. vii, 16, 73, 74, 145, 175, 206

[3] N. Li and S. Laizet. **2DECOMP&FFT-A Highly Scalable 2D Decomposition Library and FFT Interface**. In *CUG 2010 Proceedings*, 2010. vii, 74, 75, 175

[4] P. Woodward and P. Colella. **The numerical simulation of two-dimensional fluid flow with strong shocks**. *Journal of Computational Physics*, **54**:115–173, 1984. vii, viii, ix, 99, 100, 101, 102, 103, 104, 105, 113, 114, 117

[5] Dinshaw S. Balsara, Tobias Rumpf, Michael Dumbser, and Claus-Dieter Munz. **Efficient, high accuracy ADER-WENO schemes for hydrodynamics and divergence-free magnetohydrodynamics**. *J. Comput. Phys.*, **228**:2480–2516, April 2009. viii, ix, 106, 107, 110, 116, 118

[6] Gang Li and Jianxian Qiu. **Hybrid weighted essentially non-oscillatory schemes with different indicators**. *J. Comput. Phys.*, **229**:8105–8129, October 2010. viii, ix, 107, 110, 117, 119

[7] Jing Shi, Yong-Tao Zhang, and Chi-Wang Shu. **Resolution of high order WENO schemes for complicated flow structures**. *Journal of Computational Physics*, **186**(2):690 – 696, 2003. ix, 115, 118

[8] T.A. Reyhner and I. Flügge-Lotz. **The interaction of a shock wave with a laminar boundary layer**. *International Journal of Non-Linear Mechanics*, **3**(2):173 – 199, 1968. ix, 128

[9] R. J. Hakkinen, I. Greber, L. Trilling, and S. S. Abarbanel. **The Interaction of an Oblique Shock Wave with a Laminar Boundary Layer**. Memo 2-18-59W, NASA, 1959. ix, 131, 132

[10] Y. Yao, L. Krishnan, N. D. Sandham, and G. T. Roberts. **The effect of Mach number on unstable disturbances in shock/boundary-layer interactions**. *Physics of Fluids*, **19**(5):054104–+, May 2007. ix, 131, 133, 144, 205

[11] J. W. Cleary. **An experimental and theoretical investigation of the pressure distribution and flow fields of blunted cones at hypersonic Mach numbers**. Technical Report TN D-2969, NASA, 1969. ix, x, 134, 140, 141

[12] N. D. Sandham and H. C. Yee. **Performance of low dissipative high order shock-capturing schemes for shock-turbulence interactions**. Technical Report NASA-CR-208354, NASA Ames Research Center, 1998. xiii, 176, 181, 182, 183

[13] M. S. Holden and J. R. Moselle. **Theoretical and experimental studies of the shock wave-boundary layer interaction on compression surfaces in hypersonic flow**. Technical Report ARL 70-0002, Wright Patterson Aerospace Research Laboratories, 1970. xiii, 176, 185, 190, 191

[14] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer, Berlin Heidelberg, 1997. xv, 146, 148

[15] John D. Jr. Anderson. *Hypersonic and High-Temperature Gas Dynamics*. AIAA Education Series. American Institute Aeronautics and Astronautics, second edition, 2006. 1, 124, 126, 127, 134

[16] N. D. Sandham and H. C. Yee. **A numerical study of a class of TVD schemes for compressible mixing layers**. Technical Report NASA-TM-102194, NASA Ames Research Center, 1989. 2, 183

[17] D. Gottlieb and S. A. Orszag. *Numerical analysis of spectral methods*. SIAM, Philadelphia, 1977. 6

[18] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral methods in fluid dynamics*. Springer-Verlag, New York, 1987. 6

[19] C. K. W. Tam and J. C. Webb. **Dispersion-relation-preserving finite difference schemes for computational acustics**. *Journal of Computational Physics*, **107**:262–281, 1993. 8

[20] C. D. Pruett, T. A. Zang, C. L. Chang, and M. H. Carpenter. **Spatial direct numerical simulation of high speed boundary layer flows. Part I. Algorithmic consideration and validation**. *Theoretical Computational Fluid Dynamics*, **7**:397–424, 1995. 8

