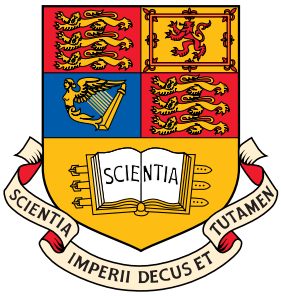
Fourier Spectral/*hp* Element Method: Investigation of Time-Stepping and Parallelisation Strategies

by

Alessandro Bolis



Imperial College London

Department of Aeronautics

This thesis is submitted for the degree of Doctor of Philosophy

and the Diploma of Imperial College London

2013

Declaration

This is to certify that the work presented in this thesis has been carried out at Imperial College London, and has not been previously submitted to any other university or technical institution for a degree or award. The thesis comprises only my original work, except where due acknowledgement is made in the text.

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.

Alessandro Bolis

Acknowledgements

After four years spent at Imperial doing my PhD I have quite a few people to thank for the help and the support I received. However, it is quite hard to recall everybody when looking back to this period. Therefore my apologies to the ones I may have forgotten in the following; these acknowledgements are for you too.

I would like to start expressing my deepest gratitude to my supervisor, Prof. Spencer Sherwin. Not just for the opportunity he gave me but also because he has patiently guided me throughout my whole research path. I would also like to thank Prof. Mike Kirby from Utah. I have actually met him in person just twice but the discussions I had with him and his willingness have been extremely useful and much appreciated.

I have been collaborating with a lot of people within the *Nektar++* group in the last four years. I would like to remember and thank some of them: Gabriele, Dave, Andrew, Julien, Cristian and Peter. I have really enjoyed working with you guys and thank you for the help, even when it simply consisted in a pint together. There is a person which requires a special note in between the *Nektar++* people, Chris Cantwell. I own him a lot. Since the very beginning of my PhD he has helped and supported me, sharing all the troubles I had to face while fighting with the code. Thanks Chris for everything, your help has been precious and fundamental. Finally I would like to thank my office mates David and Rushen for making my first years at Imperial so enjoyable.

I need also have to give some credits to the italian group of friends here in London, to my ex-flatmate Gui and to the old friends back in Italy namely “Il Prandi”, Tom, Dvd and Briz. You guys have been quite good in making me forget the frustration you go through during a PhD. Your contribution won’t be forgotten.

To conclude I really want to thank my family and in particular my parents that have been, as always, my invisible and principal sustain. Grazie mamma e grazie papa’.

Abstract

As computer hardware has evolved, the time required to perform numerical simulations has reduced, allowing investigations of a wide range of new problems. This thesis focuses on algorithm optimisation, to minimise run-time, when solving the incompressible Navier-Stokes equations. Aspects affecting performance related to the discretisation and algorithm parallelisation are investigated in the context of high-order methods. The roles played by numerical approximations and computational strategies are highlighted and it is recognised that a versatile implementation provides additional benefits, allowing an *ad-hoc* selection of techniques to fit the needs of heterogeneous computing environments.

We initially describe the building blocks of a spectral/*hp* element and pure spectral method and how they can be encapsulated and combined to create a 3D discretisation, the Fourier spectral/*hp* element method. Time-stepping strategies are also described and encapsulated in a flexible framework based on the *General Linear Method*. After implementing and validating an incompressible Navier-Stokes solver, two canonical turbulent flows are analysed.

Afterward a 2D hyperbolic equation is considered to investigate the efficiency of low- and high-order methods when discretising the spatial and temporal derivatives. We perform parametric studies, monitoring accuracy and CPU-time for different numerical approximations. We identify optimal discretisations, demonstrating that high-order methods are the computationally fastest approach to attain a desired accuracy for this problem.

Following the same philosophy, we investigate the benefits of using a hybrid parallel implementation. The *message passing model* is introduced to parallelise different kernels of an incompressible Navier-Stokes solver. Monitoring the parallel performance of these strategies the most efficient approach is highlighted. We also demonstrate that hybrid parallel solutions can be used to significantly extend the strong scalability limit and support greater parallelism.

Ai miei genitori

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computational Fluid Dynamics (CFD) is employed in many fields, such as engineering, physics and even medicine (biomedical flows). Since CFD began, one of the major challenges has been to push forward the limit of accuracy, efficiency and speed of flow simulations. This translates in having the capabilities to study more complex flows and in a greater detail; as also stated from $Orszag$ and $Israeli$ in their seminal paper (Orszag & Israeli 1974). Given the vast range of CFD applications, we can easily sense how any improvement in the methods or further understanding of the issues would be beneficial for many users. The complexity of the problem requires a thorough investigation of the computations from different angles, such as the numerical methods adopted, the algorithm design and the parallelisation approaches. Moreover, given the pace at which new computers (and super-computers) develop, a continuous effort is required to keep the algorithms up-to-date and to exploit all the benefits coming from hardware innovation (Feitelson 1999, Meuer et al. 2013).

Over the past five decades many researchers, inside and outside academia, have implemented a large variety of approaches to achieve those goals, some of which are reported and described later in this thesis. Investigations into different numerical schemes, parallelisation paradigms and algorithm efficiency have been fundamental to push the limits forward. The aim of this thesis is to build on these investigations, focusing on spectral/*hp* element and spectral methods. The final goal is to enhance the level of understanding and provide some guidelines on how to increase the global efficiency of a CFD code.

The philosophy that drives the investigations presented in this thesis derives from our firm belief that implementation flexibility can be used to promote algorithms efficiency. From our perspective, efficiency can be defined as the capability of obtaining a numerical result, characterised by desired properties, in the quickest way on a specific machine. Many variables play a role in defining the virtues of a computation. Acknowledging the wide variety of CFD applications, we can immediately sense the impossibility to realise a universal numerical approach able to optimally perform in all the possible scenarios. Generally speaking, different methods may be appropriate in different situations, suggesting that a versatile and modular implementation can supply CFD practitioners with a useful toolbox. In fact, an optimal code can be obtained through an *ad hoc* composition of appropriate techniques for a given set of requirements. The greater the flexibility, the greater the algorithm tuning capabilities, and the higher the chances of fitting the most suitable numerical approach to the problem. A sensible usage of specialised routines can help, for example

- to reach a desired level of accuracy on the final solution minimising the run-time;
- to exploit the capabilities of different machines, reducing the time required for a simulation and promoting effective portability of the code across architectures.

The accuracy of a numerical simulation generally depends on the numerical approximation of our equations and on the complexity of the problem we are studying (geometry, nature, unsteadiness, etc.). Improving the accuracy has the obvious effect of reducing the numerical error associated with the simulation. However, if we look at it from a different perspective, a deeper understanding of how the error can be reduced could also provide a tool to tune our methods to attain a specific desired accuracy reducing the computational time (different applications require different accuracies). In order to reach both high and low levels of accuracy a classical approach is to adopt *high-order numerical methods* to discretise our equations. When talking about numerical techniques we need to make a first distinction between the temporal discretisation methods and the spatial discretisation methods.

The temporal discretisation typically consists in the well known multi-stage and multi-step methods (Butcher 1987, Ascher et al. 1995). Both these families can span various level of accuracies, different orders and they can also be represented using a unique matrix notation (Butcher 2006). The spatial discretisation can also be accomplished with a variety of strategies. High-order methods for the spatial discretisation of PDEs are nowadays commonly applied. The basic spatial discretisation techniques which are classified as high-order methods are:

- the spectral method (Gottlieb & Orszag 1977);
- the high-order finite element method (Szabó & Babuška 1991);
- the high-order finite difference method (Collatz 1966).

Starting form these three fundamental approaches many other high-order techniques have been derived, which can be seen as specialisations of the mentioned methods. Some examples are the spectral/*hp* element method, the high-order finite volume method, the spectral difference method and the more recent flux-reconstruction technique (Williams et al. 2013). In contrast with their low-order counterparts, high-order methods generally take advantage of a larger set of degrees of freedom to approximate the solution and/or to build the related spatial operators. Methods such has the high-order finite difference technique construct the required spatial derivatives approximation in a point of the grid using a large number of adjacent points. When moving to spectral methods and high-order finite element/volume methods the solution is built as a combination of functions (usually called trial functions or expansion basis). If we use many functions and we properly select them (such that they can properly represent our solution) we can sensibly increase the level of accuracy of the numerical representation of the mathematical model we are discretising. The shape and complexity of these functions is dictated by the number of points used to represent them (we can not represent complex functions with a limited number of points). Therefore, increasing the number of points accounted for the representation of these functions allows the usage of more complex/appropriate functions which can approximate the solution more accurately.

In this thesis we focus on the spectral/*hp* element and spectral method only. Both

the spectral/*hp* element and spectral method have been widely applied for spatially discretise the partial differential equations typical of fluid dynamics (Canuto et al. 2007, Karniadakis & Sherwin 2005). The spectral method approximates the solution via global functions (usually Fourier series) and it has been intensively employed for the study of isotropic and homogeneous turbulence. Commonly implemented in combination with a collocation projection[1], it was described in detail by $Gottlieb$ and $Orszag$ (Gottlieb & Orszag 1977). Although this method provides, for smooth solution, an exponential convergence, it does not look attractive for problems characterised by complex geometries. On the other hand the spectral/*hp* element method combines the geometric flexibility of classical finite element techniques with the high-order convergence features of spectral methods (Karniadakis & Sherwin 2005). This technique utilises a polynomial expansion of order P to approximate the solution on a collection of elements. Applied to incompressible flows since $Patera$'s pioneering work in 1984 (Patera 1984), the approach is now also employed for modelling compressible flows in combination with discontinuous Galerkin projections (Warburton et al. 1999, Eskilsson 2005, Hesthaven & Warburton 2008). A detailed description of these methods will be provided in Chapters 2 and 3.

With the rise and development of super-computers, a key aspect of numerical simulations has become their scalability aptitude. When solving a problem using some numerical methods on a multi-core machine, we define our algorithm scalable if we can reduce the computational time proportionally to the number of cores we are using. Hence, we can theoretically solve a specific problem n times faster using n cores (linear scaling).

An algorithm can be characterised by weak or strong scalability. The former is the ability of the algorithm to keep on scaling if the number of cores grows together with the number of degrees of freedom. This is a typical feature for CFD codes, which take advantage of larger machines to study more complex flows, $e.g.$ higher Reynolds numbers. On the other hand strong scalability is the capability of the algorithm to scale even if we do not increase the problem size. Although weak scalability is beneficial for real CFD applications, it may not always be of practical interest. In this thesis we will focus on algorithmic solutions to achieve strong scalability, under the assumption that weak

[1]Spectral approximations combined with a Galerkin projection are however often implemented.

scalability is a natural consequence of its strong counterpart.

Determining and predicting the efficiency and the scalability of a parallel algorithm is not straightforward. In 1993 $Grama\ et\ al.$ provided an extensive and detailed review on practical methodologies to quantify parallel features (Grama et al. 1993). In the context of parallel computing the definition of efficiency may be ambiguous. The widely accepted definition of efficiency for a parallel algorithm is the ratio between the speed-up and the number of cores involved in the simulation (where the speed-up is the ratio between the time to perform the simulation with one core and n cores). While embracing and using this definition later on in this thesis we also keep in mind our general view on efficiency, $i.e.$ the most convenient combination of algorithms to maximise the performances of our simulations. There are many variables playing a role in determining the virtues of a parallel algorithm. Practically, the real efficiency derives from the interactions between many factors, such as

- the problem features (degrees of freedoms, boundary conditions, dimensionality, physic nature, etc.);
- the machine specifications (latency, bandwidth, cores speed, memory, caches, network topology, etc.);
- the numerical algorithm of interest;
- the libraries employed.

In addition we can approach the parallelisation of the numerical algorithm using different paradigms. The *message passing model* is the most widely used and it is the one we will focus on. An alternative would be to introduce parallelism via other paradigms, such as the *shared memory model*, which consists of multithreading strategies for CPUs (Chapman et al. 2007), GPUs (Khronos OpenCL Working Group 2008, Sanders & Kandrot 2010) and even combinations of both.

1.1 Objectives and Motivations

When solving the equations typical of fluid dynamics, the level of accuracy on the solution and the computational efficiency are fundamental aspects. In the last five decades many CFD practitioners moved to the usage of high-order methods to spatially discretise their equations. While high-order methods provide a solid base to improve numerical accuracy they also introduce some disadvantages, especially when they are coupled with explicit time-integration schemes for the solution of unsteady problems. In fact, together with the spatial accuracy, also the number of operations increases and the numerical stability constraints become more stringent (Karniadakis & Sherwin 2005). As we mentioned in the previous section, we will focus on the spectral and the spectral/*hp* element only, although some of the following considerations could be applied to other high-order methods, as can be seen in (Liang et al. 2013).

Numerical stability restrictions in CFD algorithms generally arise when explicitly time-marching the non-linear terms appearing in the Navier-Stokes equations (incompressible and compressible) and in the Euler equations (Hirsch 2007). Given the complexity of these equations, basic numerical investigations are generally carried out on simplified convective-dominated problems, such as the unsteady linear-advection equation. The common understanding is that stability restrictions for high-order methods become rapidly more stringent as the basis expansion order increases. This is because the CFL condition imposes that the eigenspectrum of the derived spatial operator must lie within the time-integration scheme stability region. Since the magnitude of the eigenvalues amplifies algebraically with the polynomial order, the maximum applicable time-step needs to be proportionally reduce to rescale (enlarge) the time-stepping stability region. Intuition suggests that seeking accuracy by increasing the expansion order may become impractical. In fact the number of time-steps required to reach a specific time-level does substantially increase.

In combination with a discontinuos Galerkin (DG) projection, the spectral/*hp* method has been widely used for the solution of hyperbolic equations. Initially proposed by $Reed$ and $Hill$ (Reed & Hill 1973) for solving neutron transport problems, it gained great popularity because of its capability of preserving phase and amplitude information

throughout time-integration, as demonstrated by $Sherwin$ (Sherwin 2000), by $Ainsworth$ in a series of papers (Ainsworth 2004*b*,*a*, Ainsworth et al. 2006) and by $De\ Basabe$ (De Basabe et al. 2008, De Basabe & Sen 2010). The numerical properties of DG spectral/*hp* element methods for hyperbolic equation solutions have been investigated by many authors as (Peterson 1991), (Cockburn & Shu 1998, 2001), (Hu & Atkins 2002), (Warburton & Hagstrom 2008) and (Hesthaven & Warburton 2008).

The numerical stability properties of high-order methods when coupled with an explicit time-integration scheme have been deeply investigated and clarified (Zhang & Shu 2010, Antonietti et al. 2012). Given that the interactions between the spatial and the temporal discretisation are well-known, the research is mainly focused on producing *ad hoc* numerical strategies which can preserve high accuracy while alleviating the stability constraints. Examples of these efforts are quite common in literature and many approaches have been followed, such as:

- Construction of tailored multi-stage time-integration schemes which show suitable stability regions for the problem of interest (Cockburn & Shu 1998, 2001, Gottlieb et al. 2001).

- Introduction of specialised routines which can alleviate the stability constraints, such as basis with a variable expansion order (Dumbser et al. 2007) or sub-stepping procedures (Lörcher et al. 2008).

While the numerical properties of DG spectral/*hp* methods for sufficiently smooth solutions are now widely recognised and analytiacally understood (Zhang & Shu 2010, Antonietti et al. 2012), the choice of discretisation parameters to achieve a given numerical error in the most computationally efficient manner are not as effectively clear. Explicit multi-stage schemes, such as Runge-Kutta methods, have been widely used (Cockburn & Shu 1998, 2001). However, they require multiple evaluations of the spatial operator at each time-step. Hence, even if alleviating the stability constraints and allowing the usage of a bigger time-step, they could not be the most efficient strategy. This last consideration becomes even more important when spatial accuracy is enhanced via an increment in the expansion basis order (as usually done for the spectral/*hp* element method). This is because the spatial operator size grows algebraically with the expansion order. As a

consequence, even if we can predict the stability constraints and the final accuracy, it is not evident how to achieve optimal computational performance.

We therefore intend to analyse the actual computational load when using the spectral/*hp* element method coupled with both explict multi-step and multi-stage schemes. The test case for our investigations will be a 2D unsteady linear-advection equation, since it is representative of the problems of interest. The goal is to highlight what are the discretisations parameters which minimise the CPU time for the solution of our test case. The idea is to highlight limitations and benefits of using the spectral/*hp* element method from a practical (computational) point of view.

Our objectives in this context are:

- Map the actual CFL restrictions as a functions of the polynomial order, the mesh size, the mesh nature (uniform/non-uniform) and the time-integration scheme adopted.
- Identify the computational efficiency trend for explicit multi-stage and multi-step schemes.
- Identify the computational efficiency trend with respect to the polynomial expansion order. We span low-order (Finite Element Method) and high-order methods (spectral/*hp* element method).
- Quantify the real accuracy on the final solution depending on the spatial/temporal discretisation and the final time.
- Provide some guidelines on what is the optimal combination of spatial/temporal discretisation to attain a desired accuracy on the solution while minimising the computational time.

So far we considered the efficiency of a simulation from the numerical methods perspective. In practical applications, due to the elevate number of operations required, parallel computing is not optional, but a real need. The issues arising when parallelising a CFD code are often an obstacle for the realisation of a performing software. When introducing parallelism in a serial algorithm, the main difficulty is to produce a parallel version of it

that can scale properly (strongly) on many cores, on different architectures and eventually can be ready to be used on parallel machines which are in development. While many solutions and guidelines are nowadays available (Karniadakis & Kirby 2003), it is often not clear what is the best approach to follow in the various scenarios typical of CFD applications. The basic research in this field is oriented in developing and optimising libraries which can be coupled to many softwares and used to enhance the level of parallelism. These libraries are based on different parallelisation paradigms, namely:

- the message passing model (Gabriel et al. 2004);
- the shared memory model for CPUs (Nichols et al. 1996, Chapman et al. 2007);
- the shared memory model for GPUs (Sanders & Kandrot 2010, Khronos OpenCL Working Group 2008).

Once the library and the parallelisation paradigm have been selected, the common practice is to optimally introduce the parallelism in the code. While the shared memory model for GPUs and CPUs has gained great popularity in last five years (Canstonguay et al. 2011), the message passing model is still the most common approach for CFD applications, especially when using numerical methods that involve an elemental discretisation. In fact, to introduce the shared memory model would require careful considerations about memory layout, especially in case we want to use GPUs. As a result, many CFD practitioners find more convenient to apply the message passing model after the serial version of the code has been realised, thus avoiding a proper design of the memory management routines. Moreover, the new-generation super-computers are generally characterised by an elevate number of nodes supplied with local memory, making the shared memory model less attractive and the message passing model the natural choice (Meuer et al. 2013).

There is a vast literature on parallelisation approaches using the message passing model. Usually researchers provide the details of their implementation, showing how it can be used to run simulations on an increasing number of processors and on various machines (Tufo & Fischer 2001, Takahashi 2003, Chan et al. 2008). The parallelisation of the spectral/*hp* element method has been intensively investigated during the last two decades (Fischer & Rønquist 1994, Fischer et al. 2008). The general approach is to implement an elemental decomposition sending different elements (or groups of elements) to

different processors. This approach requires communication between elements which are physically adjacent but they have been sent to different processors. Therefore the research in this case is directed to the mesh decomposition optimisation for parallel applications (Karypis 2013).

When a pure spectral method is involved in the discretisation, as in the Fourier spectral/*hp* element method, a modal decomposition based on the Fourier expansion orthogonality is also possible (Crawford et al. 1996). The message passing model can be used to share computations among processors sending different Fourier modes to different processors. Communication in this case takes place when global operations in the pure spectral direction are required, $i.e.$ when a FFT is required to move variables from a transformed to a non-transformed Fourier space and vice-versa. In this context the research effort has been directed to enhance the scalability of the parallel FFT algorithm (Chan et al. 2008, Li & Laizet 2010).

While the two standalone approaches (the elemental and the modal decomposition) have been deeply investigated separately, they have been rarely applied concurrently. In case of a hybrid spatial discretisations such as the Fourier spectral/*hp* element method, both the parallel techniques can be applied. However, it is often unpractical to realise an implementation which can easily introduce both approaches. A first attempt has been reported in (Hamman et al. 2007). They used a simplified numerical technique, where a 1D spectral/*hp* element method was combined with a 2D Fourier spectral method. This numerical method is very useful for flows exhibiting a periodic behaviour in two spatial directions. However, it is not applicable to complex CFD applications. Their implementation is based on a MPI cartesian virtual topology which assigns a different parallel technique to each cartesian coordinate. The results presented in (Hamman et al. 2007) clearly suggest that a flexible parallel implementation can promote parallel efficiency and can be used to extend the common parallelisation limits.

Following this last remark, we direct our efforts to the implementation of a flexible parallel algorithm based on the message passing model and the related MPI library (Gabriel et al. 2004). The idea is to extend the approach presented in (Hamman et al. 2007) to more complex CFD scenarios, thus using a 2D spectral/*hp* element method combined with a 1D Fourier expansion. Using this numerical technique, we remove the con-

straint on the periodicity of the flow in one of the spatial directions, allowing investigations of further CFD problems, such the flow past a bluff body (cylinders, airfoils, etc.). We will introduce the two standalone parallel techniques taking advantage of well-known algorithms. However, the encapsulation of the concept of parallelisation in a cascade of C++ classes will allow an extremely flexible usage of the parallel technique (separately or concurrently). The final goal is to introduce a parallelisation methodology (based on flexibility) which can be useful to capitalise our algorithms while changing the nature of the problems we are investigating and the parallel machine we are using. Furthermore, a mixed parallel approach able to enhance scalability is a first step toward a parallel software that can exploit the capabilities of new super-computers. In fact, while the standalone techniques are generally optimised to scale up to the limit of the current facilities, they are generally not ready to strongly scale on new super-computers (just weak scaling). However, combining them can provide a tool to exploit the capabilities of a new machine while their optimisation process continues. Our objectives in this context are:

- Provide practical guidelines on how to parallelise a CFD code, presenting in details the advantages and disadvantages of using two canonical approaches (namely the elemental decomposition approach and the modal decomposition approach).
- Illustrate how we can parallelise our code considering both approaches concurrently using MPI virtual topologies.
- Show how to couple the two parallel techniques to obtain a flexible and hybrid parallelisation approach.
- Investigate different types of discretisation to identify if a specific parallel approach is more appropriate than another. Therefore showing that having many parallel techniques readily available in the code can promote parallelisation efficiency.
- Demonstrate that a sensible combination of parallel techniques can be used to extend the scalability limits of our code and can also be used to recover parallel efficiency.
- Identify possible algorithmic solutions that can promote overall efficiency ($e.g.$ comparison between iterative and direct solvers for the solution of linear systems).

- Elaborate a preliminary scalability model to predict the parallel features of the mixed parallelisation approach.