[21] M. H. Carpenter, D. Gotlieb, and S. Abarbanel. **The stability of numerical boundary treatments for compact high order finite difference schemes**. *Journal of Computational Physics*, **108**:272–295, 1993. 8

[22] N. A. Adams and K. Shariff. **A high-resolution hybrid compact-ENO scheme for shock-turbulence interaction problems**. *Journal of Computational Physics*, **127**:27–51, 1996. 9, 10

[23] A. I. Tolstykh and M. V. Lipavskii. **On performance of methods with third and fifth order compact upwind differencing**. *Journal of Computational Physics*, **140**:205–232, 1998. 9

[24] M. Zhuang and R. F. Cheng. **Optimized upwind dispersion-relation-preserving finite difference schemes for computational aeroacustics**. *AIAA Journal*, **36**:2146–2148, 1998. 9

[25] D. Gaitonde and J. S. Shang. **Optimized compact-difference-based finite-volume schemes for linear wave phenomena**. *Journal of Computational Physics*, **138**:617–643, 1997. 9

[26] M. H. Kobayashi. **On a class of Padeḟinite volume methods**. *Journal of Computational Physics*, **156**:137–180, 1999. 9

[27] A. W. Cook and W. H. Cabot. **A high-wavenumber viscosity for high-resolution numerical method**. *Journal of Computational Physics*, **195**:594–601, 2004. 9

[28] A. W. Cook and W. H. Cabot. **Hyperviscosity for shock-turbulence interaction**. *Journal of Computational Physics*, **203**:379–385, 2005. 9

[29] B. Fiorina and S. K. Lele. **An artificial nonlinear diffusivity method for supersonic reacting flows with shocks**. *Journal of Computational Physics*, **222**:246–264, 2007. 9

[30] A. W. Cook. **Artificial fluid properties for large eddy simulation of compresible turbulence mixing**. *Physics of Fluids*, **19**:055103, 2007. 9

[31] A. Harten, B. Engquist, S. Osher, and S. Chakravarthy. **Uniformly high order essentially non oscillatory schemes, III**. *Journal of Computational Physics*, **71**:231–303, 1995. 9, 23, 32

[32] C. W. Shu and S. Osher. **Efficient implementation of essentially non-oscillatory shock-capturing schemes**. *Journal of Computational Physics*, **77**:439–471, 1988. 9, 41, 98

[33] C. W. Shu and S. Osher. **Efficient implementation of essentially non-oscillatory shock-capturing schemes II**. *Journal of Computational Physics*, **89**:32–78, 1983. 9

[34] S. Pirozzoli. **Conservative hybrid compact-WENO schemes for shock-turbulence interaction**. *Journal of Computational Physics*, **178**:81–117, March 2001. 10, 16, 22, 29, 30

[35] Y. X. Ren, M. Liu, and H. Zhang. **A characteristic-wise hybrid compact-WENO scheme for solving hyperbolic conservation lows**. *Journal of Computational Physics*, **192**:365–386, 2003. 10, 70

[36] B. Cockburn and C. W. Shu. **Nonlinearly stable compact schemes for shock calculations**. *SIAM Journal of Numerical Analysis*, **31**:607–627, 1994. 10

[37] K. S. Ravichandran. **Higher order KFVS algorithms using compact upwind difference operators**. *Journal of Computational Physics*, **130**:161–173, 1997. 10, 48

[38] G. H. Tu and X. J. Yuan. **A characteristic-based shock-capturing scheme for hyperbolic problems**. *Journal of Computational Physics*, **225**:2083–2097, 2007. 10, 20, 22, 30, 33, 45, 48, 71, 92, 94, 204

[39] H. Lukoff. *From Dits to Bits: a personal history of the electronic computer*. Robotics Press, Portland, OR, 1979. 11

[40] J. E. Thornton. *Design of a computer - the control data 6600*. Scott, Foresman and Co., Glenview, IL, 1970. 12