In addition, while presenting our investigations, we propose some implementation solutions to address typical problems when developing a CFD code (Kirby & Sherwin 2006*b*). As a continuation of the work presented in (Vos 2010), we show how to achieve higher level of flexibility when implementing the building-blocks of a software that solves PDEs using the spectral and the spectral/*hp* element method. Although the numerical methods introduced are well-known (Karniadakis & Sherwin 2005, Butcher 2006), we illustrate a sensible approach for their implementation and their efficient coupling. In fact, as anticipated, one of the objectives of this thesis is also to provide useful guidelines and suggestions to other CFD practitioners on how to address implementation issues.

1.1.1 *Nektar++* project

The development and the investigation of numerical methods and implementation strategies is the leading topic throughout the whole thesis. In order to examine all the possible combinations of time-integration schemes, spatial discretisations and parallelisation approaches, a flexible implementation is required, making algorithm design a key point. The algorithm should be able to embrace a range of time-integration schemes and it should provide a high level of flexibility for the spatial discretisation, to facilitate parametric studies. In addition, the ability to tune the parallelisation approach to suit specific problems and hardware features is considered fundamental to enhancing the effective performance of the code.

Nektar++ framework has been designed and developed in the past six years to accomplish these tasks by having implementation flexibility as the driving philosophy. *Nektar++* is an open source software library in development at Imperial College London (Department of Aeronautics) in collaboration with University of Utah (School of Computing). The work reported in this thesis takes advantage of this and at the same time contributed to the ongoing *Nektar++* project (Kirby & Sherwin 2006*b*). In fact, the in-

vestigations we present in the rest of this thesis have been performed during the continued development of the *Nektar++* project.

Since *Nektar++* is an ongoing project, not all the required features were available in the library when the research presented in this thesis started. Therefore, time has been dedicated to introduced all the algorithms and C++ classes necessary for our investigations. In order to perform computational efficiency investigations when time-stepping PDEs, the following work on the code has been done:

- Development, debugging, validation and profiling of 2D spatial operators for both continuous and discontinuous Galerkin projections. Special attention has been given to the DG projection to optimise calculation of boundary fluxes. A C++ class has been designed to handle memory access on the boundaries in order to obtain a fair comparison between low and high-order spatial discretisations.

- Development, debugging and validation of the time-integration class. After the time-stepping procedure has been fixed and validated some other time-integration schemes have been added and special attention has been given to implicit-explicit methods and to the handling of time-dependent boundary conditions.

- Introduction of appropriate timing routines in the code to sample computational time.

- Implementation of an advection and an advection-diffusion solver able to use *Nektar++* library (also Laplace, Poisson and Helmholtz solvers have been implemented as middle steps in the development procedure).

- Implementation of a precise CFL calculator for the problem of interest and of a general approximate CFL calculator for other practical purposes.

In order to investigate flexible parallelisation methodologies for the solution of incompressible flows, the following implementation steps were required:

- Implementation of a Fourier spectral/*hp* element method which allows different combinations of a pure spectral method and the spectral/*hp* element method to solve various types of PDE. This include also the encapsulation of basic operations (FFT, dealiasing, etc) in *Nektar++* classes and the production of various example cases.

- Development, debugging, validation and profiling of an incompressible flow solver based on the projection method of (Karniadakis et al. 1991) for 2D and 3D problems.

- Parallelisation of the code using MPI, encapsulating the concept of parallelisation in C++ classes and implementing specific C++ objects to handle the data transposition when parallelising in the pure spectral directions.

Other extra-tasks were undertaken in the development process, such as:

- Code documentation for users and future developers (Kirby & Sherwin 2006*b*).

- Code restructuring to properly introduce in the code various features, such as different treatments of the convective term, sub-stepping procedure for the Navier-Stokes equations, etc.

- Development of post-processing utilities to visualise *Nektar++* results on well-known visualisation softwares.

1.2 Outline

In Chapter 2 we provide a description of the numerical methods involved in our studies. Although some of the techniques are generally applicable to a plethora of numerical discretisations, the Chapter focuses on high-order methods. Specifically we will highlight the building blocks of a spectral/*hp* element method and a pure spectral method. Subsequently, combining these approaches, we will introduce the Fourier spectral/*hp* element method which we will employ to solve three dimensional incompressible flows. After presenting the time discretisation methods we will focus on the solution the 3D Incompressible Navier-Stokes equations with special attention to turbulent flows.

In Chapter 3 we will present our investigations on optimal time-stepping strategies for low- and high-order methods. As anticipated, we consider an hyperbolic equation, namely a 2D unsteady linear advection problem. The selected test case is of relevance for many CFD applications. In fact it can be seen as a simplified version of the explicit convective

term used in many solution processes for both compressible and incompressible flows. Taking advantage of *Nektar++* flexibility we will perform a series of parametric simulations. Varying the temporal and spatial discretisation and carefully considering numerical stability, we identify the most efficient combination of time-stepping and spectral/*hp* element discretisation for a given accuracy.

Chapter 4 contains a description of how we can systematically approach the challenge of parallelisation for the Fourier spectral/*hp* element method. We consider various approaches, highlighting their limitations and issues, and we demonstrate how to combine them to achieve greater performance and to extend the strong scalability limit. We reuse the turbulent simulations presented in Chapter 2 as test cases.

Finally in Chapter 5 we summarise the work presented in the other Chapters and discuss our findings. In addition, Appendix A contains a brief description of *Nektar++* structure and in Appendix B some extra details on the time-integration schemes are reported.

1.3 Assumptions

In this thesis we will seek optimal approaches to improve the efficiency of a CFD simulation. While the definition of efficiency may have a number of interpretations for a general algorithm, we will always associate it to the minimisation of the execution time on a fixed number of processors. Considerations about memory usage and limitations are disregarded in this thesis, although we acknowledge they may become an issue in some scenarios and we will make direct reference to memory problems as they arise. However, a detailed investigations of those issues are beyond the scope of this study. In addition, when monitoring run-time, we will always neglect set-up costs. These type of costs generally involve input-output routines, matrix construction, memory allocations, C++ object instantiations, etc. Even if they are not negligible for short simulations, they usually are for real CFD applications. In fact, the solution is generally time-stepped for many time units, thereby reducing the set-up routines to a small percentage of the overall computation.

When introducing an elemental discretisation, whether a standard 2D approach or a

Fourier spectral/*hp* element method, we restrict ourselves to 2D quadrilateral expansions. While triangular or full 3D elemental tessellations may provide some further insights for the computational issues we are investigating, they also introduce additional problems which would make our studies more intricate. Furthermore, the 2D spectral/*hp* element expansions are always assumed to be tensorial and with an identical expansion order in both the coordinate directions.

The investigation of time-stepping strategies which will be presented in Chapter 3 takes advantages of the flexible implementation of *Nektar++* . As a consequence, a large variety of time-stepping schemes could be investigated. We will constrain ourselves to three popular schemes. Essentially we will consider a comparison between the widely used fourth order Runge-Kutta and its lower order version which is the second order Runge-Kutta scheme. In addition we will compare the second order multi-stage scheme with a common multi-step scheme of the same order (Adams-Bashforth). Those schemes are carefully selected to allow a direct comparison between scheme orders (fourth and second) and scheme nature (multi-stage and multi-step).

Solution of practical CFD problems will be performed using the traditional C^0 continuous Galerkin formulation, therefore the simulations presented in Chapters 2 and 4 are the results of this type of approach. However, when investigating time-stepping strategies in Chapter 3, we will introduce a discontinuous Galerkin projection. The reasons for this choice are related to the numerical properties of the weak advection operator. In fact, hyperbolic equations discretised with a continuous Galerkin approach are characterised by purely imaginary eigenvalues. Consequently they would require time-stepping schemes whose stability region encompass the imaginary axis. On the other hand, a discontinuous Galerkin formulation inserts a damping effect, reinforcing the similarities with the convective term treatment when we solve the incompressible Navier-Stokes equations.

The results presented in this thesis have been produced using *Nektar++* during its development process. Although we state for each study the version of the code employed, we recognise that the absolute values we report can not be considered universal, but rather code and version dependent. However, the general philosophy we introduce and the trends we highlight can be beneficial for many CFD practitioners. Our considerations should facilitate other users to take more conscious decisions when approaching the implemen-

tation of a CFD algorithm.

Chapter 2

Numerical Methods

In this chapter we provide an overview of the numerical methods which have been implemented and investigated while developing *Nektar++* . In each section we present the numerical techniques providing a brief theoretical background and a verification of the implementation. Most of the presented methods can be applied to wide variety of PDEs. However, while presenting them, we focus on our final application, which is the solution of an incompressible fluid.

We start by describing the numerical techniques which have been implemented to discretise the spatial operators. In the rest of this thesis we will refer to these techniques as *high-order methods*, which incorporate facets of both the spectral and the spectral/*hp* element method. In the section dedicated to the spatial discretisation methods we reserve a final passage to describe the combination of the two discretisation strategies. We refer to this combined approach as the *Fourier spectral/hp element method*. The second section continues describing the temporal discretisation which has been implemented in *Nektar++* . We provide an initial description of both multi-step and multi-stage methods, although we focus on the implicit-explicit multi-step methods which will be used to solve the Navier-Stokes equations. Subsequently we present the algorithm that has been adopted to solve the incompressible Navier-Stokes equations, where the temporal and

spatial derivatives have been discretised using the methods reported in previous sections. To conclude the chapter we summarise what has been done in terms of implementation and verification, discussing how this will affect the studies reported in the rest of this thesis.

2.1 Spatial Discretisation

The spectral method (SM) has been initially described by $Gottlieb$ and $Orszag$ in their monograph of 1977 (Gottlieb & Orszag 1977). After that, a series of textbooks have been produced by various authors. For an overview of spectral approximations of partial differential equations the reader is referred to the works of $Boyd$ (Boyd 2001) and $Fornberg$ (Fornberg 1996). $Canuto\ et\ al.$ also provided a precise and complete description of the spectral method and its application to fluid dynamics in their well-known textbooks (Canuto et al. 2006, 2007). The spectral/*hp* element method (SEM) can be seen as a combination of the spectral method and the well know finite element method (Szabó & Babuška 1991). Introduced by $Patera$ in 1984 for fluid dynamics applications (Patera 1984), it is currently widely used in engineering. For a full and extensive discussion of the method and its applications to fluid dynamics the reader is referred to the seminal works of $Karniadakis$ and $Sherwin$ (Karniadakis & Sherwin 2005).

We initially present the method of weighted residuals in section 2.1.1 and the Galerkin projection in section 2.1.2, as an initial and common base for both the spectral and spectral/*hp* element method. In section 2.1.3 we provide a general description of the spectral/*hp* element method. Subsequently the spectral method is presented in section 2.1.4, where we provide a detailed description of the expansion basis involved in the solution approximation and the required numerical techniques. The last portion of this section is dedicated to the Fourier spectral/*hp* element method and a brief description and verification of the algorithm.

2.1.1 Weighted Residuals

When approximating a continuous dependent variable we replace it with its mathematical representation, $i.e.$ an infinite expansion with respect to the independent variables as described in Eq. (2.1). The variable $u(\boldsymbol{x})$ is represented via an infinite combination of functions $\phi_n(\boldsymbol{x})$, commonly called trial functions or expansion basis. An infinite expansion could hypothetically satisfy an infinite set of conditions, which we can use to find an infinite number of expansion coefficients $\hat{u}_n$, so that

$$u(\boldsymbol{x}) = \sum_{n=0}^{\infty} \phi_n(\boldsymbol{x})\hat{u}_n. \tag{2.1}$$

To be computationally useful this expansion must be finite, leading to a finite set of conditions we can impose, and also to an error which is the residual between the approximation and exact solution. Assuming a finite expansion with N terms, as reported in Eq. (2.2), we have N conditions to impose to find the N expansion coefficients $\hat{u}_n$.

$$u(\boldsymbol{x}) \approx u^{\delta}(\boldsymbol{x}) = \sum_{n=0}^{N} \phi_n(\boldsymbol{x})\hat{u}_n \tag{2.2}$$

A general linear differential operator $\mathbb{L}(\cdot)$ applied to the variable $u(\boldsymbol{x})$ in the *n*-dimensional domain Ω, such as

$$\mathbb{L}(u(\boldsymbol{x})) = 0, \tag{2.3}$$

represents a common differential equation. If we apply the same operator to the approximate variable $u^{\delta}(\boldsymbol{x})$, we obtain

$$\mathbb{L}(u^{\delta}(\boldsymbol{x})) = R(u^{\delta}(\boldsymbol{x})) \neq 0. \tag{2.4}$$

Moving to the finite approximation $u^{\delta}(\boldsymbol{x})$, as shown in Eq. (2.4), introduces a numerical error that we will call the *residual* $R = R(u^{\delta}(\boldsymbol{x}))$.

Solving the differential problem in Eq. (2.4) translates to finding the numerical values of the expansion coefficients $\hat{u}_n$. There is not a unique way to find the expansion coefficients, but it will depend on the restrictions we want to impose on the residual, $i.e.$ on the N conditions we can impose.

The N conditions on the residual R are imposed in integral form, such as the inner product between the residual and a set of N test functions $v_j(\boldsymbol{x})$ (or weight functions) is

zero

$$(v_j(\boldsymbol{x}), R) = 0 \qquad j = 0, \ldots, N; \tag{2.5}$$

where we define the Legendre inner product of two variables $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$ on the domain Ω as

$$(f(\boldsymbol{x}), g(\boldsymbol{x})) = \int_\Omega f(\boldsymbol{x}) g(\boldsymbol{x}) d\boldsymbol{x}. \tag{2.6}$$

The nature of these conditions, which in turn depends on the selection of the N test functions, will define the type of method and projection we apply. In the case $v_j(\boldsymbol{x}) = \delta(\boldsymbol{x} - \boldsymbol{x}_j)$ we are using a so called *collocation projection*, where δ represents the Dirac delta function. This type of projection is often employed with spectral approximations and finite difference methods. It numerically means that the approximate solution $u^\delta(\boldsymbol{x})$ satisfies the differential equation at a set of j collocation points in Ω, $i.e.$ at these points the residual is zero ($R = 0$). Another widely used projection is the *Galerkin* method, especially for finite element and spectral/hp element methods. This approach can be seen as a residual minimisation over the domain Ω. The best known is the *Bubnov-Galerkin* variant in which $v_j(\boldsymbol{x}) = \phi_j(\boldsymbol{x})$. In the following section a description of the Galerkin projection is provided, as it is the one used for all the results presented in this thesis.

2.1.2 Galerkin Projection

Numerical methods which require an elemental decomposition of the domain Ω where the problem is defined, typically use a Galerkin projection to move from the continuous physical space $u(\boldsymbol{x})$ to the discrete space of coefficients $\hat{u}_n$. Examples of these elemental approaches are the finite element method (FEM) and the spectral/*hp* element method (SEM). However, it is not uncommon to find applications of the Galerkin projection to pure spectral approximations, which alternatively often use collocation projections.

In order to illustrate the basics of a Galerkin projection we consider the linear differential equation for the variable $u(\boldsymbol{x})$ in its strong form

$$\mathbb{L}(u) = f, \tag{2.7}$$

defined over the *n*-dimensional domain Ω and with appropriate boundary conditions on the domain boundaries $\partial\Omega$.

The weak form of the equation can be obtained by multiplying Eq. (2.7) with a test function v, and integrating over the domain Ω, yielding to the problem of finding $u \in \mathcal{U}$, such that

$$\int_\Omega v\mathbb{L}(u)d\boldsymbol{x} = \int_\Omega v f d\boldsymbol{x} \qquad \forall v \in \mathcal{V}. \tag{2.8}$$

$\mathcal{U}$ and $\mathcal{V}$ are the functional spaces where the trial and test functions are respectively defined[1]. $\mathbb{L}(u)$ is a linear differential operator, for example $\mathbb{L}(u) = \nabla^2 u - \lambda u$, where λ is a real positive constant. We integrate by parts the left-hand side term of Eq. (2.8), obtaining

$$\int_\Omega \nabla v \nabla u d\boldsymbol{x} + \int_\Omega \lambda v u d\boldsymbol{x} = \int_\Omega v f d\boldsymbol{x} + \Big[v\nabla u\Big]_{\partial\Omega}. \tag{2.9}$$

Defining the bilinear form $a(u, v)$ as

$$a(u, v) = \int_\Omega \nabla v \nabla u d\boldsymbol{x} + \int_\Omega \lambda v u d\boldsymbol{x} \tag{2.10}$$

and the linear functional as

$$F(f, v) = \int_\Omega v f d\boldsymbol{x} + \Big[v\nabla u\Big]_{\partial\Omega} \tag{2.11}$$

we can rewrite the formulation in its compact version as: find $u \in \mathcal{U}$ such that

$$a(u, v) = F(f, v) \qquad \forall v \in \mathcal{V}. \tag{2.12}$$

As mentioned in section 2.1.1, in practice we solve the problem in a finite subspace of $\mathcal{U}$, which we call $\mathcal{U}^\delta$, where u is approximated by u^δ. Hence, it yields to: find $u^\delta \in \mathcal{U}^\delta$ such that

$$a(u^\delta, v^\delta) = F(f, v^\delta) \qquad \forall v^\delta \in \mathcal{V}^\delta. \tag{2.13}$$

Choosing $\mathcal{U}^\delta = \mathcal{V}^\delta$ and

$$u^\delta = \sum_{n\in\mathcal{N}} \phi_n \hat{u}_n, \tag{2.14}$$

the final formulation can be written as: find $\hat{u}_n$ with $n \in \mathcal{N}$ such that

$$\sum_{n\in\mathcal{N}} \hat{u}_n a(\phi_n, \phi_m) = F(f, \phi_m) \qquad \forall m \in \mathcal{N} \tag{2.15}$$

where $\mathcal{N}$ indicates the number of degrees of freedom. In matrix notation we obtain

$$\boldsymbol{A}^T \hat{\boldsymbol{u}} = \hat{\boldsymbol{f}} \tag{2.16}$$

[1]The commonly used Bubnov-Galerkin approach implies that $\mathcal{U} = \mathcal{V}$.

where

$$\boldsymbol{A}^T[n][m] = a(\phi_n, \phi_m) = \int_\Omega \nabla\phi_m \nabla\phi_n d\boldsymbol{x} + \int_\Omega \lambda\phi_m\phi_n d\boldsymbol{x} \tag{2.17a}$$

$$\hat{\boldsymbol{f}}[m] = \int_\Omega \phi_m f d\boldsymbol{x} + \boldsymbol{\Gamma}, \tag{2.17b}$$

where $\boldsymbol{\Gamma}$ represents the boundary conditions contribution $\Big[v\nabla u\Big]_{\partial\Omega}$.

2.1.3 Spectral/*hp* Element Method

Spectral/*hp* element methods have been introduced by $Patera$ in 1984 (Patera 1984). Although these methods are commonly used in the field of fluid dynamics (Carmo et al. 2011, Sharma et al. 2011), they have also been applied in other engineering areas, such as bio-engineering (Alastruey et al. 2011) and structural engineering (Nogueira Jr. & Bittencourt 2007). This method can be considered as an extension of the more widespread finite element method. It combines the high geometric flexibility, typical of finite element discretizations, with the exponential convergence properties of spectral methods. Classically the finite element method approximates the solution as a series of linear functions on the subdomains (elements) in which the original domain is partitioned. Extending this approach, the spectral/*hp* element method uses a series of high-order polynomials on each subdomain to carry out the solution approximation. Assuming a series of $P+1$ linearly independent polynomials spanning the polynomial space $\mathcal{P}_P$ (where P is the maximum polynomial degree), the error for a sufficiently smooth solution is a function of the mesh-size h and the polynomial degree P , and an expansion u^δ has the property

$$||u - u^\delta|| \leq Ch^P||u|| \approx \mathcal{O}(h^P). \tag{2.18}$$

Eq. (2.18) implies that the error on the solution decreases as we refine the mesh (reduction of h) or as we increase the polynomial degree P.

Because of the high level of accuracy and the ability to discretise complex geometries, the spectral/*hp* element method has been intensively applied to numerous branches of fluid dynamics, such as stability analysis of complex flows, biomedical flow simulations in complicated domains and turbulent simulations. $Rønquist$ in 1988 described in his PhD thesis (Rønquist 1988) all the aspects of a spectral/*hp* element method applied to the solution of the three-dimensional Navier-Stokes equation. He used a conjugated

gradient method and a multi-grid approach in combination with a optimal-order Legendre spectral/*hp* element dicretisation. During the 1990's $Karniadakis$ presented various applications of the spectral/*hp* element method to fluid dynamics for compressible and incompressible flows. We recall his seminal works (Karniadakis 1990) in which he solved the three-dimensional Navier-Stokes equations using a combination of the spectral and spectral/*hp* element method highlighting the geometrical flexibility of this approach. $Sherwin$, while focusing on biomedical flows, extended previous studies to encapsulate triangular domains, unstructured meshes and a complete formulation using a modal basis (Sherwin & Karniadakis 1995). Stability analysis of transient and turbulent flows, especially in complex geometries, has been thoroughly investigated in the last two decades using the spectral/*hp* element method. The works of $Sherwin$, $Blackburn$ and $Barkley$ provide a vast literature reference on the topic. For brevity we just mention one of their joint efforts (Barkley et al. 2007), where they performed a large-scale stability analysis based on a spectral/*hp* element based code. Applications of the method to the solution of turbulent flows have also appeared throughout the literature. Also in this case the available literature is extensive. We recall the work of $McIver\ et\ al.$ (McIver et al. 2000) where a tubulent pipe flow has been resolved using the spectral/*hp* element method and a comparison is provided between the use of Cartesian and cylindrical coordinates.