[41] M. J. Flynn. **Some Computer Organizations and Their Effectiveness**. *IEEE Transactions on Computers*, **C-21**:948–960, September 1972. 13, 14

[42] X. H. Sun and S. Moitra. **A fast parallel tridiagonal algorithm for a class of CFD applications**. Technical Paper 3585, NASA, 1996. 16, 66

[43] A. Povitsky. **Parallel directional split solver based on reformulation of pipelined Thomas algorithm**. Technical Report CR-1998-208733, NASA, 1998. 16, 66

[44] F. Ladeinde, X. Cai, M. Visbal, and D. Gaitonde. **Parallel implementation of curvilinear high-order formulas**. *International Journal of Computational Fluid Dynamics*, **17**:2886–2898, 2003. 16, 66

[45] Datta V. Gaitonde. **Further Development of a Navier-Stokes Solution Procedure Based on Higher-Order Formulas**. In 37th *AIAA Aerospace Sciences Meeting and Exibit*, number AIAA-1999-0557, Reno, NV, January 1999. American Institute of Aeronautics and Astronautics. 16, 69, 91, 149, 206

[46] T. K. Sengupta, A. Dipankar, and A. Kameswara Rao. **A new compact scheme for parallel computing using domain decomposition**. *Journal of Computational Physics*, **220**:654–677, 2007. 16, 70, 76, 91, 146, 149, 206

[47] J. Chao, A. Haselbacher, and S. Balachandar. **A massively parallel multi-block compact-WENO scheme for compressible flows**. *Journal of Computational Physics*, **228**:7473–7491, 2009. 17, 70, 91, 206

[48] W. Huang and D. K. Tafti. **A Parallel Computing Framework for Dynamic Power Balancing in Adaptive Mesh Refinement Applications**. In *Proceedings of Parallel CFD 1999*, Williamsburg, VA, US (1999), May 1999. 17

[49] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. **High-performance parallel implicit CFD**. *Parallel Computing*, **27**(4):337 – 362, 2001. 17

[50] R. Rabenseifner, G. Hager, and G. Jost. **Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes**. In 17th *Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 427–436, Weimar, Germany, February 2009. 17

[51] J. Stam. **Stable fluids**. In *Proceedings of the* 26th *annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. 18

[52] Y. Liu, X. Liu, and E. Wu. **Real-Time 3D Fluid Simulation on GPU with Complex Obstacles**. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, PG '04, pages 247–256, Washington, DC, USA, 2004. IEEE Computer Society. 18

[53] M. Harris. **Fast fluid dynamics simulation on the GPU**. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, Los Angeles, California, 2005. ACM. 18

[54] W. Li, X. Wei, and A. Kaufman. **Implementing lattice Boltzmann computation on graphics hardware**. *The Visual Computer*, **19**:444–456, 2003. 10.1007/s00371-003-0210-6. 18

[55] W. Li, Z. Fan, X. Wei, and A. Kaufman. **GPU-Based flow simulation with complex boundaries**. In *GPU Gems 2*, pages 747–764. Addison-Wesley, 2003. 18

[56] J. Tölke and M. Krafczyk. **TeraFLOP computing on a desktop PC with GPUs for 3D CFD**. *Int. J. Comput. Fluid Dyn.*, **22**:443–456, August 2008. 18

[57] V. Simek, R. Dvorak, F. Zboril, and J. Kunovsky. **Towards Accelerated Computation of Atmospheric Equations Using CUDA**. In *Proceedings of the UKSim 2009: 11th International Conference on Computer Modelling and Simulation*, pages 449–454, Washington, DC, USA, 2009. IEEE Computer Society. 18

[58] C. E. Scheidegger, J. L. D. Comba, and R. D. Da Cunha. **Practical CFD Simulations on Programmable Graphics Hardware using SMAC**. *Computer Graphics Forum*, **24**:715–728, 2005. 18

[59] J. M. Cohen, , and J. Molemake. **A Fast Double Precision CFD Code Using CUDA**. In 21st *International Conference on Parallel Computational Fluid Dynamics (ParCFD2009)*, 2009. 18, 19