2.1.3.1 Domain decomposition

The spectral/hp element method follows the finite element approach by decomposing the domain Ω into a series of subdomains (elements) Ω^e so that

$$\Omega = \bigcup_e \Omega^e \tag{2.19a}$$

$$\Omega^e \cap \Omega^k = \emptyset \quad \forall e \neq k. \tag{2.19b}$$

Basic operations, such as differentiation or integration, are carried out on a reference space Ω^{std}, derived from Ω^e using a mapping system. For each element this map transforms the coordinates in physical space $\boldsymbol{x}$ of Ω^e into the reference system coordinates $\boldsymbol{\xi}$. In the case of a two-dimensional domain $\boldsymbol{x} = [x_1, x_2]^T$ the mapping system can be written as $x_1 = \chi_1^e(\xi_1, \xi_2)$ and $x_2 = \chi_2^e(\xi_1, \xi_2)$. Fig. 2.1 depicts a simple interpretation of the

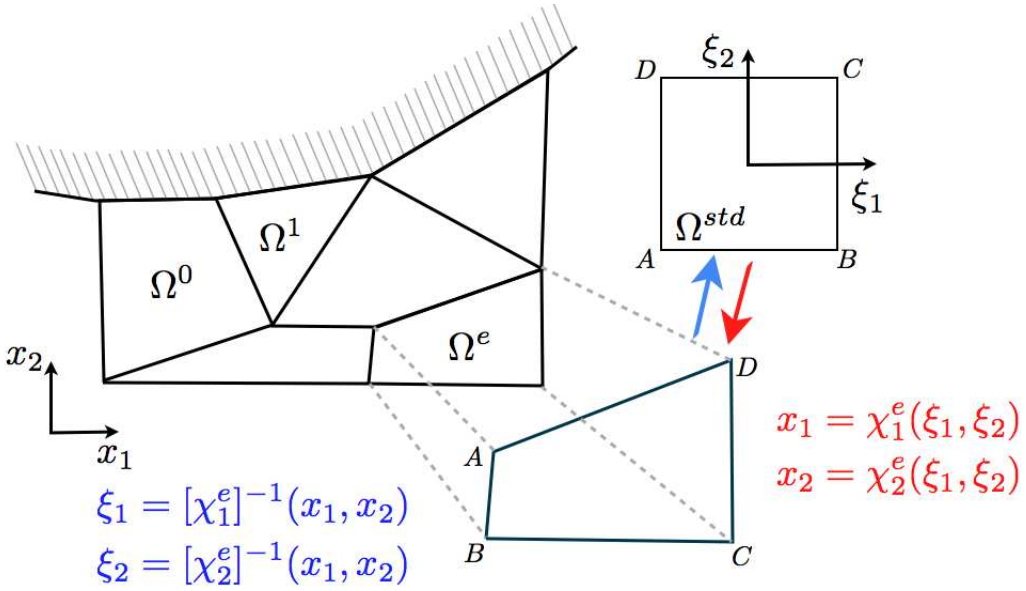


Figure 2.1: Graphical illustration of a mapping system between the real element and the standard element. The mapping is assumed to be invertible.

process. If we consider, for example, a one-dimensional case, we can define a linear mapping $\chi^e : \Omega^e \rightarrow \Omega^{std}$ such that

$$x = \chi^e(\xi) = \frac{1-\xi}{2}x^e + \frac{1+\xi}{2}x^{e+1} \quad \xi \in \Omega^{std} \tag{2.20}$$

and its inverse mapping, which is

$$\xi = [\chi^e]^{-1}(x) = 2\frac{x - x^e}{x^{e+1} - x^e} - 1 \quad x \in \Omega^e \tag{2.21}$$

where $\Omega^{std} = [-1, 1]$ and $\Omega^e = [x^e, x^{e+1}]$. This isoparametric mapping can be easily extended to bi-dimensional elements and it is common practice to use the same expansion used to approximate the variable to build the transformation operators. For bi-dimensional elements for example we have that

$$x_i = \chi_i(\xi_1, \xi_2) = \sum_{n \in \mathcal{N}} \phi_n(\xi_1, \xi_2)\hat{x}_n^i. \tag{2.22}$$

As a result from mapping our elements from a local to a standard system, we can define a spectral/*hp* expansion on each one of the element Ω^e as

$$u(x_1, x_2) = \sum_{n \in \mathcal{N}} \phi_n(\chi_1^{-1}(x_1, x_2), \chi_2^{-1}(x_1, x_2))\hat{u}_n \tag{2.23}$$

where we have dropped the index e to simplify the notation. For non iso-parametric mapping system and extension to curved elements the reader is referred to (Karniadakis & Sherwin 2005) for a full treatment of the topic.

2.1.3.2 Assembly

Once we have decomposed the domain Ω into a set of $\mathcal{E}$ elements and we have locally performed the required operations[2] (such as integration, differentiation, etc.), we need to impose some sort of connectivity between those subdomains, in order to solve the global problem. Depending on the formulation we are applying, these connectivity rules can change. In classical continuous Galerkin projections C^0 continuity is imposed across the elements. This is imposed by ensuring that the local solution on each element boundaries is identical to the boundary solution of the adjacent element. Another approach is to apply a discontinuous Galerkin projection where the connectivity between elements is assured by imposing continuity of boundary fluxes, hence allowing the variable to be discontinuous through elements edges. The discontinuous Galerkin approach is briefly described in next chapter.

In case of a continuous Galerkin projection we can represent our global bi-dimensional variable as

$$u(x_1, x_2) = \sum_{n \in \mathcal{N}^g} \phi_n^g(x_1, x_2)\hat{u}_n^g = \sum_{e \in \mathcal{E}} \sum_{n \in \mathcal{N}} \phi_n^e(x_1, x_2)\hat{u}_n^e, \tag{2.24}$$

where $\hat{u}_n^g$ are the $\mathcal{N}^g$ global coefficients which can be retrieved from the $\mathcal{E} \times \mathcal{N}$ elemental local coefficients. In vectorial notation we can define $\hat{\boldsymbol{u}}^l$ as the collection of all the elemental coefficients $\hat{\boldsymbol{u}}^e$ as

$$\hat{\boldsymbol{u}}^l = \begin{bmatrix} \hat{\boldsymbol{u}}^1 \\ \hat{\boldsymbol{u}}^2 \\ \vdots \\ \hat{\boldsymbol{u}}^{\mathcal{E}} \end{bmatrix}. \tag{2.25}$$

The relation between $\hat{\boldsymbol{u}}^l$ and the global coefficients vector $\hat{\boldsymbol{u}}^g$ can be seen as a scattering

[2]Operations are carried on the standard element Ω^{std} and then mapped back to the generically shaped element.

of the global one onto the local ones as

$$\hat{\boldsymbol{u}}^l = \boldsymbol{\mathcal{A}}\hat{\boldsymbol{u}}^g, \tag{2.26}$$

where $\mathcal{A}$ is an assembly matrix, which concatenates all the elements into a global expansion and applies the connectivity rules between elements. The assembly matrix entries are ± 1 and it is generally sparse.

An example of this procedure can be appreciated when assembling the global mass matrix $\boldsymbol{M}$, which is defined as

$$\boldsymbol{M}^g = \int_\Omega \phi_m^g(x_1, x_2)\phi_n^g(x_1, x_2)dx_1dx_2. \tag{2.27}$$

Locally we can evaluate the elemental contributions as

$$\boldsymbol{M}^e = \int_\Omega \phi_m^e(x_1, x_2)\phi_n^e(x_1, x_2)dx_1dx_2. \tag{2.28}$$

and assemble them using the scattering matrix $\boldsymbol{\mathcal{A}}$ as

$$\boldsymbol{M}^g = \boldsymbol{\mathcal{A}}^\top[\boldsymbol{M}^e]\boldsymbol{\mathcal{A}} \tag{2.29}$$

where

$$[\boldsymbol{M}^e] = \begin{bmatrix} \boldsymbol{M}^1 & 0 & 0 & \cdots & 0 \\ 0 & \boldsymbol{M}^2 & 0 & \cdots & 0 \\ 0 & 0 & \boldsymbol{M}^3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \boldsymbol{M}^{\mathcal{E}} \end{bmatrix}. \tag{2.30}$$

Other operators can be assembled in the same manner. For brevity we omit the construction of other operators although we report some basic concepts of numerical integration and differentiation in sections 2.1.5 and 2.1.6 respectively.

2.1.3.3 Basis Type

For the spectral/*hp* element method, the generic expansion basis series ϕ_p, spanning the polynomial space of order P, is composed by a series of functions (or modes). The subset of these modes which have non-zero support on the boundaries[3] are called *boundary*

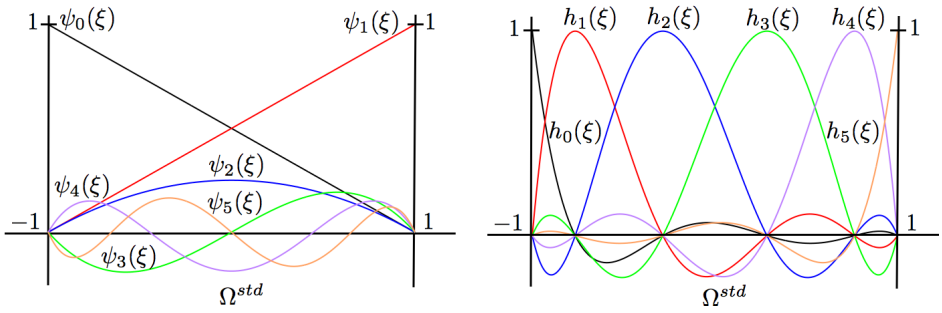
[3]Numerically speaking for boundary we intend the quadrature points located on the boundaries of the element.

modes. On the other hand the remaining modes are defined to be zero on all boundaries and nonzero in the interior part or the element, hence they are named *interior modes*. This type of basis facilitates the imposition of C^0 continuity through elements. In fact just the the boundary modes are required to be assembled as unique global modes using the assembly procedure described in the previous section. Throughout all the thesis only boundary-interior decomposed basis will be taken into account, either nodal or modal, and we provide a brief description of them in this section.

The modal expansion basis we use consists of modified Jacobi polynomials $\mathbb{P}_p^{\alpha,\beta}(\xi)$ which span the polynomial space of order P. Selecting $\alpha = 1$ and $\beta = 1$ and using linear basis as boundary modes, we construct the one-dimensional expansion $\phi_p(\xi)$ as

$$\phi_p(\xi) = \psi_p(\xi) = \begin{cases} \frac{1-\xi}{2} & for \quad p = 0 \\ \frac{1-\xi}{2}\,\frac{1+\xi}{2}\,\mathbb{P}_{p-1}^{1,1}(\xi) & for \quad 0 < p < P \\ \frac{1+\xi}{2} & for \quad p = P. \end{cases} \tag{2.31}$$

Fig. 2.2(a) shows the modified modal basis described by Eq. (2.31).The quadrature points for this basis type, required for numerical integration, are the Gauss-Lobatto-Legendre.



(a) Modal Expansion (b) Nodal Expansion

Figure 2.2: Graphical representation of a fifth order modal (a) and nodal (b) 1D basis on the standard element ($P = 5$). The modal and nodal basis refer to Eq. (2.31) and Eq. (2.32) respectively.

The nodal basis consists of Lagrange polynomials with the zeros corresponding to the Gauss-Lobatto-Legendre quadrature points represented in Fig. 2.2(b). This choice of basis is usually caller the spectral element method. Lagrange basis is shaped starting from the nodal points ξ_q (in our case the quadrature points). Given $P + 1$ nodal points ξ_q for

$0 \leq q \leq P$ the Lagrange polynomial $h_p(\xi)$ is defined as

$$\phi_p(\xi) = h_p(\xi) = \frac{\prod_{q=0,q\neq p}^{P}(\xi - \xi_q)}{\prod_{q=0,q\neq p}^{P}(\xi_p - \xi_q)}. \tag{2.32}$$

By definition this basis is boundary-interior decomposed, since all the modes are non-zero on the quadrature point they refer to; and zero otherwise.

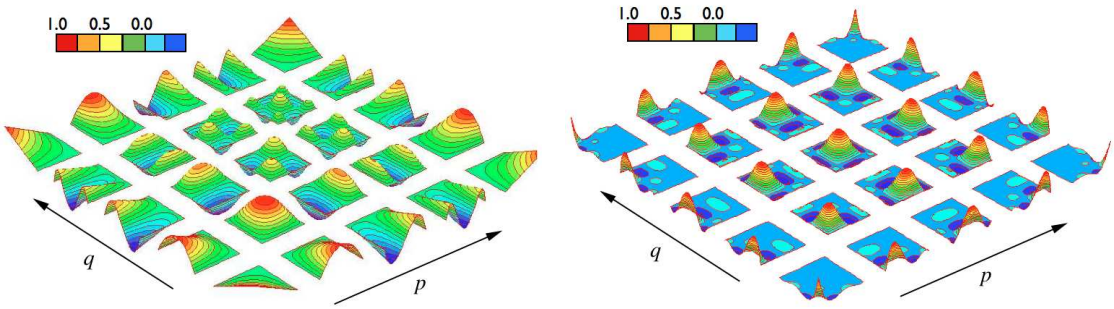
2.1.3.4 Tensorial Expansion Basis

We restrict our attention on bi-dimensional expansions basis, since they are the ones utilised for the rest of the thesis. Within the standard reference system, our generic variable u is approximated via a series of bi-dimensional functions (the basis functions $\boldsymbol{\phi}_n(\xi_1, \xi_2)$) obtained as the tensor product between two one-dimensional basis functions.

For a standard quadrilateral element defined as the bi-unit square $\mathcal{Q}^2 = \{(\xi_1, \xi_2) \in [-1, 1] \times [-1, 1]\}$, the generic variable $u(\xi_1, \xi_2)$ is approximated as

$$u^{\delta}(\xi_1, \xi_2) = \sum_{n=0}^{N} \phi_n(\xi_1, \xi_2)\hat{u}_n = \sum_{p=0}^{P}\sum_{q=0}^{P} \phi_p(\xi_1)\phi_q(\xi_2)\hat{u}_{pq}. \tag{2.33}$$

The basis $\phi_p(\xi_1)$ and $\phi_q(\xi_2)$ are mono-dimensional basis spanning the polynomial space of order P. A visualisation of a tensorial expansion basis on a standard quadrilateral region is depicted in Fig. 2.3. For the turbulent flow simulations reported in section 2.3.2 we will make use of the nodal expansion basis illustrated in Fig. 2.3(b).



(a) Modal Expansion (b) Nodal Expansion

Figure 2.3: Construction of a 2D quadrilateral expansion basis as a tensor product of two 1D basis. Modal basis (a) and nodal basis (b). Courtesy of (Karniadakis & Sherwin 2005).

2.1.4 Spectral Method

The spectral method performs a global discretisation approach, approximating the solution as a linear combination of continuous functions. These functions are global and non-zero over the solution domain. Unlike element-based methods, such as the FEM and the SEM, spectral method does not require any domain partition, i.e. any meshing process because they are designed for domains of the form $[a, b]^n$, where n is the number of dimensions. The idea is to substitute in the PDEs the generic variable $u(\boldsymbol{x})$ with a truncated series expansion as reported in Eq. (2.2). Classic choices for the expansion basis $\boldsymbol{\Phi}_i(\boldsymbol{x})$ are Fourier expansions, Chebyshev polynomials or Jacobi polynomials depending on the type of boundary conditions to be imposed. (Boyd 2001, Canuto et al. 2006, 2007).

As previously mentioned, the spectral method can be applied along with a collocation or a Galerkin projection. In the first case, also called pseudo-spectral approach, the unknown coefficients $\hat{u}_i$ are the solution values on the collocation points. In the case of a Galerkin projection, numerical integrals need to be calculated, scaling the expansion coefficients $\hat{u}_i$ with the integration weights w_i.

In the field of computational fluid dynamics, the spectral method is often used in combination with Fourier expansions to study periodic flows in simplified geometries (Orszag & Israeli 1974, Orszag 1980, Hussaini & Zang 1987). The spectral method provides very low-error approximations which can, if the solution is smooth, converge exponentially. The use of Fourier series limits the type of problems which can be studied. In particular, the solution has to be periodic in the direction along which it is approximated with a Fourier expansion.

An example of a typical fluid dynamics problem which is studied with this approach is isotropic turbulence. Isotropic turbulence allows a three-dimensional spectral approach in combination with harmonic series because it is fully periodic in all the spatial dimensions. Also, turbulent channel flow problems are often investigated using a spectral method combined with harmonic functions. Compared to the isotropic turbulence problem, the channel flow problem has just two periodic directions. As a consequence one spatial direction has to be approximated with another expansion type (usually Jacobi or Chebyshev polynomials).

Investigations of these fluid dynamic problems using the spectral method started in 1971 with the work of $Orszag$ and $Patterson$ (Patterson & Orszag 1971). In these studies the spectral method was used to investigate isotropic incompressible turbulence. The authors solved the incompressible Navier-Stokes equations in a cubic box with periodic boundary conditions. They presented algorithms to solve the Fourier-transformed 3D Navier-Stokes equations, using the FFT algorithm to switch between the physical and the wave-vector space. They also presented methods to remove aliasing errors which are more efficient than previous techniques for dealiasing discrete Fourier transforms. $Gottlieb$ and $Orszag$ in 1977 presented a monograph entitled "*Numerical Analysis of Spectral Methods: Theory and Applications*" in which the spectral method is carefully described and fluid dynamics applications are highlighted (Gottlieb & Orszag 1977). In 1981 $Rogallo$ (Rogallo 1981) used the spectral method to investigate homogenous turbulence, extending the work of (Patterson & Orszag 1971) and comparing the results with experimental data. He used a 3D Fourier expansion, the FFT algorithm and a moving coordinate system. $Rogallo$ also emphasised dealiasing techniques, in relation to the Navier-Stokes equations non-linear term. The convective term acts to increase the wave-numbers space. This leads to the introduction of aliasing errors. The author reported two general methods to remove aliasing. These two techniques have been previously presented by (Patterson & Orszag 1971) and will be briefly described in section 2.1.4.2. We would also like to recall the seminal works of $Moin$, Kim and $Moser$ (Moin & Kim 1982, Kim et al. 1987, Kim 1989, Moser et al. 1999) as a reference for numerical investigations of turbulent channel flows using spectral methods.

We avoid a detailed description of previous numerical investigations of fluid dynamic problems because it is beyond the scope of this chapter. Therefore we move our attention to a description Fourier basis, since they are the ones used throughout the rest of this thesis. In section 2.1.7 we will combine the SEM method with a 1D SM where the expansion basis is a Fourier series. This approach, as we will highlight later on in the chapter, allows a modal decoupling of the 3D domain thanks to the orthogonality of the Fourier basis.

2.1.4.1 Fourier Basis

Assuming a one-dimensional domain $\Omega = [a, b]$ defined by the linear coordinate z, the Fourier series of a function $u(z)$ represents the formal expansion of $u(z)$ in terms of the Fourier orthogonal system and it is defined as

$$u(z) = \sum_{k=-\infty}^{+\infty} \hat{u}_k \phi_k(z) \tag{2.34}$$

where the expansion functions $\phi_k(z)$ are defined as

$$\phi_k(z) = e^{ikz} \tag{2.35}$$

The Fourier system is orthogonal over the interval $(0, 2\pi)$ and it can be formally written as follows[4]:

$$\int_0^{2\pi} \phi_k(z)\bar{\phi}_l(z)dz = 2\pi\delta_{kl} = \begin{cases} 0 & if \quad k \neq l \\ 2\pi & if \quad k = l \end{cases} \tag{2.36}$$

The Fourier coefficients $\hat{u}_k$ are defined as

$$\hat{u}_k = \frac{1}{2\pi} \int_0^{2\pi} u(z)e^{-ikz}dz \qquad k = 0, \pm 1, \pm 2, \ldots \tag{2.37}$$

In some cases it is convenient to convert the general Fourier expansion (where the functions are exponential) in a *cosine* or *sine* Fourier expansion, in which the coefficients are respectively

$$\hat{a}_k = \frac{1}{2\pi} \int_0^{2\pi} u(z)\cos(kz)dz \qquad k = 0, 1, 2, \ldots \tag{2.38}$$

$$\hat{b}_k = \frac{1}{2\pi} \int_0^{2\pi} u(z)\sin(kz)dz \qquad k = 0, 1, 2, \ldots \tag{2.39}$$

and are related to the previous expansion via the following relation[5]

$$\hat{u}_k = \hat{a}_k + \boldsymbol{i}\hat{b}_k \qquad k = 0, 1, 2, \ldots \tag{2.40}$$

In numerical applications that make use of the Fourier expansions to approximate a function $u(z)$, the Fourier series can not be implemented precisely because the coefficients

[4] $\bar{\phi}_l(z)$ is the complex conjugate of $\phi_l(z)$ and δ_{kl} is the Kronecker delta

[5] If $u(z)$ is a real valued function, $\hat{a}_k$ and $\hat{b}_k$ are real numbers and $\hat{u}_{-k} = \bar{\hat{u}}_k$

are not known in closed form and hence must be approximated. The solution to overcome the problem is the use of a discrete Fourier transform (DFT) and its related series of coefficients. Passing from a continuous to a discrete approach, a set of points has to be selected, which are called nodes or grid points, defined as

$$z_j = \frac{2\pi j}{N} \qquad j = 0, 1, 2, \ldots, N-1 \qquad N > 0 \tag{2.41}$$

Using this set of points, the discrete Fourier coefficients of a function $u(z)$ in $[0, 2\pi]$ are

$$\hat{u}_k = \frac{1}{N} \sum_{j=0}^{N-1} u(z_j) e^{-ikz_j} \qquad k = -N/2, \ldots, N/2 - 1 \tag{2.42}$$

and the inversion formula is

$$u^{\delta}(z_j) = \sum_{k=-N/2}^{N/2-1} \hat{u}_k e^{ikz_j} \qquad j = 0, \ldots, N-1 \tag{2.43}$$

2.1.4.2 Dealiasing

In a numerical environment, when multiplying two variables approximated with a finite Fourier series, the common approach is to represent the resulting product using a finite Fourier series which has the same number of modes as the two original ones. This process introduces an error in the resulting Fourier expanded variable called *aliasing*. In order to show the aliasing effect assume we have two vectors we want to multiply $\hat{u}_k$ and $\hat{v}_k$ and we transform them in their physical format $u(z_j)$ and $v(z_j)$, as reported in Eqs. (2.44) and (2.45).

$$u(z_j) = \sum_{k=-N/2}^{N/2-1} \hat{u}_k e^{ikz_j} \qquad j = 0, 1, \ldots, N-1 \tag{2.44}$$

$$v(z_j) = \sum_{k=-N/2}^{N/2-1} \hat{v}_k e^{ikz_j} \qquad j = 0, 1, \ldots, N-1 \tag{2.45}$$

We then perform the multiplication in physical space to get the resulting vector $s(z_j)$

$$s(z_j) = u(z_j) v(z_j). \tag{2.46}$$

Hence we transform back the result

$$\tilde{s}_k = \frac{1}{N} \sum_{j=0}^{N-1} s(z_j) e^{-ikz_j} \qquad k = -\frac{N}{2}, \ldots, \frac{N}{2} - 1 \tag{2.47}$$

However, if we were to take Eq. (2.44) and Eq. (2.45) and multiply them in a spectral setting, we see that

$$\tilde{s}_k = \sum_{m+n=k} \hat{u}_k \hat{v}_k + \sum_{m+n=k\pm N} \hat{u}_k \hat{v}_k = \hat{s}_k + aliasing. \tag{2.48}$$

As shown in Eq. (2.48), the resulting vector $\tilde{s}_k$ is not the expected $\hat{s}_k$ but there is an additional term, the aliasing error.

The first possible technique for removing aliasing is the *removal by truncation*, where the N length alias-free product of two Fourier series of length N is obtained using a $3N/2$ length Fourier transform. This technique is known as *padding* or the $3/2 - rule$. The two coefficients vectors to be convoluted $\hat{u}_k$ and $\hat{v}_k$ in Eq. (2.44) and Eq. (2.45) are extended and padded with zeros (the final length of the vectors is $3/2$ of the original one) . We then calculate $u(z_j)$ and $v(z_j)$ with an DFT counting $3N/2$ modes (the last $N/2$ coefficients are zero). After applying Eq. (2.46), we transform back the result $s(z_j)$. In this way aliasing effects are confined in the last $N/2$ coefficients $\tilde{s}_k$ which are outside the original vector of length N. Dumping those coefficients we obtain an alias-free product of two Fourier series of length N. This is the most commonly used method and the one we decided to use for our applications. The latest developments in this approach are the works of $Bowman$ and $Roberts$ (Bowman & Roberts 2011, Roberts & Bowman 2011) describing a technique to implicitly dealias a convolution product.

The second method reported by $Rogallo$ (Rogallo 1981) is the *aliasing removal by phase shifts*. This technique is based on the consideration that a shift in the physical space grid results in the multiplication of the Fourier modes by a phase factor in the wave-space. If we perform Fourier series products on the shifted grid and then we shift the results back to the original one, we have that the aliasing error is multiplied by a phase factor. This factor can be used to eliminate or reduce aliasing. The aliasing removal by phase-shifting is rarely used because it requires a memory doubling and a greater number of operations.

2.1.5 Numerical Integration

Contrary to what happens in case of a collocation[6] projection, in a Galerkin method integrals such as the ones shown in Eq. (2.17a) need to be evaluated. These integrals are generally evaluated throughout a quadrature formula (see Eq. (2.49)), where the value of the integral is approximated as the sum of the function values in Q quadrature points $\boldsymbol{x}_j$ multiplied by Q quadrature weights w_j. In a spectral/*hp* element method integrals are carried out on standard domain Ω^{std} into which the real domain Ω^e is mapped . The number of points/weights and their values define the quadrature formula. The quadrature formulas mostly used along with Galerkin projections are the well-known *Gaussian quadrature* (Stroud 1966) rules,

$$\int_{\Omega} \boldsymbol{\Psi}(\boldsymbol{x}) d\boldsymbol{x} \approx \sum_{j=0}^{Q-1} w_j \boldsymbol{\Psi}(\boldsymbol{x}_j). \tag{2.49}$$

Gaussian quadrature formulas are very accurate for the integration of smooth functions and they are defined on the standard domain. If our variable u is defined (or mapped) into the standard domain, for the one-dimensional case we can write

$$\int_{-1}^{1} u(\xi) d\xi \approx \sum_{j=0}^{Q-1} w_j u(\xi_j) \tag{2.50}$$

allowing the exact integration of polynomials $\mathcal{P}_P$ of order P greater than $Q-1$. If we assume that the integrand $u(\xi)$ is a polynomial of order P, the maximum value of P we can exactly integrate with a Gaussian quadrature formula depends on the nature of the quadrature points ξ_j. The quadrature formula is exact if:

- $u(\xi)$ is a polynomial of order $2Q-1$ and the quadrature points ξ_j do not include the extremes of the interval $[-1,1]$ (Gauss-Legendre points).

- $u(\xi)$ is a polynomial of order $2Q-2$ and the quadrature points ξ_j include one of the two extremes of the interval $[-1,1]$ (Gauss-Radau-Legendre points).

- $u(\xi)$ is a polynomial of order $2Q-3$ and the quadrature points ξ_j include both the extremes of the interval $[-1,1]$ (Gauss-Lobatto-Legendre points).

[6] In the collocation approach the test functions are Dirac deltas which extract the values of the trial functions in the collocation points, removing the need of actually calculate the integral.

Extending Eq. (2.50) to higher tensor product dimensions is quite trivial, for example, for a quadrilateral standard element we have

$$\int_{-1}^{1}\int_{-1}^{1} u(\xi_1,\xi_2)d\xi_1 d\xi_2 \approx \sum_{j=0}^{Q_1-1} w_j \left[\sum_{i=0}^{Q_2-1} w_i u(\xi_{1j},\xi_{2i})\right]. \tag{2.51}$$

However, we need to integrate our variable on the real domain Ω^e, which can be achieved by introducing the coordinate transformation described in section 2.1.3 as

$$\int_{\Omega^e} u(x_1,x_2)dx_1dx_2 = \int_{\Omega^{std}} u(\xi_1,\xi_2)|J|d\xi_1 d\xi_2 \tag{2.52}$$

where J is the Jacobian of the the trasformation, defined as

$$J = \begin{vmatrix} \dfrac{\partial x_1}{\partial \xi_1} & \dfrac{\partial x_2}{\partial \xi_1} \\ \dfrac{\partial x_1}{\partial \xi_2} & \dfrac{\partial x_2}{\partial \xi_2} \end{vmatrix} = \frac{\partial x_1}{\partial \xi_1}\frac{\partial x_2}{\partial \xi_2} - \frac{\partial x_2}{\partial \xi_1}\frac{\partial x_1}{\partial \xi_2}. \tag{2.53}$$

2.1.6 Numerical Differentiation

When solving partial differential equations, a fundamental requirement is the approximation of the derivatives of our variables. In the case of a spectral method using a Fourier expansion as the basis, the differentiation is as trivial as scaling each spectral coefficient. Differentiation of Fourier based expansions is briefly shown in section 2.1.7. In this section we focus on giving an overview of basic differential operators for a general polynomial basis. If we assume our discrete variable $u^\delta(\boldsymbol{x})$ can be represented through a polynomial expansion of order P such as

$$u^\delta(\boldsymbol{x}) \approx \sum_{n=0}^{P} \phi_n(\boldsymbol{x})\hat{u}_n \tag{2.54}$$

the derivative can be defined as

$$\frac{du^\delta(\boldsymbol{x})}{d\boldsymbol{x}} = \sum_{n=0}^{P} \frac{d\phi_n(\boldsymbol{x})}{d\boldsymbol{x}}\hat{u}_n \tag{2.55}$$

When applying this approach to a spectral/*hp* element discretisation a natural choice is to perform derivatives in the standard domain. A convenient technique is the *collocation*

differentiation, which yields the values of the derivatives in the quadrature points we are using. Following this method we approximate our variable on the standard domain as

$$u(\xi) \approx u^{\delta}(\xi) = \sum_{j=0}^{Q-1} h_j(\xi) u(\xi_j), \tag{2.56}$$

where $h_j(\xi)$ are the Lagrange polynomials described in Eq. (2.32), interpolating our variable through a set of Q quadrature points ($u(\xi_j)$). Hence we can evaluate the derivative in a quadrature point ξ_i as

$$\frac{du(\xi_i)}{d\xi} \approx \frac{du^{\delta}(\xi_i)}{d\xi} = \sum_{j=0}^{Q-1} \frac{dh_j(\xi_i)}{d\xi} u(\xi_j) = \sum_{j=0}^{Q-1} d_{ij} u(\xi_j) \tag{2.57}$$

which translates to having the derivative in the ξ_i quadrature points based on the variable values in the same ξ_j points.

Extension to bi-dimensional domains follows easily. If we consider a standard quadrilateral element, where

$$u^{\delta}(\xi_1, \xi_2) = \sum_{i,j} h_{ij}(\xi_1, \xi_2) u(\xi_{1_i}, \xi_{2_j}) \tag{2.58}$$

and $h_{ij}(\xi_1, \xi_2)$ is a tensorial basis, defined as

$$h_{ij}(\xi_1, \xi_2) = h_i(\xi_1) h_j(\xi_2), \tag{2.59}$$

the derivative with respect to one of the coordinates (for example ξ_1) at a quadrature point $\xi_{sr} = [\xi_{1_s}, \xi_{2_r}]$ is

$$\frac{\partial u^{\delta}(\xi_{1_s}, \xi_{2_r})}{\partial \xi_1} = \sum_{i,j} \frac{dh_{ij}(\xi_{1_s}, \xi_{2_r})}{d\xi_1} u(\xi_{1_i}, \xi_{2_j}) = \sum_{i,j} \frac{dh_i(\xi_{1_s})}{d\xi_1} h_j(\xi_{2_r}) u(\xi_{1_i}, \xi_{2_j}). \tag{2.60}$$

Recalling the collocation properties of Lagrange basis $h_j(\xi_{2_r}) = \delta_{jr}$ and substituting $dh_i(\xi_{1_s})/d\xi_1 = d_{si}$ as described in Eq. (2.57), we can write

$$\frac{\partial u^{\delta}(\xi_{1_s}, \xi_{2_r})}{\partial \xi_1} = \sum_{i,j} d_{si} \delta_{jr} u(\xi_{1_i}, \xi_{2_j}) = \sum_{i,j} d_{si} u(\xi_{1_i}, \xi_{2_r}). \tag{2.61}$$

In order to move our differential operator to the generic domain Ω^e, we apply the chain rule taking advantage from the metric described in section 2.1.7, so that

$$\begin{bmatrix} \frac{\partial u}{\partial x_1} \\ \frac{\partial u}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi_1}{\partial x_1} & \frac{\partial \xi_2}{\partial x_1} \\ \frac{\partial \xi_1}{\partial x_2} & \frac{\partial \xi_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial \xi_1} \\ \frac{\partial u}{\partial \xi_2} \end{bmatrix} = \frac{1}{J} \begin{bmatrix} \frac{\partial x_2}{\partial \xi_2} & -\frac{\partial x_2}{\partial \xi_1} \\ -\frac{\partial x_1}{\partial \xi_2} & \frac{\partial x_1}{\partial \xi_1} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial \xi_1} \\ \frac{\partial u}{\partial \xi_2} \end{bmatrix}. \tag{2.62}$$

2.1.7 Fourier Spectral/*hp* Element Method

Following the approach presented by $Karniadakis$ in 1990 (Karniadakis 1990), we study the discretisation of three-dimensional problem by combining the spectral method and the spectral/*hp* element method as illustrated in Fig. 2.4 . The first step of this technique

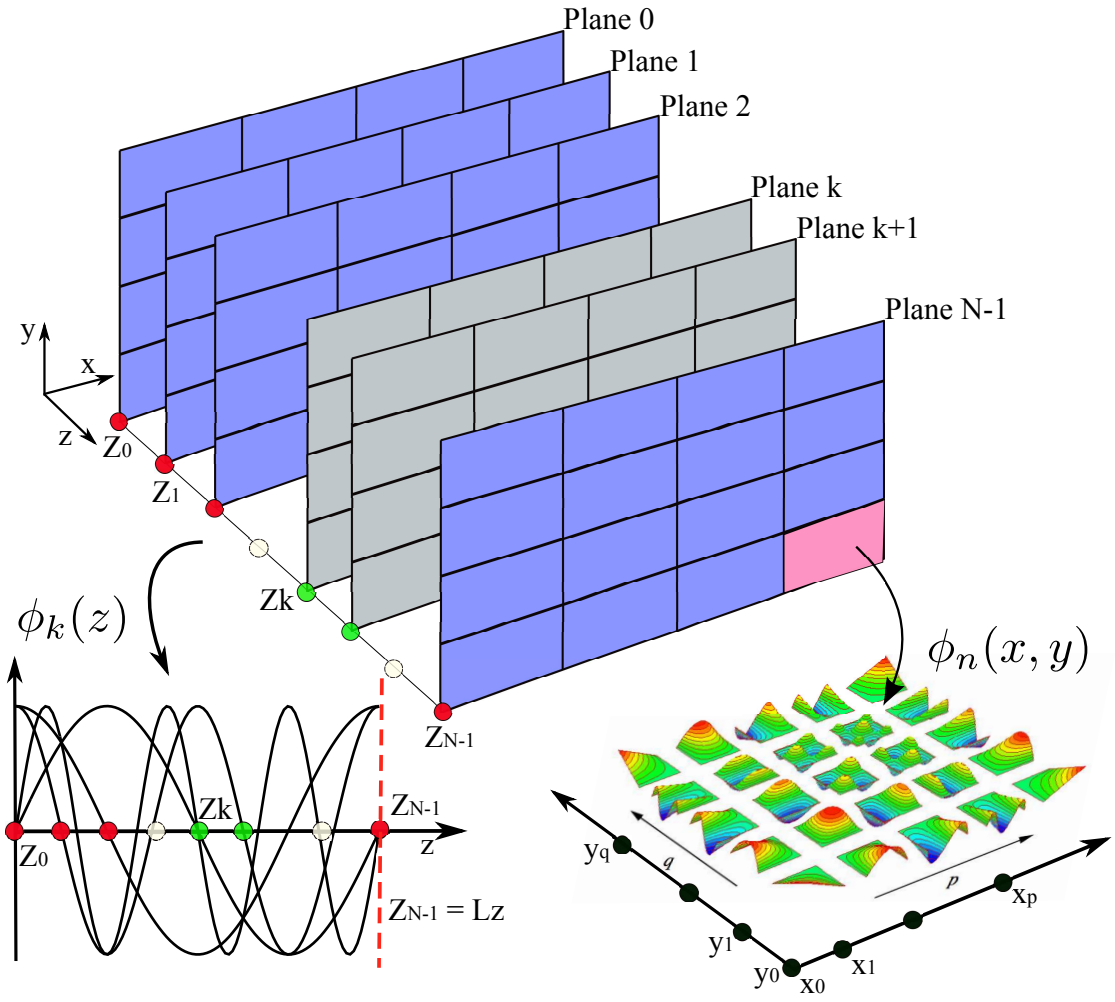


Figure 2.4: Structure of a three-dimensional Cartesian expansion using a spectral/*hp* element method in xy-plane (12 quadrilateral elements) and a spectral method in z-direction.

consists of using a bi-dimensional spectral/*hp* element method to spatially discretise the problem in the xy plane, hence a mesh of 2D elements is required. On each element the variable $u(x, y)$ is approximated with $u^{\delta}(x, y)$ using the quadrilateral (or triangular) tensorial expansion basis described in section 2.1.3.4. The bi-dimensional discrete variable

$u^\delta(x,y)$ can be written as

$$u^\delta(x,y) = \sum_n \phi_n(x,y)\hat{u}_n = \sum_{p=0}^{P}\sum_{q=0}^{P} \phi_p(x)\phi_q(y)\hat{u}_{pq}. \tag{2.63}$$

The third dimension, the $z-$direction, is introduced replicating the 2D problem on each one of the N quadrature points of the $z-$direction expansion basis. We can write the discrete variable $u^\delta(x,y,z)$ as

$$u^\delta(x,y,z) = \sum_{pqk} \phi_{pq}(x,y)\phi_k(z)\hat{u}_{nk} = \sum_{pqk} \phi_{pqk}(x,y,z)\hat{u}_{pqk}. \tag{2.64}$$

A pseudo-code describing the object-orientated approach of *Nektar++* is reported in **Algorithm 1**. Compared to a full 3D spectral/*hp* element approach, using a series of 2D planes reduces the size of the matrix problem involved in the solution process, also leading to a reduction in the amount of memory used.

```
// Create a Fourier spectral/hp element expansion
plane = constructor2D(mesh2D, P, boundary conditions)
// Duplication of the planes in z−direction
for i = 0 to 2k − 1 do
    PlanesVector[i] = plane;
end
// Transposition object to shuffle data across planes
transposition = constructor(N^planes)
// DFT object to perform transformations
dft = constructor(type, N^planes)
```

Algorithm 1: Fourier spectral/*hp* element method construction in *Nektar++* .

For the study of incompressible flows, and for fluid dynamics applications in general, a classic choice for the global expansion basis $\phi_k(z)$ is an harmonic expansion. We select $\phi_k(z)$ to be a real cosine/sine Fourier series of the form

$$\phi_k(z) = \cos(k\pi z/L_z) + sin(k\pi z/L_z), \tag{2.65}$$

which in vectorial notation can be written as

$$\phi_k(z) = \begin{bmatrix} \cos(k\pi z/L_z) \\ \sin(k\pi z/L_z) \end{bmatrix} \qquad \hat{u}_k = \begin{bmatrix} \hat{u}_k^c & \hat{u}_k^s \end{bmatrix} \tag{2.66}$$

where k is the mode number and L_z is the domain length in the z-direction. The combination of the spectral expansion with the elemental 2D discretisation yields to the following data structure

$$u(x,y,z) = \begin{bmatrix} u(x,y,z_0) \\ u(x,y,z_1) \\ . \\ . \\ . \\ u(x,y,z_{N-1}) \end{bmatrix} \qquad \hat{u}_{pqk} = \begin{bmatrix} \hat{u}_{pq0}^c \\ \hat{u}_{pq0}^s \\ . \\ . \\ \hat{u}_{pqk}^c \\ \hat{u}_{pqk}^s \end{bmatrix} \tag{2.67}$$

where $u(x,y,z_i)$ is the vector containing the solution values at the quadrature points of each one of the N 2D planes. The quadrature points in $z-$direction are selected to be equally-spaced on the interval $0 \leq z \leq L_z$. Hence we have $N/2$ Fourier-cosine modes with $0 \leq k < N/2$ and $N/2$ Fourier-sine modes with $0 \leq k < N/2$.

The usage of the Fourier basis requires that both the solution function and the underlying geometry be homogeneous (periodic) in the $z-$direction to satisfy the Fourier basis essential property, i.e. $u(z_0) = u(z_{N-1})$.

Transformations between physical and coefficient space are usually represented as a matrix-vector multiplications. For the spectral expansion described above, if we choose for example a number of Fourier modes $N = 4$ ($i.e.$ two cosine and two sine trial functions with $k = [0\ 1]$) and four equally spaced points $\{z_0, z_1, z_2, z_3\}$, the transformation matrix for a variable $u(z)$ expanded over L_z is

$$\boldsymbol{B} = \begin{bmatrix} cos(k_0\pi z_0/L_z) & sin(k_0\pi z_0/L_z) & cos(k_1\pi z_0/L_z) & sin(k_1\pi z_0/L_z) \\ cos(k_0\pi z_1/L_z) & sin(k_0\pi z_1/L_z) & cos(k_1\pi z_1/L_z) & sin(k_1\pi z_1/L_z) \\ cos(k_0\pi z_2/L_z) & sin(k_0\pi z_2/L_z) & cos(k_1\pi z_2/L_z) & sin(k_1\pi z_2/L_z) \\ cos(k_0\pi z_3/L_z) & sin(k_0\pi z_3/L_z) & cos(k_1\pi z_3/L_z) & sin(k_1\pi z_3/L_z) \end{bmatrix}. \tag{2.68}$$

2.1.7.1 Helmholtz Problem

Solving the Helmholtz equation is a fundamental step for the solution of a wide range of problems. In case of fluid dynamics applications the solution of elliptic problems occurs quite often and it is fundamental part of the scheme to be used throughout this thesis. The three-dimensional Helmholtz equation for the variable $u = u(x, y, z)$ is

$$\nabla^2 u + \lambda u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + \lambda u = f. \tag{2.69}$$

As the operator is linear, it can be re-written as a 2D Laplacian operator in xy (associated with the spectral/*hp* element method) plus a 1D Laplacian for the spectral part of the discretisation

$$\nabla^2_{2D} u + \frac{\partial^2 u}{\partial z^2} + \lambda u = f \tag{2.70}$$

where

$$\nabla^2_{2D} u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}. \tag{2.71}$$

The second-order spatial derivate in the $z-$direction is then applied to the solution expansion

$$\frac{\partial^2 u}{\partial z^2} = \frac{\partial^2}{\partial z^2} \sum_{pqk} \phi_{pq}(x, y)\phi_k(z)\hat{u}_{pqk} = \sum_{pqk} \phi_{pq}(x, y) \frac{\partial^2}{\partial z^2} \phi_k(z)\hat{u}_{pqk}. \tag{2.72}$$

Since we are using a Fourier basis we can differentiate our variable in wave space by multiplying the solution by the wave number $\beta_k = k\pi/L_z$. In fact we have that

$$\frac{\partial}{\partial z}\phi_k(z) = \frac{\partial}{\partial z} \cos(\beta_k z) = -\beta_k \sin(\beta_k z) \tag{2.73}$$

and

$$\frac{\partial}{\partial z}\phi_k(z) = \frac{\partial}{\partial z} \sin(\beta_k z) = \beta_k \cos(\beta_k z). \tag{2.74}$$

Therefore, the second derivatives will always be

$$\frac{\partial^2}{\partial z^2}\phi_k(z) = -\beta_k^2 \phi_k(z). \tag{2.75}$$

Since only second order spatial derivatives are involved, the three-dimensional Helmholtz problem can be seen as a series of two-dimensional decoupled Helmholtz problems (if we are in Fourier space), where in each equation the Helmholtz coefficients are modified to give

$$\nabla^2_{2D} u + (\lambda - \beta_k^2)u = f. \tag{2.76}$$

From a practical point of view, the 2D Helmholtz equation has to be solved as many times as the number of homogenous modes. In case of problems in which the first spatial derivatives respect to z is required, such as the non-linear term in the Navier-Stokes equations, we can not solve the equations on each plane independently. Therefore inter-planes operations are required. Such operations are performed with the help of a series of shuffling/unshuffling routines which reorder data structures to facilitate calculations along the grid of 1D spectral expansions.

The most expensive operation in the z-direction is the transformation between physical and coefficient space, naturally performed via a matrix-vector multiplication, as shown before. As the number of spectral collocation points is increased, the common matrix-vector multiplication becomes inefficient (Boyd 2001). The natural approach is to perform the transformation between physical and coefficient space using the FFT algorithm. FFTW is one of the most common and portable library to perform the FFT and it is currently developed at MIT by $Frigo$ and $Johnson$ (Frigo & Johnson 2005). This library will be used to perform all the transformations to and from Fourier space.

2.1.8 Verification of the Algorithm

The reliability of the algorithm has been initially tested using an elliptic problem. Numerical tests have been carried out using four-quadrilateral elements in a 2D mesh extended in $z-$direction to solve a 3D Helmholtz equation. The domain length in the third dimension is $L_z = 5$, discretised with 8 Fourier modes. The bi-dimensional domain is a square of size $x \in [0, 1] \times [0, 1]$ and Fig. 2.6 depicts both the domian and the solution. Boundary conditions are of Dirichlet type and directly taken from the exact solution reported in Eq. (2.77). The exact solution of this problem is

$$u_{ex} = \sin(\pi x) \sin(\pi y) \sin(2\pi z/L_z) \tag{2.77}$$

given the forcing term

$$f = -(\lambda + 2\pi^2 + 4\pi^2/L_z^2) \sin(\pi x) \sin(\pi y) \sin(2\pi z/L_z). \tag{2.78}$$

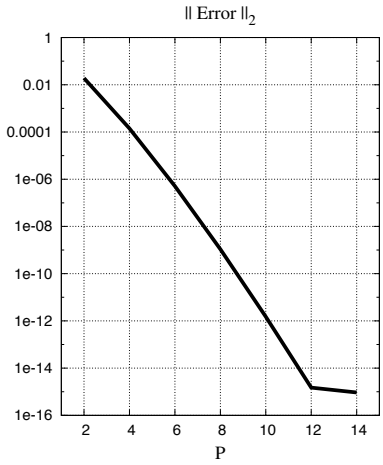


Figure 2.5: Error convergence as the polynomial expansion order is increased in the 2D planes for the 3D Helmholtz problem reported in Fig. 2.6.