[60] A.A. Amsden and F.H. Harlow. **The SMAC method: a numerical technique for calculating incompressible flows**. Technical Report LA-4370, Los Alamos National Laboratory, 1970. 19

[61] T. Brandvik and G. Pullan. **Acceleration of a two-dimensional Euler flow solver using commodity graphics hardware**. *Journal Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, **221**(12):1745–1748, 2007. 19

[62] T. Brandvik and G. Pullan. **Acceleration of a 3D Euler Solver using Commodity Graphics Hardware**. In 46th *AIAA Aerospace Sciences Meeting*, number AIAA-2008-607, Reno, NV, January 2008. American Institute of Aeronautics and Astronautics. 19

[63] E. Elsen, P. LeGresley, and E. Darve. **Large calculation of the flow over a hypersonic vehicle using a GPU**. *J. Comput. Phys.*, **227**:10148–10161, December 2008. 19

[64] A. S. Antoniou, Karantasis K. I., E. D. Polychronopoulos, and J. A. Ekaterinaris. **Acceleration of a Finite-Difference WENO Scheme for Large-Scale Simulations on Many-Core Architectures**. In *Proceedings of 48th AIAA Aerospace Science Meeting*, number AIAA-2010-0525, Orlando, FL (2010), January 2010. American Institute of Aeronautics and Astronautics. 19

[65] M. Griebel and P. Zaspel. **A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations**. *Computer Science - Research and Development*, **25**:65–73, 2010. 19

[66] T. Hagen, K. A. Lie, and J. Natvig. **Solving the Euler Equations on Graphics Processing Units**. In Vassil Alexandrov, Geert van Albada, Peter Sloot, and Jack Dongarra, editors, *Computational Science ICCS 2006*, **3994** of *Lecture Notes in Computer Science*, pages 220–227. Springer Berlin / Heidelberg, 2006. 19, 85

[67] D. Levy, G. Puppo, and G. Russo. **A fourth-order central WENO scheme for multidimensional hyperbolic systems of conservation law**. *SIAM J. Sci. Comput.*, **24**:480–506, 2002. 19

[68] S. Y. Chou and D. Baganoff. **Kinetic flux-vector splitting for the Navier-Stokes equations**. *Journal of Computational Physics*, **130**:217–230, 1997. 20, 46, 48, 49, 57, 64, 204

[69] R. J. Le Vecque. *Numerical methods for conservation laws*. Lectures in Mathematics. Birkhauser-Verlag, Basel, 1990. 23

[70] B. Gustafsson. **The convergence rate for difference approximations to mixed initial boundary value problems**. *Math. Comput.*, **29**:396–406, 1975. 29

[71] A. Harten. **High Resolution Schemes for Hyperbolic Conservation Laws,**. *Journal of Computational Physics*, **49**(3):357–393, 1983. 30

[72] G. H. Tu, X. J. Yuan, Z. Q. Xia, and Z. Hu. **A class of compact upwind TVD difference schemes**. *Applied Mathematics and Mechanics*, **27**:765–772, 2006. 30, 31, 32, 93

[73] S. Chapman and T. G. Cowling. *The mathematical theory of non-uniform gases*. Cambridge University Press, Cambridge, 1960. 38, 49, 55

[74] Kevin W. Thompson. **Time dependent boundary conditions for hyperbolic systems**. *Journal of Computational Physics*, **68**(1):1–24, 1987. 43

[75] T. J. Poinsot and S. K. Lelef. **Boundary conditions for direct simulations of compressible viscous flows**. *Journal of Computational Physics*, **101**(1):104–129, 1992. 43

[76] B. Landmann, A. Haselbacher, J. Chao, and C. Yu. **Characteristic boundary conditions for compressible viscous flows on curvilinear grids**. In *Proceedings of 48th AIAA Aerospace Science Meeting*, number AIAA-2010-1084, Orlando, FL (2010), January 2010. 43

[77] D. Gaitonde and M. Visbal. **High-order schemes for Navier-Stokes equations: algorithm and implementation into FDL3D**. Technical Report AFRL-VA-WP-RT-1998-3060, Air Force Research Laboratory, Write-Patterson Air Force Base, 1998. 43

[78] X. Zhong. **High-order finite-difference schemes for numerical simulation of hypersonic boundary layer transition**. *Journal of Computational Physics*, **144**:662–709, 1998. 47, 60

[79] H. Harold Grad. **On the kinetic theory of rarefied gases**. *Communications on Pure and Applied Mathematics*, **2**:331–407, 1949. 49

[80] G. A. Bird. *Molecular gas dynamics and the direct simulation of gas flows*. Clarendon Press, Oxford, 1994. 49, 56

[81] S. Y. Chou. *Kinetic flux-vector splitting for the Navier-Stokes equations*. PhD thesis, Stanford University, 1995. 56

[82] G. N. Patterson. *Molecular flow of gases*. Wiley, New York, 1956. 57

[83] D. I. Pullin. **Direct simulation methods for compressible inviscid ideal-gas flows**. *Journal of Computational Physics*, **34**:231–244, 1980. 57

[84] J. C. Mandal and S. M. Deshpande. **Kinetic flux vector splitting for Euler equations**. *Computer and Fluids*, **23**:447–478, 1994. 57

[85] E. R. Mallet and J. C. Pullin. **Numerical study of hypersonic leeward flow over a blunt nosed delta wing**. *AIAA Journal*, **33**:1626–1633, 1995. 57

[86] Yao Zhang, Jonathan Cohen, and John D. Owens. **Fast tridiagonal solvers on the GPU**. *SIGPLAN Not.*, **45**:127–136, January 2010. 86

[87] Dominik Goddeke and Robert Strzodka. **Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed-Precision Multigrid**. *IEEE Trans. Parallel Distrib. Syst.*, **22**:22–32, January 2011. 86

[88] Peter D. Lax. **Weak solutions of nonlinear hyperbolic equations and their numerical computation**. *Communications on Pure and Applied Mathematics*, **7**(1):159–193, 1954. 94

[89] J. J. Quirk. **A contribution to the great Riemann solvers debate**. *International Journal of Numerical Methods in Fluids*, **18**:115–173, 1994. 104

[90] G. Ben-Dor. **Analytical solution of double-Mach reflections**. *AIAA Journal*, **18**:1036–1043, 1980. 111

[91] P. A. Lagerstrom. *Laminar flow theory.* Princeton University Press, Princeton, 1964. 120, 121

[92] E. R. Van Driest. **Investigation of boundary layer in compressible fluids using the Crocco method**. Technical Report TN 2597, NASA, 1952. 126

[93] L. Crocco. **Sulla trasmissione del calore da una lamina piana a un fluido scorrente ad alta velocità**. *L'Aerotecnica*, **XII**:181–197, 1932. 126

[94] Edgar Katzer. **On the lengthscales of laminar shock/boundary-layer interaction**. *Journal of Fluid Mechanics*, **206**:477–496, 1989. 133, 144, 205

[95] Bono Wasistho. *Spatial Direct Numerical Simulation of Compressible Boundary Layer flow.* PhD thesis, Universiteit Twente, Enschede, December 1997. 133, 144, 205

[96] W. S. Don and C. B. Quillen. **Numerical simulation of shock-Cylinder interactions I. Resolution**. *Journal of Computational Physics*, **122**:244–265, 1995. 146, 157

[97] M. Nordén, H. Sverker, and M. Thuné. **OpenMP versus MPI for PDE solvers based on regular sparse numerical operators**. *Future Generation Computer Systems*, **22**:194–203, 2006. 175

[98] David H. Rudy, James L. Thomas, and Ajay Kumar. **Computation of laminar viscous-inviscid interactions in high-speed internal flows**. In *Computational Fluid Dynamics Symposium on Aeropropulsion*, pages 473–486, NASA Lewis Research Centre, January 1991. 176, 186