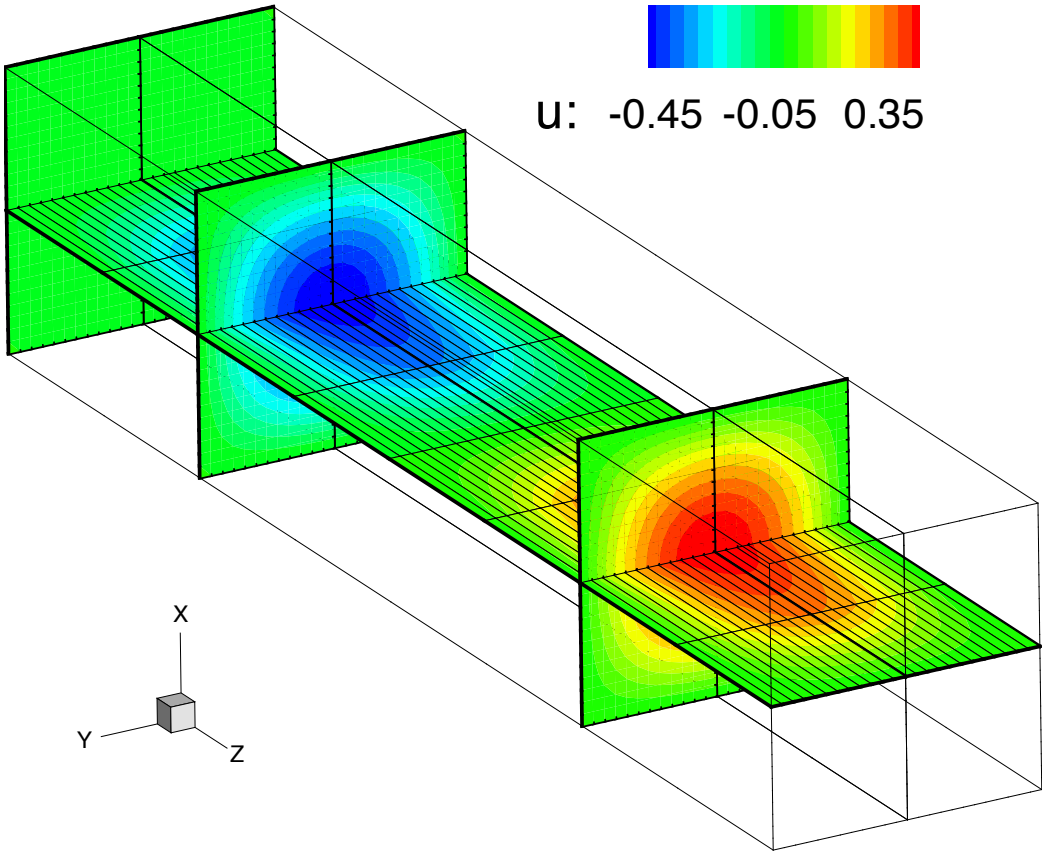


Figure 2.6: Solution of a 3D Helmholtz problem using the Fourier spectral/*hp* element method. The polynomial expansion $P = 10$ in combination with 8 Fourier modes.

Fig. 2.5 shows the convergence of the spatial error as the polynomial order P is increased. Convergence of the error with respect to the number of Fourier modes is negligible in this case given the harmonic nature of the solution, as the Fourier basis can exactly represent the solution.

2.2 Temporal Discretisation

Once the spatial operators have been discretised with an appropriate spatial technique (as reported in previous sections), the next step is to time-integrate our equations. One of the most standard choices is the use of multi-step schemes, where the solution at the next time-level is based on previous time-level solutions. Multi-stage schemes instead take advantages of the involved operator applications at various time-stages (not necessary corresponding to previous time-levels). In both cases the time-integration method is meant to be applied to an ODE (or system of ODEs), where the only explicit differentiation is the time derivative. Spatial derivatives are encapsulated within the spatial operators. This classical technique to reduce a system of PDEs to a system ODEs is called the Method of Lines (MoL) and we will describe it in section 2.2.1.

Multi-step and multi-stage schemes are both well-known and described in literature, hence we avoid a detailed treatment of them and we refer the reader to the complete work of $Wood$ (Wood 1990). However, classic considerations are that multi-step schemes require more memory for the storage of multiple time-steps, but they are computationally cheaper in term of floating-point operations. Multi-stage schemes on the other hand, while requiring less memory, may become fairly expensive because of the greater number of operator evaluations. Therefore, it is typical to encounter some uncertainties when we face the decision of selecting one approach or the other. To overcome these doubts we generalise the time-stepping formulation using the General Linear Method approach (GLM), where multi-step and multi-stage schemes can be implemented in the same manner, regardless if they are implicit, explicit or implicit-explicit.

$Butcher$s unifying General Linear Methods (Butcher 2006, 2009) allow one to move from one time-integration scheme to another simply by selecting the related pre-stored

coefficients matrix. This pre-calculated matrix contains the coefficients associated with a particular scheme. These coefficients are recast in a unified way leading to a generalised pre-calculated block-matrix for each time-integration scheme (see appendix B). This approach allows one to easily investigate various schemes facilitating numerical studies.

In the following we summarise the basic ideas of the time-integration approach we adopted and reported in (Vos et al. 2011). We start by presenting the Method of Lines and then we introduce the GLM. In section 2.2.3 the method is extended to encompass implicit-explicit methods. These schemes will be used to deal with the incompressible Navier-Stokes equations in section 2.3. Subsequently we dedicate a small section to the treatment of time-dependent boundary conditions and then we report a brief verification of the implementation.

2.2.1 The Method of Lines

Time-stepping schemes are commonly applied to initial value problems, which can be represented by ordinary differential equations. We are mostly interested in physical processes which, instead, are modelled via time-dependent partial differential equations. In this section we show briefly how to reduce a PDE system to an ODE system using the MoL, to which we can then apply the General Linear Method. The discussion will be carried out using the scalar advection diffusion equation as an example. The PDE reported in Eq. (2.79) will be spatially discretised with a spectral/*hp* element method. The problem is described by the following set of equations:

$$\frac{\partial u}{\partial t} + \nabla \cdot \boldsymbol{F}(u) = \nabla^2 u \qquad \text{in } \Omega \times [0, \infty) \tag{2.79a}$$

$$u(\boldsymbol{x}, t) = g_D(\boldsymbol{x}, t) \qquad \text{on } \partial\Omega_D \times [0, \infty) \tag{2.79b}$$

$$\frac{\partial u}{\partial n}(\boldsymbol{x}, t) = g_N(\boldsymbol{x}, t) \cdot \boldsymbol{n} \qquad \text{on } \partial\Omega_N \times [0, \infty) \tag{2.79c}$$

$$u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}) \qquad \text{in } \Omega \tag{2.79d}$$

where Ω is a bounded domain of $\mathbb{R}^d$ with boundary $\partial\Omega = \partial\Omega_D \bigcup \partial\Omega_N$ and $\boldsymbol{n}$ denotes the outward normal to the boundary $\partial\Omega$.

Applying a Galerkin projection (as reported in section 2.1.2) we multiply Eq. (2.79a) by a smooth test function $v = v(\boldsymbol{x})$, which by definition is zero on all Dirichlet bound-

aries. If we integrate over the entire spatial domain we obtain the following variational formulation: find $u \in \mathcal{U}$ such that[7]

$$\int_\Omega v\frac{\partial u}{\partial t}d\boldsymbol{x} - \int_\Omega vf(u)d\boldsymbol{x} = \int_\Omega v\nabla^2 u d\boldsymbol{x}, \quad \forall v \in \mathcal{V}, \tag{2.80}$$

where $\mathcal{U}$ and $\mathcal{V}$ are suitably chosen trial and test spaces respectively. We obtain the weak form of the diffusion operator by applying the divergence theorem to the right-hand-side term yielding: find $u \in \mathcal{U}$ such that

$$\int_\Omega v\frac{\partial u}{\partial t}d\boldsymbol{x} - \int_\Omega vf(u)d\boldsymbol{x} = -\int_\Omega \nabla v \cdot \nabla u d\boldsymbol{x} + \int_{\partial\Omega} v\nabla u \cdot \boldsymbol{n} d\boldsymbol{x}, \quad \forall v \in \mathcal{V}. \tag{2.81}$$

As $v(\partial\Omega_D)$ is equal to zero, only Neumann conditions will give contributions to the boundary integral, and we enforce the conditions weakly through substituting $\nabla u = \boldsymbol{g}_N$ in the boundary integral. In order to impose Dirichlet boundary conditions one can choose to adopt a lifting strategy where the solution is decomposed into a known function, u^D and an unknown homogeneous function u^H, *i.e.*

$$u(\boldsymbol{x}, t) = u^H(\boldsymbol{x}, t) + u^D(\boldsymbol{x}, t). \tag{2.82}$$

Here u^D satisfies the Dirichlet boundary conditions, $u^D(\partial\Omega_D) = g_D$, and the homogeneous function is equal to zero on the Dirichlet boundary, $u^H(\partial\Omega_D) = 0$. The weak form (2.81) can then be formulated as: Find $u^H \in \mathcal{U}^H$ such that,

$$\begin{aligned}\int_\Omega v\frac{\partial(u^H + u^D)}{\partial t}d\boldsymbol{x} - \int_\Omega vf(u^H + u^D)d\boldsymbol{x} = &-\int_\Omega \nabla v \cdot (\nabla u^H + \nabla u^D)d\boldsymbol{x} \\ &+ \int_{\partial\Omega_N} v\boldsymbol{g}_N \cdot \boldsymbol{n} d\boldsymbol{x}, \quad \forall v \in \mathcal{V}.\end{aligned} \tag{2.83}$$

Following a finite element discretisation procedure, the solution is expanded in terms of a globally C^0-continuous expansion basis ϕ_n that spans the finite dimensional solution space $\mathcal{U}^\delta$. We also decompose this expansion basis into the homogeneous basis functions ϕ_n^H and the basis functions ϕ_n^D having support on the Dirichlet boundary such that

$$u^\delta(\boldsymbol{x}, t) = \sum_{n\in\mathcal{N}^H} \phi_n^H(\boldsymbol{x})\hat{u}_n^H(t) + \sum_{n\in\mathcal{N}^D} \phi_n^D(\boldsymbol{x})\hat{u}_n^D(t). \tag{2.84}$$

[7] $f(u) = -\nabla \cdot \boldsymbol{F}(u)$ in the following.

Finally, employing the same expansion basis ϕ_n^H to span the test space $\mathcal{V}$, Eq. (2.83) leads to the semi-discrete system of ODEs

$$\left[\begin{array}{cc} \boldsymbol{M}^{HD} & \boldsymbol{M}^{HH} \end{array} \right] \frac{d}{dt} \left[\begin{array}{c} \hat{\boldsymbol{u}}^D \\ \hat{\boldsymbol{u}}^H \end{array} \right] = - \left[\begin{array}{cc} \boldsymbol{L}^{HD} & \boldsymbol{L}^{HH} \end{array} \right] \left[\begin{array}{c} \hat{\boldsymbol{u}}^D \\ \hat{\boldsymbol{u}}^H \end{array} \right] + \boldsymbol{\Gamma}^H + \hat{\boldsymbol{f}}^H \tag{2.85}$$

where

$$\begin{aligned}
\boldsymbol{M}^{HH}[n][m] &= \int_\Omega \phi_n^H \phi_m^H d\boldsymbol{x} && n \in \mathcal{N}^H, m \in \mathcal{N}^H, \\
\boldsymbol{M}^{HD}[n][m] &= \int_\Omega \phi_n^H \phi_m^D d\boldsymbol{x} && n \in \mathcal{N}^H, m \in \mathcal{N}^D, \\
\boldsymbol{L}^{HH}[n][m] &= \int_\Omega \nabla \phi_n^H \cdot \nabla \phi_m^H d\boldsymbol{x} && n \in \mathcal{N}^H, m \in \mathcal{N}^H, \\
\boldsymbol{L}^{HD}[n][m] &= \int_\Omega \nabla \phi_n^H \cdot \nabla \phi_m^D d\boldsymbol{x} && n \in \mathcal{N}^H, m \in \mathcal{N}^D, \\
\hat{\boldsymbol{f}}^H[n] &= \int_\Omega \phi_n^H f(u) d\boldsymbol{x} && n \in \mathcal{N}^H, \\
\boldsymbol{\Gamma}^H[n] &= \int_{\partial\Omega_N} \phi_n^H \boldsymbol{g}_N \cdot \boldsymbol{n} d\boldsymbol{x} && n \in \mathcal{N}^H.
\end{aligned}$$

This can be rewritten in terms of the unknown variable $\hat{\boldsymbol{u}}^H$ as

$$\frac{d\hat{\boldsymbol{u}}^H}{dt} = \left(\boldsymbol{M}^{HH}\right)^{-1} \left\{ - \left[\begin{array}{cc} \boldsymbol{L}^{HD} & \boldsymbol{L}^{HH} \end{array} \right] \left[\begin{array}{c} \hat{\boldsymbol{u}}^D \\ \hat{\boldsymbol{u}}^H \end{array} \right] + \boldsymbol{\Gamma}^H + \hat{\boldsymbol{f}}^H - \boldsymbol{M}^{HD} \frac{d\hat{\boldsymbol{u}}^D}{dt} \right\}, \tag{2.86}$$

which, in the absence of Dirichlet boundary conditions, simplifies to

$$\frac{d\hat{\boldsymbol{u}}}{dt} = -\boldsymbol{M}^{-1} \left(\boldsymbol{L}\hat{\boldsymbol{u}} - \boldsymbol{\Gamma} \right) + \boldsymbol{M}^{-1} \hat{\boldsymbol{f}}. \tag{2.87}$$

2.2.2 General Linear Method

We have just shown how to reduce a PDE to and ODE using the Method of Lines, i.e. from Eq. (2.79) we deduced the initial value problem reported in Eq. (2.87). At this point we can apply a time-integration method to propagate the equation in time. The General Linear Method (GLM) connects the two main time-integration schemes families, $i.e.$ the multi-step methods and the multi-stage methods. Linear multi-step methods use a collection of r input parameters from the previous time-levels to obtain the solution at the next time-level. Linear multi-stage methods approximate the solution at the new time-level by linearly combining the solution at s intermediate stages. The standard initial

value problem in autonomous form is represented by the ODE,

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \tag{2.88}$$

where $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^N$. The n^{th} step of the general linear method comprising of r steps and s stages is then formulated as:

$$\boldsymbol{Y}_i = \Delta t \sum_{j=1}^{s} a_{ij} \boldsymbol{F}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \leq i \leq s \tag{2.89a}$$

$$\boldsymbol{y}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij} \boldsymbol{F}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \leq i \leq r \tag{2.89b}$$

where $\boldsymbol{Y}_i$ are called the stage values and $\boldsymbol{F}_i$ are called the stage derivatives. Both quantities are related by the differential equation:

$$\boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Y}_i). \tag{2.89c}$$

The matrices $A = [a_{ij}]$, $U = [u_{ij}]$, $B = [b_{ij}]$, $V = [v_{ij}]$ are characteristic of a specific method, and as a result, each scheme can be uniquely defined by the partitioned $(s+r) \times (s+r)$ matrix

$$\begin{bmatrix} A & U \\ B & V \end{bmatrix}. \tag{2.90}$$

For a more concise notation, it is convenient to define the vectors $\boldsymbol{Y}, \boldsymbol{F} \in \mathbb{R}^{sN}$ and $\boldsymbol{y}_i^{[n-1]}, \boldsymbol{y}_i^{[n]} \in \mathbb{R}^{rN}$ as follows:

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{Y}_1 \\ \boldsymbol{Y}_2 \\ \vdots \\ \boldsymbol{Y}_s \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} \boldsymbol{F}_1 \\ \boldsymbol{F}_2 \\ \vdots \\ \boldsymbol{F}_s \end{bmatrix}, \quad \boldsymbol{y}^{[n-1]} = \begin{bmatrix} \boldsymbol{y}_1^{[n-1]} \\ \boldsymbol{y}_2^{[n-1]} \\ \vdots \\ \boldsymbol{y}_r^{[n-1]} \end{bmatrix}, \text{ and } \boldsymbol{y}^{[n]} = \begin{bmatrix} \boldsymbol{y}_1^{[n]} \\ \boldsymbol{y}_2^{[n]} \\ \vdots \\ \boldsymbol{y}_r^{[n]} \end{bmatrix}. \tag{2.91}$$

Using these vectors, it is possible to write Eq. (2.89a) and Eq. (2.89b) in the form

$$\begin{bmatrix} \boldsymbol{Y} \\ \boldsymbol{y}^{[n]} \end{bmatrix} = \begin{bmatrix} A \otimes I_N & U \otimes I_N \\ B \otimes I_N & V \otimes I_N \end{bmatrix} \begin{bmatrix} \Delta t \boldsymbol{F} \\ \boldsymbol{y}^{[n-1]} \end{bmatrix} \tag{2.92}$$

where I_N is the identity matrix of dimension $N \times N$ and $\otimes$ is the Kronecker product. Note that it is the first element of the input vector $\boldsymbol{y}^{[n-1]}$ and output vector $\boldsymbol{y}^{[n]}$ which represents

the solution at the corresponding time-level, *i.e.* $\boldsymbol{y}_1^{[n]} = \boldsymbol{y}_n = \boldsymbol{y}(t_0 + n\Delta t)$. The other sub-vectors $\boldsymbol{y}_i^{[n]}$ $(2 \leq i \leq r)$ refer to the approximation of an auxiliary set of parameters inherent to the scheme. These parameters can, for example, be comprised of solutions at earlier time-levels. Some examples of how common multi-step and multi-stage schemes can be represented in matrix-form are reported in Appendix B.

2.2.3 Implicit-Explicit GLM Extension

GLM is extended in this section to fit implicit-explicit (IMEX) schemes. IMEX schemes are used to time-integrate ODEs of the form

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(\boldsymbol{y}) + \boldsymbol{g}(\boldsymbol{y}), \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \tag{2.93}$$

where $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^N$ is typically a stiff term and $\boldsymbol{g} : \mathbb{R}^N \to \mathbb{R}^N$ is a non-linear function (or where $\boldsymbol{f}$ and $\boldsymbol{g}$ have different time-scales).

The main advantage of an IMEX method is that it combines two different type of schemes to time-integrate different operators. Practically, one would like to use an implicit scheme for the stiff term in order to avoid an excessively small time-step. At the same time an explicit integration of the non-linear term is preferred, as it avoids the expensive matrix-inversion deriving from an implicit treatment of the convective terms.

IMEX linear multi-step schemes and IMEX Runge-Kutta schemes can be unified into an IMEX general linear method formulation, *i.e.*

$$\boldsymbol{Y}_i = \Delta t \sum_{j=1}^{s} a_{ij}^{\mathrm{IM}} \boldsymbol{F}_j + \Delta t \sum_{j=1}^{s} a_{ij}^{\mathrm{EX}} \boldsymbol{G}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \leq i \leq s \tag{2.94a}$$

$$\boldsymbol{y}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{IM}} \boldsymbol{F}_j + \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{EX}} \boldsymbol{G}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{y}_j^{[n-1]}, \qquad 1 \leq i \leq r \tag{2.94b}$$

where the stage derivatives $\boldsymbol{F}_i$ and $\boldsymbol{G}_i$ are defined as:

$$\boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Y}_i), \quad \boldsymbol{G}_i = \boldsymbol{g}(\boldsymbol{Y}_i). \tag{2.94c}$$

Adopting a matrix formulation similar to Eq. (2.92), this can be written in the form

$$\left[\begin{array}{c} \boldsymbol{Y} \\ \hline \boldsymbol{y}^{[n]} \end{array}\right] = \left[\begin{array}{c:c|c} A^{\mathrm{IM}} \otimes I_N & A^{\mathrm{EX}} \otimes I_N & U \otimes I_N \\ \hline B^{\mathrm{IM}} \otimes I_N & B^{\mathrm{EX}} \otimes I_N & V \otimes I_N \end{array}\right] \left[\begin{array}{c} \Delta t \boldsymbol{F} \\ \Delta t \boldsymbol{G} \\ \hline \boldsymbol{y}^{[n-1]} \end{array}\right]. \tag{2.95}$$

The stiffly stable schemes used for the incompressible Navier-Stokes solver (which have an IMEX nature) can be formulated as a general linear method. Using the coefficients reported in Table 2.1 we obtain for example the following partitioned-matrix for the second-order variant:

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|ccc} \frac{2}{3} & 0 & \frac{4}{3} & -\frac{1}{3} & \frac{4}{3} & -\frac{2}{3} \\ \hline \frac{2}{3} & 0 & \frac{4}{3} & -\frac{1}{3} & \frac{4}{3} & -\frac{2}{3} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right] \quad \text{with} \quad \boldsymbol{y}^{[n]} = \left[\begin{array}{c} \boldsymbol{y}_n \\ \boldsymbol{y}_{n-1} \\ \Delta t \boldsymbol{F}_n \\ \Delta t \boldsymbol{F}_{n-1} \end{array}\right]. \tag{2.96}$$

where the values in the first two rows have been scaled with γ_0 compared to the values in Table 2.1. Some other examples are reported in appendix B. In **Algorithm 2** we show a pseudocode to illustrate the basic steps of the solution process for an IMEX scheme using *Nektar++* framework. Further details can be found in $Vos\ et\ al.$ (Vos et al. 2011).

2.2.4 Time-Dependent Boundary Conditions

The initial value problem described in Eq. (2.87) is a simplified version of our real reduced problem, i.e. the one described by Eq. (2.86). In the real case we intend to solve for the unknown degrees of freedom $\hat{u}^H$ knowing the imposed degrees of freedom $\hat{u}^D$. It clearly appears from Eq. (2.86) that not only the definition of the Dirichlet degrees of freedom is required, but also their time-derivative. Although the value $\hat{\boldsymbol{u}}^D$ of the Dirichlet boundary conditions would typically be given for arbitrary t, a prescription of its time rate-of-change $\dfrac{d\hat{\boldsymbol{u}}^D}{dt}$ is not usually available. In the following we demonstrate how we can remove this dependence.

We assume that an arbitrary implicit-explicit GLM is applied to Eq. (2.86), where we decide to treat the diffusion term and the time-derivative of the Dirichlet degrees of freedom implicitly. The convective part of the equation is treated explicitly[8].

[8]This is a common approach also used in the velocity-correction scheme presented in section 2.3.

input : the vector $\boldsymbol{y}^{[n-1]}$

output: the vector $\boldsymbol{y}^{[n]}$

for $i = 1$ **to** s **do**

`// calculate the temporary variable` $\boldsymbol{x}_i$

(1) $\boldsymbol{x}_i = \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{IM}} \boldsymbol{F}_j + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \boldsymbol{G}_j + \sum_{j=1}^{r} u_{ij} \boldsymbol{y}_j^{[n-1]}$

`// calculate the stage value` $\boldsymbol{Y}_i$

(2) solve $\left(\boldsymbol{Y}_i - a_{ii}^{\mathrm{IM}} \Delta t \boldsymbol{f}(\boldsymbol{Y}_i) \right) = \boldsymbol{x}_i$

`// calculate the explicit stage derivative` $\boldsymbol{G}_i$

(3) $\boldsymbol{G}_i = \boldsymbol{g}(\boldsymbol{Y}_i)$

`// calculate the implicit stage derivative` $\boldsymbol{F}_i$

(4) $\boldsymbol{F}_i = \boldsymbol{f}(\boldsymbol{Y}_i) = \frac{1}{a_{ii}^{\mathrm{IM}} \Delta t} \left(\boldsymbol{Y}_i - \boldsymbol{x}_i \right)$

end

for $i = 1$ **to** r **do**

`// calculate` $\boldsymbol{y}_i^{[n]}$

(5) $\boldsymbol{y}_i^{[n]} = \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{IM}} \boldsymbol{F}_j + \Delta t \sum_{j=1}^{s} b_{ij}^{\mathrm{EX}} \boldsymbol{G}_j + \sum_{j=1}^{r} v_{ij} \boldsymbol{y}_j^{[n-1]}$

end

Algorithm 2: IMEX scheme solution process - pseudocode.

The $i-th$ stage, which we denote as $\hat{\boldsymbol{u}}_i^H$ for convenience, can be written as

$$\begin{aligned} \hat{\boldsymbol{u}}_i^H =& \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \left[\left(\boldsymbol{M}^{HH} \right)^{-1} \left(\hat{\boldsymbol{f}}_j^H - \boldsymbol{M}^{HD} \frac{d\hat{\boldsymbol{u}}^D}{dt}\bigg|_j \right) \right] \\ &+ \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \left[\left(\boldsymbol{M}^{HH} \right)^{-1} \hat{\boldsymbol{g}}_j^H \right] + \sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{H[n-1]}, \end{aligned} \tag{2.97}$$

where we have made the substitution

$$\hat{\boldsymbol{f}}_j^H = - \begin{bmatrix} \boldsymbol{L}^{HD} & \boldsymbol{L}^{HH} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}_j^D \\ \hat{\boldsymbol{u}}_j^H \end{bmatrix} + \boldsymbol{\Gamma}_j^H. \tag{2.98}$$

If we recognise that the variable $\hat{\boldsymbol{u}}^D$ satisfies the ODE

$$\left(\hat{\boldsymbol{u}}^D \right)' = \frac{d\hat{\boldsymbol{u}}^D}{dt}, \tag{2.99}$$

we can apply the same GLM to this ODE as the one we have used for the original ODE in terms of $\hat{\boldsymbol{u}}^H$, ($i.e.$ Eq. (2.97)), to arrive at

$$\hat{\boldsymbol{u}}_i^D = \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \left.\frac{d\hat{\boldsymbol{u}}^D}{dt}\right|_j + \sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{D[n-1]}, \tag{2.100}$$

where we do not have any explicit stage derivatives (or more precisely, $\boldsymbol{G}_j = 0$) due to the fact that we choose to treat the right-hand-side term $\dfrac{d\hat{\boldsymbol{u}}^D}{dt}$ in Eq. (2.99) implicitly. However, this is an arbitrary choice and we could have chosen to treat this term explicitly in both Eq. (2.97) and Eq. (2.99). The dependence from $\dfrac{d\hat{\boldsymbol{u}}^D}{dt}$ can be eliminated substituting Eq. (2.100) into Eq. (2.97), yielding

$$\begin{aligned}\hat{\boldsymbol{u}}_i^H =& \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \left[\left(\boldsymbol{M}^{HH}\right)^{-1} \hat{\boldsymbol{f}}_j^H\right] + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \left[\left(\boldsymbol{M}^{HH}\right)^{-1} \hat{\boldsymbol{g}}_j^H\right] \\ &+ \left(\boldsymbol{M}^{HH}\right)^{-1} \boldsymbol{M}^{HD} \left[\sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{D[n-1]} - \hat{\boldsymbol{u}}_i^D\right] + \sum_{j=1}^{r} u_{ij} \hat{\boldsymbol{u}}_j^{H[n-1]}.\end{aligned} \tag{2.101}$$

This can also be rewritten as

$$\begin{aligned}\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_i^H + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_i^D =& \Delta t \sum_{j=1}^{i} a_{ij}^{\mathrm{IM}} \hat{\boldsymbol{f}}_j^H + \Delta t \sum_{j=1}^{i-1} a_{ij}^{\mathrm{EX}} \hat{\boldsymbol{g}}_j^H \\ &+ \sum_{j=1}^{r} u_{ij} \left[\boldsymbol{M}^{HH} \hat{\boldsymbol{u}}_j^{H[n-1]} + \boldsymbol{M}^{HD} \hat{\boldsymbol{u}}_j^{D[n-1]}\right].\end{aligned} \tag{2.102}$$

For brevity we avoid all the implementation and technical details which have been reported in (Vos et al. 2011). Further information can also be found in (Kirby & Sherwin 2006*b*).

2.2.5 Verification of the Algorithm

A bi-dimensional linear-advection-diffusion equation has been used as a test case for the IMEX scheme implementation. In this case, as shown before, the diffusion operator has been treated implicitly and the advection term explicitly.

$$\frac{\partial u}{\partial t} + \alpha_x \frac{\partial u}{\partial x} + \alpha_y \frac{\partial u}{\partial y} = \nu \nabla u \tag{2.103}$$

The exact solution of Eq. (2.103), setting $\nu = 1$, is:

$$u = e^{-2\pi^2 t} \sin(\pi(x - \alpha_x t)) \sin(\pi(y - \alpha_y t)) \tag{2.104}$$

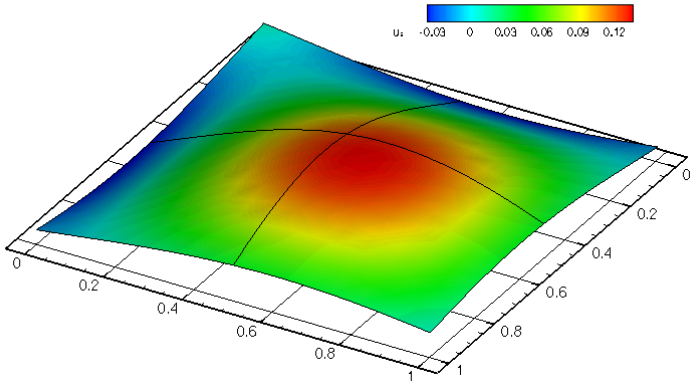


Figure 2.7: Numerical solution of a bi-dimensional linear-advection diffusion equation using a spectral/*hp* element method (4 elements and $P = 9$) and an IMEX scheme for time-integration.

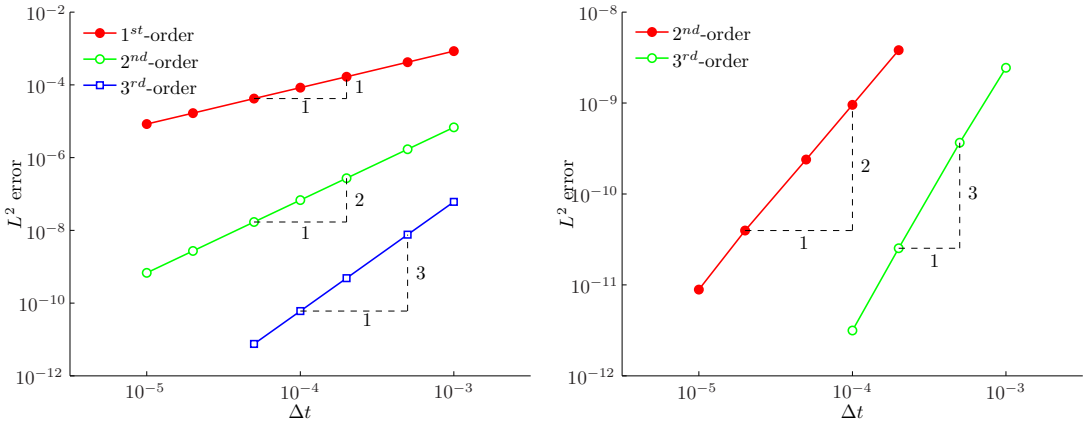
In **Algorithm 3** we show a pseudo-code which illustrates the basic set up of the time-integration procedure in *Nektar++* .

Fig. 2.7 illustrates the solution u obtained with the third order multi-step IMEX scheme with $P = 9$ and Dirichlet boundary conditions (calculated using the exact solution). The domain is a square of size $\boldsymbol{x} \in [0\,,1] \times [0\,,1]$ discretised with 4 quadrilaterals. The advection coefficients are $\alpha_x = \alpha_y = 1$.

YourClass solver(inputs) `// your solver`
TimeIntegrationMethod $SCHEME$ `// the scheme`
TimeIntegrationSchemeOperators ODE `// the operators`
`// using functors we identify the methods to be used`
ODE.DefineOdeRhs(&YourClass::YourExplicitOperatorFunction,solver)
ODE.DefineProjection(&YourClass::YourProjectionFunction,solver)
ODE.DefineImplicitSolve(&YourClass::YourImplicitOperatorFunction,solver)
`// setting the time-stepping scheme`
$SCHEME$ = eIMEX1
TimeIntegrationSchemeKey IntKey($SCHEME$)
numMultiSteps=1
TimeIntegrationSchemeSharedPtr IntegrationSchemes[numMultiSteps]
TimeIntegrationSolutionSharedPtr ODEsolution
IntegrationSchemesr[0] = TimeIntegrationSchemeManager()[IntKey]
ODEsolution = IntegrationSchemes[0]→InitializeScheme(Δt,U,t_0 ,ODE)
`// time-stepping`
for $i = 0$ **to** N_{steps} **do**
  U = IntScheme[0]-→TimeIntegrate(Δt,ODEsolution,ODE)
end

Algorithm 3: Pseudo-implementation of an unsteady solver in *Nektar++* .

Some tests have been undertaken to check the effective time-convergence of the IMEX multi-step and multi-stage schemes. In Fig. 2.8 the convergence ratio of the error decreasing the time-step Δt is shown. The figure highlights the converge ratio of schemes.



(a) Multi-Step (b) Multi-Stage

Figure 2.8: IMEX schemes converge rate with Δt for an unsteady advection-diffusion problem.

2.3 Incompressible Flows

Considering an incompressible, isothermal flow with constant density and viscosity, the governing equations are the incompressible Navier-Stokes equations, which dictate conservation of mass and conservation of momentum along the three dimensions. In terms of primitive variables $(\boldsymbol{V}, p)$, the equation are written as

$$\frac{\partial \boldsymbol{V}}{\partial t} + \boldsymbol{V} \cdot \nabla \boldsymbol{V} = -\nabla p + \nu \nabla^2 \boldsymbol{V} \tag{2.105}$$

$$\nabla \cdot \boldsymbol{V} = 0 \tag{2.106}$$

where p is the kinematic pressure field, ν is the kinematic viscosity and $\boldsymbol{V} = [u, v, w]^\top$ the velocity vector.

In the following we describe the numerical approach used to solve this set of equations, where no turbulence models are used. Therefore all the simulations presented are considered Direct Numerical Simulations (DNS) and appropriate resolution is necessary to resolve all the length scales.

2.3.1 Velocity Correction Scheme

The first issue when the Navier-Stokes system has to be solved is to decide in which way we want to deal with the velocity-pressure coupling. Various approaches can be used, starting from the coupled methods (e.g. $Uzawa$ algorithm) passing to methods that perform a change of variables (e.g. velocity-vorticity formulation and streamfunction-vorticity formulation). There is also a third approach where the Navier-Stokes system is split into a series of decoupled equations for the pressure and the velocity.

In *Nektar++* a *stiffly stable splitting scheme* in primitive variables is adopted, as presented in the work of $Karniadakis$, $Israeli$ and $Orszag$ (Karniadakis et al. 1991). The splitting scheme decouples the velocity field $\boldsymbol{V}$ from the pressure p, leading to an explicit treatment of the advection term and an implicit treatment of the pressure and the diffusion terms. Eqs. (2.107a) to (2.107d) show the steps taken to solve the incompressible Navier-Stokes equations. The values of the coefficients γ_0 , α_q and β_q of the multi-step implicit-explicit schemes are given in Table 2.1 for orders 1-3.

Eq. (2.107a) describes the first step of the scheme, which consists of calculating the advection term explicitly and combining it with the solution at previous time-steps, to create the first intermediate field $\hat{\boldsymbol{V}}$, and denotes J the order of the time-stepping scheme. The pressure solution at the new time level is obtained by solving a Poisson Eq. (2.107b) with consistent boundary conditions – Eq. (2.107e). In the third step of the scheme the second intermediate field $\hat{\hat{\boldsymbol{V}}}$ is calculated as shown in Eq. (2.107c). This intermediate field is then used as a forcing term in the Helmholtz problem which results from the reformulation of Eq. (2.107d). The divergence-free constraint is introduced into the splitting scheme via $\hat{\hat{\boldsymbol{V}}}$. In fact the Poisson equation for the pressure is obtained taking the divergence of Eq. (2.107c) under the assumption that $\nabla \cdot \hat{\hat{\boldsymbol{V}}} = 0$. The divergence-free condition is therefore introduced implicitly via Eq. (2.107c) and Eq. (2.107b) and reinforced by the high-order pressure boundary condition as reported in (Karniadakis et al. 1991). However, the new velocity field $\boldsymbol{V}^{n+1}$ is not divergence-free. As clarified in (Karniadakis et al. 1991), the resulting velocity field is affected by a bounded divergence error.

$$\frac{\hat{\boldsymbol{V}} - \sum_{q=0}^{J-1} \alpha_q \boldsymbol{V}^{n-q}}{\Delta t} = -\sum_{q=0}^{J-1} \beta_q [(\boldsymbol{V} \cdot \nabla)\boldsymbol{V}]^{n-q} \tag{2.107a}$$

$$\nabla^2 p^{n+1} = \nabla \cdot \left(\frac{\hat{\boldsymbol{V}}}{\Delta t}\right) \tag{2.107b}$$

$$\frac{\hat{\hat{\boldsymbol{V}}} - \hat{\boldsymbol{V}}}{\Delta t} = -\nabla p^{n+1} \tag{2.107c}$$

$$\frac{\gamma_0 \boldsymbol{V}^{n+1} - \hat{\hat{\boldsymbol{V}}}}{\Delta t} = \nu \nabla^2 \boldsymbol{V}^{n+1} \tag{2.107d}$$

$$\frac{\partial p}{\partial n}^{n+1} = -\Big[\frac{\partial \boldsymbol{V}}{\partial t}^{n+1} + \nu \sum_{q=0}^{J-1} \beta_q (\nabla \times \omega)^{n-q} + \sum_{q=0}^{J-1} \beta_q [(\boldsymbol{V} \cdot \nabla)\boldsymbol{V}]^{n-q}\Big] \cdot n \tag{2.107e}$$

In Eq. (2.107e) the time derivative of the velocity field at the new time level $n+1$ is required ($\omega = \nabla \times \boldsymbol{V}$). While a standalone time-integration is possible, it is not the most efficient choice. As reported in (Blackburn & Sherwin 2004), the whole high-order pressure boundary condition $\frac{\partial p}{\partial n}^{n+1}$ can be extrapolated starting from its values at previous time levels. This approach implies that the time-derivative of the velocity $\frac{\partial \boldsymbol{V}}{\partial t}^{n+1}$ is explicitly calculated using the same coefficients β_q used for the other terms in Eq. (2.107e). This can be written as

$$\frac{\partial p}{\partial n}^{n+1} = -\Big[\frac{1}{\Delta t}\sum_{q=0}^{J-1} \beta_q (\boldsymbol{V})^{n-q} + \nu \sum_{q=0}^{J-1} \beta_q (\nabla \times \omega)^{n-q} + \sum_{q=0}^{J-1} \beta_q [(\boldsymbol{V} \cdot \nabla)\boldsymbol{V}]^{n-q}\Big] \cdot n. \tag{2.108}$$

Since the summations representing the three extrapolations are identical, we can rewrite Eq. (2.108) as

$$\frac{\partial p}{\partial n}^{n+1} = -\sum_{q=0}^{J-1} \beta_q \Big\{\Big[\frac{1}{\Delta t}(\boldsymbol{V}) + \nu(\nabla \times \omega) + [(\boldsymbol{V} \cdot \nabla)\boldsymbol{V}]\Big] \cdot n\Big\}^{n-q}, \tag{2.109}$$

which illustrates the extrapolation process of $\frac{\partial p}{\partial n}^{n+1}$.

There is not a unique way to treat the convection operator appearing on the righthand side of Eq. (2.107a). Here we adopt the two classical forms defined in Table 2.2, although other forms such as the rotational form are also used (Karniadakis & Sherwin 2005).

The convective form requires a smaller number of operations with respect to the skew-symmetric one. However, in the case of turbulent simulations, the skew-symmetric form generally leads to smaller aliasing errors (Blaisdell et al. 1996), which often reduces the need to apply dealiasing techniques.

Table 2.1: Stiffly stable splitting scheme coefficients

	1st order	2nd order	3rd order
γ_0	1	$3/2$	$11/6$
α_0	1	2	3
α_1	0	$-1/2$	$-3/2$
α_2	0	0	$1/3$
β_0	1	2	3
β_1	0	-1	-3
β_2	0	0	1

Table 2.2: Advection term forms

Convective	$(\boldsymbol{V}\cdot\nabla)\boldsymbol{V}$
Skew-symmetric	$\frac{1}{2}\Big[(\boldsymbol{V}\cdot\nabla)\boldsymbol{V}+\nabla\cdot(\boldsymbol{V}\boldsymbol{V})\Big]$

2.3.2 Verification of the Algorithm

In this section we present some of the flow simulations which have been performed with *Nektar++* to validate the incompressible flow solver. Both 2D and 3D test cases have been considered and tested. For brevity we report here just few of them. All the selected cases are canonical and well known, hence we limit our post-processing, providing just the information required to demonstrate the correctness of the simulations and of the algorithms.

2.3.2.1 Kovasznay Flow

In 1948, Kovasznay solved the problem of steady, laminar flow behind a two-dimensional grid. This exact solution to the NS equations is given by:

$$u = 1 - e^{\lambda x}\cos 2\pi y \tag{2.110a}$$

$$v = \frac{\lambda}{2\pi}e^{\lambda x}\sin 2\pi y \tag{2.110b}$$

$$p = \frac{1}{2}(1 - e^{2\lambda x}) \tag{2.110c}$$

$$\lambda = \frac{1}{2\nu} - \left[\frac{1}{4\nu^2} + 4\pi^2\right]^{\frac{1}{2}} \tag{2.110d}$$

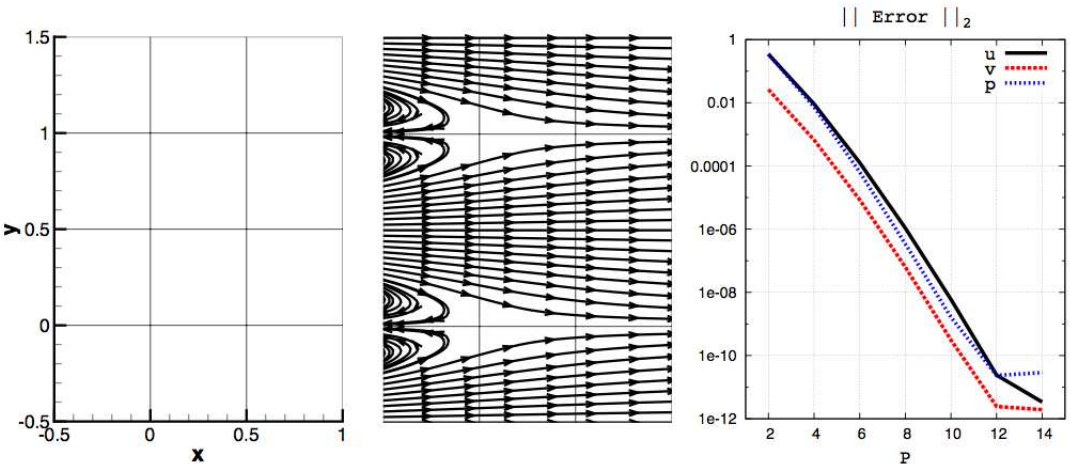


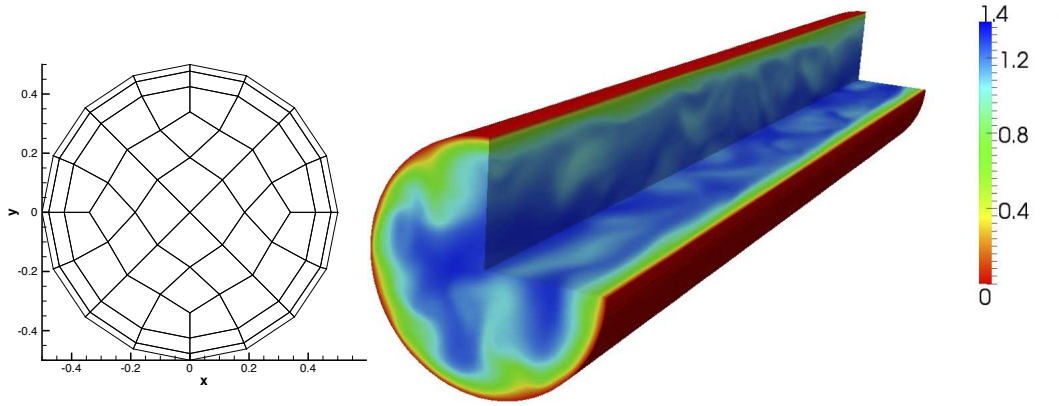
Figure 2.9: Solution of a 2D Kovasznay flow. From left to right: the 12-element mesh, the solution streamlines with $P = 7$ and the error convergence with P in the L^2 norm for the two velocity components u, v and the pressure field p.

In Fig. 2.9 the streamlines of the 2D solution obtained with *Nektar++* are shown. The rectangular domain is defined as $\boldsymbol{x} \in [-0.5\,, 1] \times [-0.5\,, 1.5]$ and it has been discretised with 12 quadrilateral elements. The solution looks similar to the low-speed flow of a viscous fluid past an array of cylinders. The Reynolds number is $Re = 1/\nu = 40$. Eq. (2.110a) to Eq. (2.110c) have been used to set Dirichlet boundary conditions for the flow variables. A study of the convergence features of the spectral/hp element method has been done in this case. The error converges exponentially to 10^{-12} for high polynomial expansions as shown in Fig. 2.9.

2.3.2.2 Turbulent Pipe Flow

Investigations of turbulence in pipes started with the experiment of $Reynolds$ in 1883 and have played a fundamental role in the study and understanding of turbulence. In this section we presented the DNS of a turbulent pipe at $Re_b \approx 3000$ based on the bulk velocity u_b, corresponding to $Re_\tau = 220$ based on the friction velocity u_τ. We refer and compare our results to the work of $McIver\ et\ al.$ (McIver et al. 2000). In this work a

very similar turbulent pipe ($Re_\tau = 443$) was studied using an identical numerical method. We also used the experimental results of *den Toonder et al.* (den Toonder & Nieuwstadt 1997) to ensure the correctness of our results. Although there is a small discrepancy between the values of Re_τ, the general behaviour of the flow is similar. The selection of this slightly reduced Reynolds number is due to practicality, since the simulation has been run in serial while developing and testing the *Nektar++* incompressible flows solver. The pipe we investigate has a diameter $D = 1$, a length $L = 5$ and it has been discretised in the cross section with 64 quadrilateral elements as shown in Fig. 2.10(a). In the flow direction we have used 128 Fourier modes with $k = 0, \ldots, 63$; therefore 128 grid points. On each element we utilise the nodal expansion basis described in Eq. (2.32) with $P = 7$ and a second order IMEX scheme with $\Delta t = 0.002$ to time step the Navier-Stokes equations. The advection term is solved explicitly using the skew-symmetric form and it is smoothed using the C^0 projection of the non-linear term at each time step (Blackburn & Sherwin 2004). In Fig. 2.10(b) the contours of the axial velocity are shown, which qualitatively indicate the turbulent nature of the flow. We assume a bulk velocity $u_b = 1$ defined as



(a) Cross-section Mesh (b) Axial Velocity

Figure 2.10: Turbulent pipe flow simulation at $Re_\tau = 220$ using the Fourier spectral/*hp* element method. (a) the $xy-$plane 2D mesh made of 64 quadrilaterals and (b) a contour plot of the axial velocity along the pipe. The fluid flows in $z-$direction.

$$u_b = \frac{1}{A}\int_A u(r)dA = \frac{1}{A}\int_0^{D/2}\int_0^{2\pi} u(r)rdrd\theta = 1, \tag{2.111}$$

where $A = \pi D^2/4$ is the area of the pipe cross-section and r the radius of the pipe[9]. Assuming $\nu = 1/3000$ we can define the Reynolds number based on the bulk velocity and the diameter as

$$Re_b = \frac{u_b D}{\nu} = 3000. \tag{2.112}$$

Since the Fourier expansion in $z-$direction is periodic by definition, we can not impose a pressure difference at the extremes of the pipe. Since we need to drive the flow in that direction, the pressure drop along the pipe is numerically imposed using a forcing term F. This forcing term is derived from $Blasius$ formula, which relates the pressure drop in a pipe to the Reynolds number as

$$F = \frac{\Delta p}{L} = \frac{1}{2} f \frac{u_b^2}{D} \tag{2.113}$$

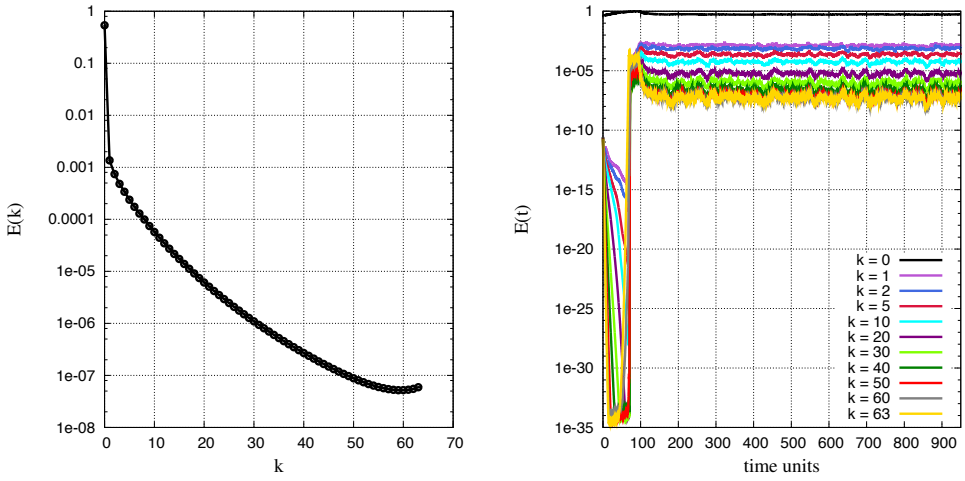
where

$$f = \frac{0.3164}{2Re_b^{0.25}}. \tag{2.114}$$

No-sip boundary conditions are imposed along the pipe walls for the three velocity components (homogeneous Dirichlet). For the pressure field, high-order boundary conditions are imposed as reported in Eq. (2.107e). The singularity of the Poisson equation for the $k = 0$ mode is solved by pinning one degree of freedom to an arbitrary chosen value - zero in this case. The initial condition is set using a plug condition for the axial velocity and normally distributed noise (with amplitude 0.001) is added to all the velocity components, in order to energise all the Fourier modes.

In Fig. 2.11 the modal energy distribution is shown with respect to both time and the Fourier mode number. Distribution of energy with respect to the mode number k is presented in Fig. 2.11(a) and the energy trend agree with what reported in (McIver et al. 2000). The bump in the energy profile which can be observed at $k \sim 60$ is due to aliasing effects. Fig. 2.11(b) depicts the energy time-trend associated with some of the Fourier modes where the transition to turbulence can be observed at $t \sim 70$. The second part of the the post-processing consists of averaging the axial velocity profile in time and space to produce a velocity distribution along the non-dimensional radius r/D and the viscous

[9]We are actually using a cartesian system, i.e. $r^2 = x^2 + y^2$.



(a) E(k) (b) E(t)

Figure 2.11: Modal energy distribution for a turbulent pipe flow simulation at $Re_\tau = 220$. In (a) the energy distribution respect to the Fourier modes frequency k averaged over 1000 time units and in (b) the modal energy behaviour with time, where the transition to turbulence can be observed at $t \sim 70$. Data reported in (a) show good agreement with what reported in (McIver et al. 2000).

wall unit y^+. Given the following turbulence common quantities

$$u_\tau = \sqrt{\frac{\tau_w}{\rho}} \qquad \tau_w = \frac{\Delta p D}{4L} \qquad y = \frac{D}{2} - r \tag{2.115}$$

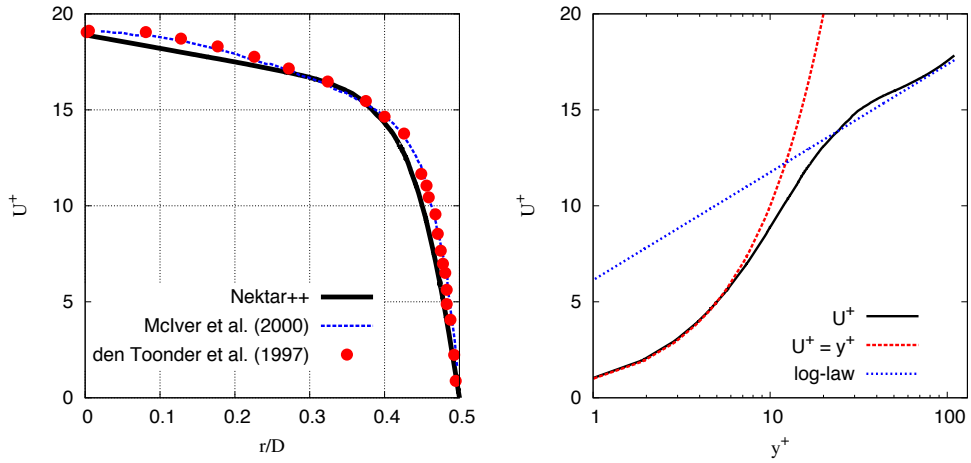
we can extrapolate the Reynolds number based on the friction velocity, which is

$$Re_\tau = \frac{D u_\tau}{\nu} = 220 \tag{2.116}$$

and the non-dimensional wall coordinate and velocity as

$$y^+ = \frac{y u_\tau}{\nu} \qquad U^+ = \frac{u}{u_\tau}. \tag{2.117}$$

Fig. 2.12(a) shows the non-dimensional velocity profile along the pipe radius as also reported in (McIver et al. 2000). In Fig. 2.12(b) we present the U^+ distribution as a function of the distance y^+ from the walls. We compare the averaged data obtained with *Nektar++* with the analytical curve representing the linear velocity distribution $U^+ = y^+$. To verify that the velocity distribution is consistent with a turbulent simulation we note that in Fig. 2.12(b), this linear velocity distribution, typical of the viscous sub-layer, matches with the numerical data for $y^+ < 5$. After the buffer layer we can observe the



(a) $U^+(r)$ (b) $U^+(y^+)$

Figure 2.12: Velocity profile in a turbulent pipe at $Re_\tau = 220$. In (a) the distribution of the non-dimensional velocity U^+ along the non-dimensional pipe radius r/D (solid line). The results are compared with the numerical results of $McIver\ et\ al$ (McIver et al. 2000) and the experimental results of $den\ Toonder\ et\ al.$ (den Toonder & Nieuwstadt 1997). Discrepancies are due to the imposition of a constant pressure gradient instead of a constant mass flow. In (b) U^+ is plotted against the viscous wall unit y^+. Results show good agreement with what reported in $Pope$'s text book (Pope 2000).

U^+ distribution matches also with the $log - law$ defined as

$$U^+ = \frac{1}{\mathcal{K}} \ln y^+ + C^+ \qquad \mathcal{K} = 0.41 \qquad C^+ = 6.13. \tag{2.118}$$

The values for the $Von\ Karman$ constant $\mathcal{K}$ and C^+ have been taken from the turbulence text book of $Pope$ (Pope 2000).

2.3.2.3 Turbulent Channel Flow

The DNS of turbulent channels has been one of the most studied numerical experiments in the last six decades. The problem has two natural periodic dimensions. This feature has always encouraged practitioners to discretise the physical domain using a Fourier spectral method in the two directions non-normal to the walls. $Kim\ et\ al.$ in 1987 presented a detail characterisation of a DNS at $Re_\tau = 180$ (Kim et al. 1987) using a Fourier spectral method in the two periodic dimensions (x and z) and a Chebyshev polynomial expansion in y, i.e. perpendicularly to the walls. In this section we present similar investigation,

but we use a Fourier spectral method just in the $z-$direction and a spectral/*hp* element method to discretise the $xy-$plane, as reported in Fig. 2.13.

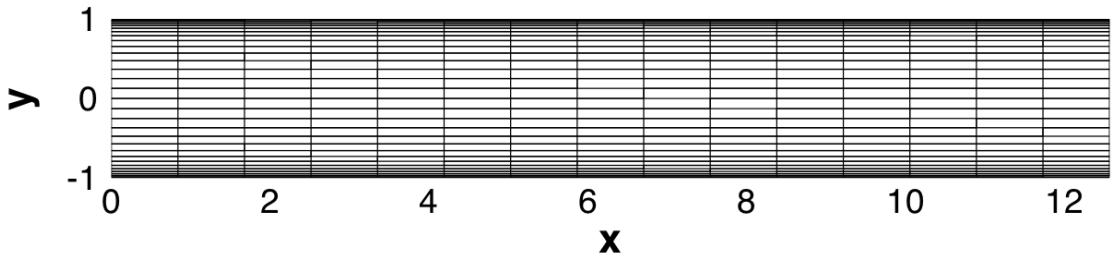
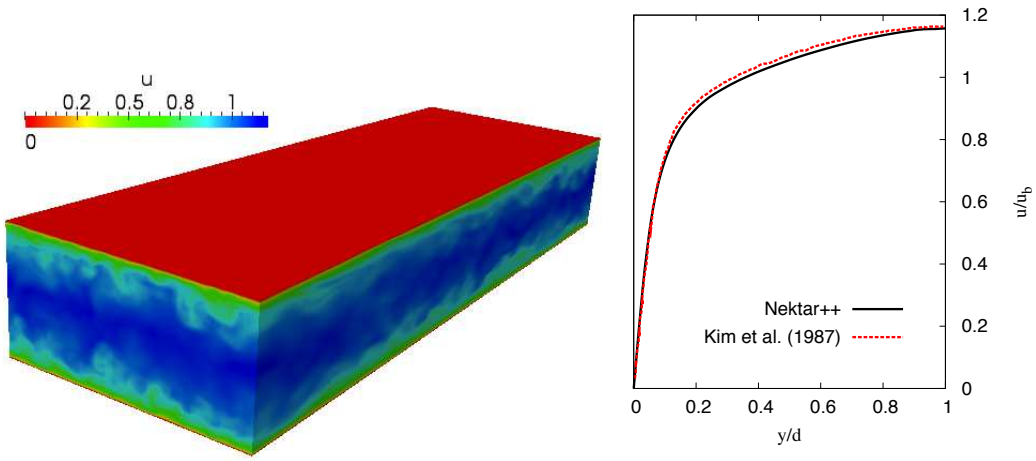


Figure 2.13: Bi-dimensional mesh used to discretise the turbulent channel flow at $Re_\tau = 180$. The 2D mesh replicates the mesh used by $Koberg$ (Koberg 2007). Extension in $z-$direction is obtained with a 64-modes Fourier expansion.

The domain size is chosen to be $x \in [0, 4\pi]$, $y \in [-1, 1]$ and $z \in [0, 4\pi/3]$ as in the work of $Koberg$ (Koberg 2007). In his work $Koberg$ discretised the xy-plane with 450 quadrilaterals elements, 15 in the streamwise direction x and 30 in the spanwise direction y. He demonstrated the discretisation was fine enough to capture turbulent features, hence we use the same spatial discretisation, shown in Fig. 2.13. A nodal basis ($P = 6$) has been used in combination with a second order stiffly-stable time integration scheme ($\Delta t = 0.001$). Pressure and velocity are set to be periodic along x and the flow is driven by a numerical forcing term defined as in Eq. (2.113), where the friction factor is taken from the reference text book of $Pope$ (Pope 2000). Zero Dirichlet boundary conditions are set for the velocity field on the walls in combination with high-order boundary condition for the pressure. The advection term is treated using the skew-symmetric form and the spectral vanishing viscosity technique (Kirby & Sherwin 2006*a*) has been used to stabilise the simulation across the transition to turbulence. The Fourier spectral discretisation consists of a 64-mode Fourier expansion.

In Fig. 2.14(a) we present the contour of the axial velocity in the channel where we can see the turbulent nature of the flow. The mean velocity profile in the streamwise direction is depicted in Fig. 2.14(b) where the results are compared with those reported in (Kim et al. 1987).

In order to perform some further statistical analysis, we define the fluctuations of the



(a) Axial velocity (b) Mean velocity profile

Figure 2.14: Axial velocity for a turbulent channel simulation at $Re_\tau = 180$. In (a) the axial velocity contours are presented and in (b) the mean axial velocity profile is plotted agains the non-dimensional distance from the wall. Results are compared with the numerical experiment reported in (Kim et al. 1987).

$i-th$ velocity component as

$$u_i' = u_i - \bar{u}_i \tag{2.119}$$

where $\bar{u}_i$ is the mean value of the components, defined as

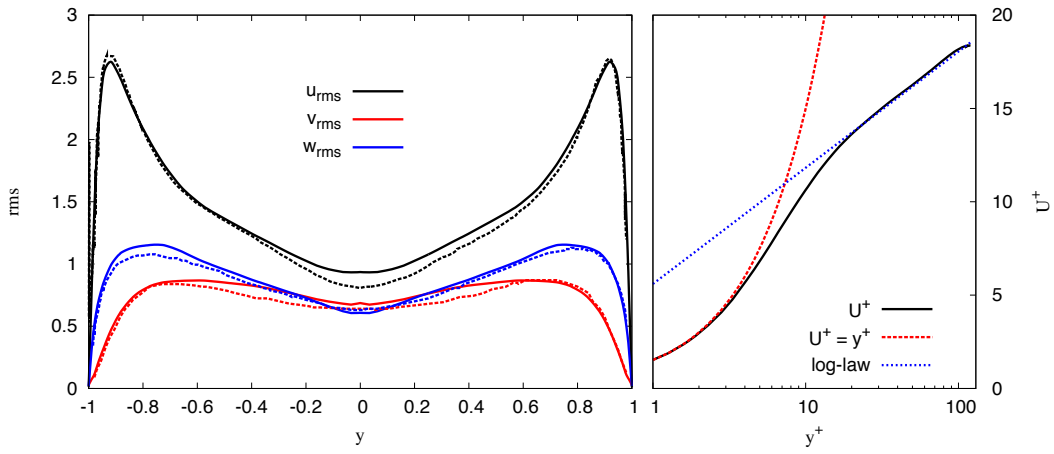
$$\bar{u}_i = < u_i > = \frac{1}{M}\sum_{j=0}^{M}(u_i)_j \tag{2.120}$$

and M in the number of time-samples. We specify the variance of the fluctuations as

$$< u_i'u_i' > = \frac{1}{M}\sum_{j=0}^{M}(u_i'u_i')_j. \tag{2.121}$$

In Fig. 2.15(a) we present the square root of the variance ($i.e.$ the rms) for each velocity component (solid lines) and we compare them with the results of $Kim\ et\ al.$ (Kim et al. 1987) (dashed lines). Fig. 2.15(b) shows the mean value of the non-dimensional axial velocity profile U^+ compared with what is reported in (Pope 2000) as we previously did for the pipe flow. The values for the log-law constants $\mathcal{K}$ and C^+ have been taken from (Pope 2000) as

$$U^+ = \frac{1}{\mathcal{K}}\ln y^+ + C^+ \qquad \mathcal{K} = 0.4 \qquad C^+ = 5.5. \tag{2.122}$$



(a) $< u_i' u_i' >^{1/2}$ (b) $U^+(y^+)$

Figure 2.15: In (a) the three components rms velocity fluctuations (solid lines) compared with the results of $Kim\ et\ al.$ (Kim et al. 1987) (dashed lines). Discrepancies are due to the imposition of a constant pressure gradient instead of a constant mass flow. In (b) the non-dimensional velocity U^+ behaviour along the viscous wall units compared with what reported in (Pope 2000).

We observed good agreement between the numerical simulation performed with *Nektar++* and the reference case (Kim et al. 1987), showing that the discretisation is clearly capturing the features of the flow correctly.

2.4 Discussion

In this chapter we briefly presented the theoretical background related to the numerical strategies we have implemented and used in the studies which follow in the next chapters. All of the numerical techniques presented here have been implemented within the *Nektar++* framework, but the C++ implementation details have been omitted for brevity. A technical description of the full framework and routines can be found on the *Nektar++* website (Kirby & Sherwin 2006*b*). However, we completed the description of each section with verification to demonstrate that the algorithms implemented work as intended.

We described the basics of the numerical methods we implemented for the 2D spatial discretisation and the time-integration. The 2D spatial discretisation allows an easy switch between various elemental shape and basis type. It is also designed to encourage

the use of different computational approaches to perform operations locally and globally, as also reported in (Vos 2010). Furthermore, the encapsulation of the building blocks of a spectral/*hp* element method allows an easy switch between continuous and discontinuous Galerkin projection; both optimised to work with high and low order polynomial expansions. A generic formulation that facilitates the usage of a wide range of time-integration methods via a unique interface has also been presented. The algorithm is based on the General Linear Method theory, well known within the ODE community although rarely applied by PDE solvers. This unified implementation should allow users to explore the variety of methods that exist to solve an unsteady problem. The specific implementation should also push the users to try new methods, since it is as quick as filling a new coefficients matrix. The high flexibility and efficiency achieved in the spatial and temporal discretisation will be used in Chapter 3, where we will investigate optimal combinations of space-time discretisations to solve an hyperbolic problem.

The 3D extension, in which we combine a 2D spectral/*hp* element method and a Fourier spectral discretisation, will be used in Chapter 4 to investigate parallelisation strategies. The implementation of the 3D Fourier spectral/*hp* element method and of the incompressible Navier-Stokes solver will be tested along with different approaches in term of communications and domain decomposition. Exploiting the needs of various use cases, we intend to create some guidelines when parallelising a turbulent problem. The focus on turbulent simulation required also the implementation of various stabilisation techniques, which we omit for brevity but which can be found in (Kirby & Sherwin 2006*b*).

Chapter 3

Time-Stepping Strategies

High-order spectral/*hp* element methods, utilising element-wise polynomial spaces of order $P \geq 1$, are gaining prominence for the efficient discretisation of time-dependent problems. The exponential convergence of the solution with increasing polynomial order results in lower numerical errors for the same number of degrees of freedom when compared with linear finite element methods (Karniadakis & Sherwin 2005, Gottlieb & Orszag 1977). As a consequence, long time-integration can potentially be achieved more accurately and more efficiently than may be possible with traditional low-order methods.

While the numerical properties of high-order methods for sufficiently smooth solutions are now widely recognised in the asymptotic limit, the choice of discretisation parameters to achieve a given numerical error in the most computationally efficient manner are not as effectively understood. Unlike discretizations for linear finite element methods, those for high-order techniques can be considered a function of both mesh element size (h) and polynomial order (P), which greatly enriches the space of possible spatial dis-

cretizations. Furthermore, the element-wise data locality of these methods has the consequence that traditional operator implementation techniques for low-order finite element methods, where elemental matrices are coalesced into a single large sparse global matrix, may not be the most efficient approach when dealing with higher polynomial orders. For example, local operator implementation using an element-by-element approach has been shown to be more computationally efficient in two dimensions (Vos et al. 2010) on CPUs, with the performance difference being more pronounced in three dimensions (Cantwell et al. 2011*b*), when using a continuous Galerkin projection. GPUs are more efficient when there is limited indirection, hence the local element-by-element approach is the best choice even for linear finite element methods (Markall et al. 2013). Sum-factorisation (Orszag 1980) exploits the tensor-product nature of the high-order elemental construction to cast the elemental operations as a sequence of smaller matrix-matrix products which improves the efficiency still further for very high polynomial orders. As a consequence, understanding the computational efficiency of these different implementation strategies and hardware choices across the space of possible discretizations is non-trivial.

With knowledge of the most efficient technique with which to apply an operator for a specific polynomial order, one might then ask what the optimal choice of discretisation should be to achieve a given solution accuracy at the minimal computational cost (Cantwell et al. 2011*a*). In this case runtime is now a function of both mesh element size and polynomial order, and there exists a subspace of possible discretizations which satisfy the error constraint, from which we seek the minimum runtime.

Given a time-dependent problem to solve with a prescribed accuracy on the final solution, we would like to establish the combination of discretisation parameters, operator implementation and time integration scheme which minimises the solution time. It is commonly understood that achieving accurate solutions when integrating over long time periods requires the use of high-order time integration schemes. However, for shorter time integration periods spatial errors may dominate, so it is important to understand when high-order schemes are appropriate and when lower order schemes will suffice and offer the best performance.

As reported in (Bolis et al. 2013), we extend previous mentioned studies by identifying general trends for the optimal selection of spatial and temporal discretisation for time-

dependent problems. When integrating in time, the efficiency of the algorithm depends not just on the implementation of the spatial operator and its cost per application, but also on the number of time steps needed to reach the desired final time and the cost of each step. In case for example of explicit time-stepping methods the number of time steps is related to the discretisation through the CFL condition, which restricts the size of the time-step based on the eigenspectrum of the discretised spatial operator. The stability region of the chosen time integration scheme must enclose all eigenvalues of this operator to ensure numerical stability.

In this study we use a discontinuous Galerkin projection and thus restrict ourselves to considering the local matrix and sum-factorisation approaches. We also restrict our numerical investigation to a rotating Gaussian transported under a 2D hyperbolic unsteady linear advection problem on a square domain with upwinded Dirichlet boundary conditions. While this test problem is not necessarily representative of the complexity of typical fluid-flow applications, it is sufficiently non-trivial to establish basic trends which can be applied to other more complex PDE problems and will highlight the most important aspects of the spatial and temporal discretisation.

3.1 Application to Fluid Dynamics

Computational efficiency, numerical stability and accuracy are fundamental aspects when solving the equations typical of fluid dynamics, such as the incompressible Navier-Stokes equations reported in section 2.3. Nevertheless, it is often unpractical to analyse these numerical features on the problem of interest. In fact, the complexity of the equations introduces some difficulties in understanding the overall computational characteristics of a numerical approach. Hence, it is common practice to separately investigate the various terms composing the numerical model. When analysing the performance of a single term of a model, the main task is to reproduce accurately the computational load and the numerical features of this component as a standalone/reduced problem.

The unsteady linear-advection problem is commonly used to investigate the numerical features of discretisations in case of convective-dominate equations. This problem

is also fairly representative of the convective term treatment in many projection algorithms implied in the solution of the incompressible Navier-Stokes equation (Karniadakis & Sherwin 2005). In fact, the number of operations required at each time-step to perform the advection calculation does not change from the linear to the non-linear case. This is because the advection calculation is generally performed in a collocation fashion, hence it reduces to a vector-vector multiplication after the spatial derivatives have been calculated. An overview of projection methods for the incompressible Navier-Stokes equations can be found in (Guermond & Minev 2006).

The projection method we reported in section 2.3.1 is a practical example of how the convective term is generally treated when incompressible flows are solved decoupling the velocity field from the pressure field. The explicit time-integration scheme adopted in this case is a tailored multi-step scheme constructed to fit the needs of the specific projection method (Karniadakis et al. 1991). However, we can theoretically use any explicit time-integration scheme to advance in time the Navier-Stokes non-linear term.

At this point we also need to consider the numerical features of our reduced model, $i.e.$ the unsteady linear-advection equation. Given that it replicates the number of operations per time-step of the non-linear case, we need also to reproduce the same characteristic in terms of numerical stability and accuracy on the solution. When explicitly time-stepping an hyperbolic equation, such as the unsteady advection equation, the first issue we encounter is the selection of an appropriate time-integration scheme. The advection operator deriving from an elemental CG discretisation is characterised by a purely imaginary eigenspectrum (Karniadakis & Sherwin 2005). To guarantee numerical stability, the stability region of the explicit scheme must encompass the imaginary axis (we will see it in more detail in section 3.5). In case of the velocity-correction scheme reported in section 2.3.1 the issue is naturally solved by the implicit-explicit time-integration coupling, which is characterised by an enlarged stability region that includes also the imaginary axis. To reinforce numerical-stability similarities between the approach adopted in the velocity-correction scheme and our standalone investigations we chose to apply a discontinuous Galerkin projection. This approach introduces a dumping effect that translates into a shift of the eigenvalues of advection operator. Therefore, in case of a DG projection, the eigenvalues are not purely imaginary (Sherwin 2000). As a consequence we

can perform our investigations using classical explicit time-integration schemes. These basic time-stepping schemes show a stability region which does not widely encompass the imaginary axis, as the explicit component of the IMEX scheme in section 2.3.1 would theoretically do.

Although the whole numerical method we use in this chapter is different from the numerical approach followed in section 2.3.1, it allows us to highlight the actual issues regarding computational efficiency. In fact we can reproduce quite accurately:

- the number of operations per time-step;
- the numerical stability constraint deriving from the CFL condition;
- the accuracy features and trends with respect to the spatial discretisation.

In addition, the investigation performed using non-uniform meshes highlights the practical issues arising in real CFD applications, where some very small elements are required somewhere in the domain to properly capture the flow features.

3.2 Case of Study

We investigate the relative performance of a second-order Adams-Bashforth scheme and second- and fourth-order Runge-Kutta schemes when time-stepping a 2D linear advection problem discretised using a spectral/*hp* element technique for a range of different mesh sizes and polynomial orders. Numerical experiments explore the effects of short (2 wavelengths) and long (32 wavelengths) time integration for sets of uniform and non-uniform meshes. The choice of time-integration scheme and discretisation together fixes a CFL limit which imposes a restriction on the maximum time-step which can be taken to ensure numerical stability. The number of steps, together with the order of the scheme, affects not only the runtime but also the accuracy of the solution. Through numerical experiments we systematically highlight the relative effects of spatial resolution and choice of time integration on performance and provide general guidelines on how best to achieve the minimal execution time in order to obtain a prescribed solution accuracy. The significant role played by higher polynomial orders in reducing CPU-time while preserving

accuracy becomes more evident, especially for uniform meshes, compared to what has been typically considered when studying this type of problem.

The test problem considered is that of the 2D unsteady advection equation on a $[-1, 1]^2$ domain, in which an off-centred Gaussian function is advected about the origin under a constant rotational divergence-free velocity field $\mathbf{V}$. The problem is mathematically expressed as

$$\frac{\partial u}{\partial t} + \nabla \cdot \boldsymbol{F}(u) = 0, \tag{3.1a}$$

$$\boldsymbol{F}(u) = \boldsymbol{V}u, \tag{3.1b}$$

$$\nabla \cdot \boldsymbol{V} = 0, \tag{3.1c}$$

$$\boldsymbol{V} = [2\pi y, -2\pi x]^\top = [V_x, V_y]^\top, \tag{3.1d}$$

with the exact solution for all times t given by

$$u(x, y, t) = e^{-\alpha[(x-\beta\cos 2\pi t)^2 + (y-\beta\sin 2\pi t)^2])}. \tag{3.1e}$$

The parameters α and β govern the shape and position of the Gaussian function, respectively. They are fixed at

$$\alpha = 41, \quad \text{and} \quad \beta = 0.3, \tag{3.2}$$

in order to produce a Gaussian function with a standard deviation of $\sigma = 0.11$, passing through the domain in a prescribed circle of radius 0.3 centred at the origin. The Gaussian function attains a maximum value of $\mathcal{O}(10^{-9})$ on the domain boundary when the centre passes at its closest point, allowing the use of weakly-imposed zero-Dirichlet boundary conditions on all four edges. We explored the impact of the initial condition/boundary condition incompatibility issue; after examination, we concluded it does not affect the results presented in this study. The domain is discretised in space using high-order spectral/*hp* elements which are briefly described in the following section.

3.3 Discontinuous Galerkin Projection

As with other finite element methods, a domain Ω is decomposed into a set of non-overlapping elemental regions, Ω_e, such that $\Omega = \bigcup \Omega_e$. We consider only the case of

conformal meshes. Basic operations, such as differentiation or integration, are carried out on a reference element Ω_{st}, to which each physical element is mapped using an isoparametric coordinate mapping $\chi : \Omega_e \rightarrow \Omega_{st}$. In two dimensions, this maps the physical coordinates (x_1, x_2) of Ω_e onto the reference space coordinates (ξ_1, ξ_2) as mentioned in section 2.1.3. Within the reference space, a variable u is approximated via an expansion in terms of a set of N two-dimensional basis functions $\phi_n(\xi_1, \xi_2)$ as reported in section 2.1.3.4.

We now apply the method of weighted residuals with a Galerkin projection. We derive a weak formulation of our problem by multiplying Eq. (3.1a) by smooth test functions, v, and integrating over Ω to arrive at

$$\int_\Omega v \frac{\partial u}{\partial t} \, \mathrm{d}\boldsymbol{x} + \int_\Omega v \nabla \cdot \boldsymbol{F}(u) \, \mathrm{d}\boldsymbol{x} = 0. \tag{3.3}$$

Defining $\mathcal{P}^P(\Omega_e)$ as the space of polynomials of order P, the discrete approximation $u^\delta \in \mathcal{U}^\delta$ of the variable u and the discrete approximations of test functions $v^\delta \in \mathcal{V}^\delta$, where

$$\mathcal{U}^\delta = \{u \in (L^2(\Omega))^2 : u|_{\Omega_e} \in (\mathcal{P}^P(\Omega_e))^2, \forall \, \Omega_e \in \Omega\} \tag{3.4a}$$

$$\mathcal{V}^\delta = \{v \in (L^2(\Omega))^2 : v|_{\Omega_e} \in (\mathcal{P}^P(\Omega_e))^2, \forall \, \Omega_e \in \Omega\}, \tag{3.4b}$$

we arrive at the equivalent discrete weak formulation,

$$\int_{\Omega_e} v^\delta \frac{\partial u^\delta}{\partial t} \, \mathrm{d}\boldsymbol{x} + \int_{\Omega_e} v^\delta \nabla \cdot \boldsymbol{F}(u^\delta) \, \mathrm{d}\boldsymbol{x} = 0, \tag{3.5}$$

from which a matrix system can be constructed (see (Karniadakis & Sherwin 2005)).

For the discontinuous Galerkin method we require a mechanism for information to propagate across element boundaries without affecting the stability of the method. Applying the divergence theorem to the second integral of Eq. (3.5) we obtain

$$\int_{\Omega_e} v^\delta \frac{\partial u^\delta}{\partial t} \, \mathrm{d}\boldsymbol{x} + \int_{\partial\Omega_e} v^\delta \boldsymbol{F}(u^\delta) \cdot n \, \mathrm{d}\boldsymbol{s} - \int_{\Omega_e} \nabla v^\delta \cdot \boldsymbol{F}(u^\delta) \, \mathrm{d}\boldsymbol{x} = 0. \tag{3.6}$$

The coupling is therefore achieved through the boundary fluxes represented by the second integral in Eq. (3.6). The approach used to calculate these fluxes dictates the stability of the method. In this study we use an up-wind scheme. Defining u^δ_- to be the value of the solution u^δ on the boundary of a given element e and u^δ_+ to be the solution on the same

boundary of an adjacent element, the boundary flux, denoted with $\tilde{\boldsymbol{f}}^e(u^\delta_-, u^\delta_+)$, is defined as

$$\tilde{\boldsymbol{f}}^e(u^\delta_-, u^\delta_+) = \begin{cases} \boldsymbol{V} u^\delta_-, & \boldsymbol{V} \cdot \boldsymbol{n}^e \geq 0, \\ \boldsymbol{V} u^\delta_+, & \boldsymbol{V} \cdot \boldsymbol{n}^e < 0, \end{cases} \tag{3.7}$$

where $\boldsymbol{n}^e$ denotes the outward-pointing normal to the element. For more details concerning continuous and discontinuous Galerkin formulations and for the case of more complicated hyperbolic problems (where it may be necessary to use an approximated Riemann solver) see (Karniadakis & Sherwin 2005, Zienkiewicz et al. 2003, Hesthaven & Warburton 2008).

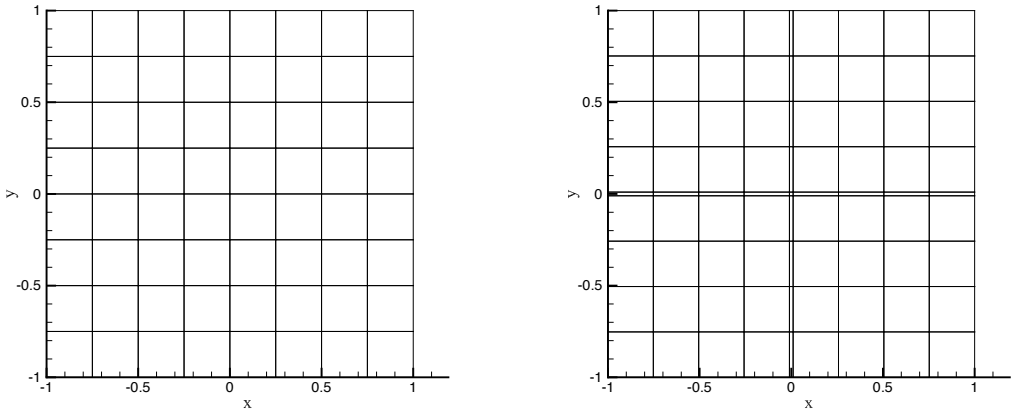
3.4 Domain discretisation

The domain $\Omega = [-1, 1]^2$ is discretised using a range of quadrilateral meshes of both a uniform and non-uniform nature. Uniform meshes are structured regular grids of $N \times N$ elements, where N is in the range $1, \ldots, 8$. An example is shown in Fig. 3.1(a) for $N = 8$. We also consider five non-uniform meshes which contain a mixture of small and large elements. For these we take the four uniform meshes where N is even and add a narrow vertical and horizontal band of elements of width $h = 0.01$ in the centre of the mesh, an example of which is shown in Fig. 3.1(b). Although this mesh contains 81 elements, for the purpose of comparison we denote this mesh as being *non-uniform* $N = 8$ since it is approximately equivalent to the $N = 8$ uniform mesh.

The Gaussian function given in Eq. (3.1e) at $t = 0$ is projected onto each mesh and used as an initial condition for the simulation. An example is shown in Fig. 3.2 and is the discretised form of the exact solution on the mesh shown in Fig. 3.1(a) with $P = 11$.

3.5 CFL control

The stability of an explicit time integration scheme is governed by the CFL condition (Hirsch 2007). The CFL condition is a stability condition which imposes that the space-time numerical domain of dependence has to include the analytical one. To formalise the definition of the CFL condition we consider a one-dimensional hyperbolic equation of the



(a) Uniform Mesh (b) Non-uniform Mesh

Figure 3.1: Examples of test meshes used in the study. A uniform mesh with 64 elements (a) and the equivalent non-uniform mesh (b) with 81 elements. The non-uniform mesh includes a narrow cross of elements in the centre of the mesh.

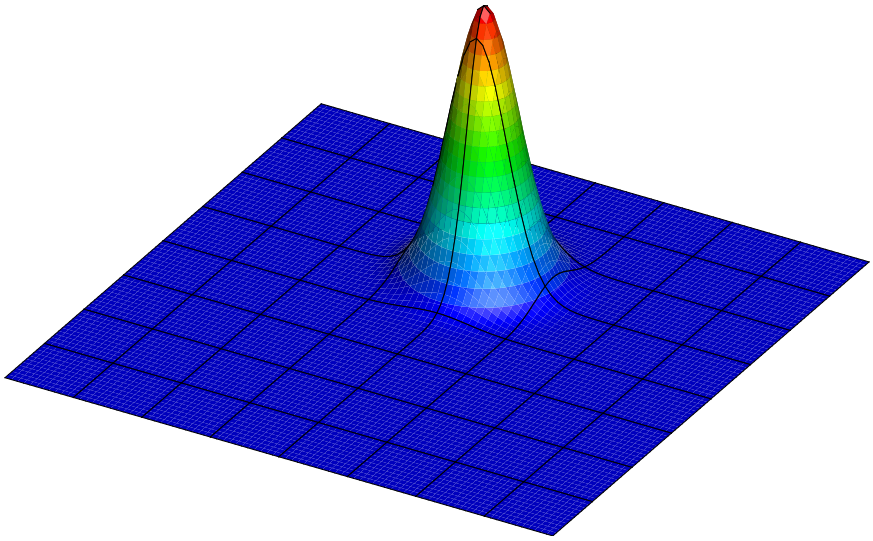


Figure 3.2: 2D unsteady advection problem, initial condition projected on 64 uniform elements with $P = 11$.

form

$$\frac{\partial u}{\partial t} + \boldsymbol{V}\frac{\partial u}{\partial x} = 0 \tag{3.8}$$

discretised in space using a grid Δx and in time with a time-step Δt. Given the definition of $Courant$ number, which is

$$C = \frac{\boldsymbol{V}\Delta t}{\Delta x}, \tag{3.9}$$

the CFL condition translates into defining the admissible values of C, which in turn means selecting Δx and Δt such that $C \leq C_{max}$. When using explicit time-stepping schemes the numerical stability is guaranteed if $C_{max} = 1$, therefore $0 < C \leq 1$. On the other hand C_{max} can be greater than 1 if we time-step the equation using an implicit schemes.

While the CFL condition can be easily defined and fulfilled in the canonical example of Eq. (3.8), in real CFD applications the scenario becomes more complex. When moving from one-dimensional to two- and three-dimensional problems not only the module of the convective velocity $\boldsymbol{V}$ is important, but also its direction with respect to the spatial discretisation, which can locally vary. Moreover the spatial discretisation may be not uniform ($e.g.$ non-uniformly discretised elemental approaches) making difficult to decide what is the actual value of Δx. In addition, when using an elemental discretisation, also the shape of the elements plays a role in defining the actual CFL restriction (Karniadakis & Sherwin 2005).

As reported in (Karniadakis & Sherwin 2005), when we are dealing with a non trivial spatial discretisation such as the spectral/*hp* element method, the CFL condition can be rewritten as

$$\Delta t \leq C \frac{\alpha}{|\lambda_{dom}|}. \tag{3.10}$$

In this relation α represents the value obtained at the intersection of the stability regions of the time-integration scheme with the dominating eigenvalue (λ_{dom}) of our spatial operator, $i.e.$ the advection operator. We also introduced C, which plays the role of the $Courant$ number. For the advection operator the value of λ_{dom} is a function of the discretisation itself, the convective velocity and the time-integration scheme we are using. For the spectral/*hp* element method it is recognised that $|\lambda_{dom}| = \boldsymbol{f}(\boldsymbol{V}, h, P)$, where both the module and the direction of the convective velocity $\boldsymbol{V}$ are important (Karniadakis & Sherwin 2005). The value of Δt can be interpreted as rescaling the stability region of the time-integration scheme and it can be written more formally as

$$\Delta t = C \inf_j \left\{ \frac{\alpha(\theta_j)}{r_j} : \lambda_j = r_j e^{\theta_j} \in \Lambda \right\}, \tag{3.11}$$

where C is conceptually similar to the $Courant$ number (stability is assured if $0 < C \leq 1$), Λ is the eigenspectrum of the discrete spatial operator and $\alpha(\theta_j)$ denotes the distance from the origin of the boundary of the stability region of the time-integration scheme

along the azimuthal of the jth eigenvalue. The bound imposed by $\Delta t_{\max}$ for $C = 1$ ensures the stability region is large enough to enclose all the eigenvalues of spatial operator. Selecting $C < 1$ we introduce a safety margin, reducing Δt even more and therefore making the stability region even larger.

In common practice the required Δt is estimated numerically via an algorithm which approximates the actual imposition of the CFL condition. The reason for that is because a precise evaluation of the CFL condition is computationally too expensive. In fact the eigenvalues calculation of the spatial operator may become prohibitive, especially for non-linear hyperbolic problems typical of CFD applications, where the operator varies at each time-step. There are many algorithmic variants for the CFL condition estimator which depend on the spatial discretisation technique adopted and on the desired level of accuracy on the estimation of the required time-step. They are generally based on the local (elemental) evaluation of the CFL condition using an approximate value for $|\boldsymbol{V}|$ and an approximate value for Δx. Other approaches exist and they are based on the empirical evaluation of Λ to directly apply Eq. (3.10). Attempts have been made to understand the behaviour of the eigenspectrum for spectral/*hp* element methods with respect to changes in the discretisation. $Sherwin$ (Sherwin 2000) investigated (semi-analytically) the behaviour of the 1D hyperbolic equation, discretised with continuous and discontinuous Galerkin methods, showing that discontinuous projections have significant damping effects at high frequencies. $Karniadakis$ and $Sherwin$ (Karniadakis & Sherwin 2005) indicate a growth rate of the maximum eigenvalue proportional to P^2 for two-dimensional meshes, both for CG and DG projections. $Warburton$ (Warburton 1999, Warburton et al. 1999) performed a study to understand the trend of the eigenvalues for 2D hyperbolic problems and DG projections which showed similar results.

In any case the output of this type of algorithm is generally a value for the admissible Δt, assuming $C = 1$. Selecting $C = 1$ is theoretically the most efficient choice, in fact it translates into selecting the biggest admissible time-step Δt_{max} after the spatial discretisation has been fixed. Nevertheless, because of the non-precise nature of these algorithms, the CFL condition is never exactly fulfilled and the resulting time-step is just an approximation of Δt_{max}. Depending on the algorithm approximation technique and on the specific problem, the resulting Δt_{max} could be not small enough to enforce nu-

merical stability. Therefore, is common practice to select a value of C which is smaller than 1, to introduce a safety margin (the value can vary from case to case). In fact, reducing C translates into proportionally reducing the Δt_{max} deriving from the approximate evaluation of the CFL condition. This approach is often adopted by CFD practitioners and the maximum value of C for which numerical stability is achieved is often called the *maximum CFL number* for the specific simulation.

In this study we do not use any approximate algorithm but we evaluate precisely the value of Δt_{max} to enforce numerical stability for each one of the accounted discretisations, therefore always selecting $C = 1$. Since we are not approximating the imposition of the CFL condition, we do not need any safety margin, $i.e.$ the maximum CFL number is always 1 for our simulations.

The first step is to express the semi-discrete system in Eq. (3.5) in terms of the coefficients as

$$\frac{d}{dt}\mathbf{u} = \mathbf{A}\mathbf{u}, \tag{3.12}$$

where $\mathbf{A}$ represents the discretisation of the linear advection operator and $\mathbf{u}$ is the vector of expansion coefficients. The temporal derivative is discretised using three explicit time integration schemes. The first method is the multi-step second-order Adams-Bashforth (AB2) scheme. We also consider both the second- and fourth-order explicit Runge-Kutta schemes (RK2 and RK4) described by the following Butcher tables:

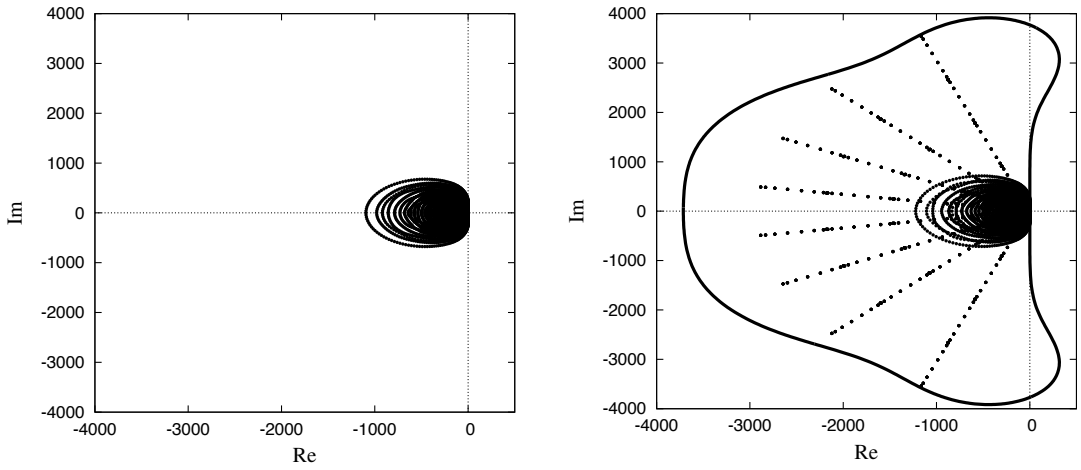
$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \qquad \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \tag{3.13}$$

To maintain numerical stability, the eigenvalue spectrum of $\mathbf{A}$ must lie within the stability region of the chosen time-integration scheme.

For each of the test cases considered in this study, the full eigenspectrum of the weak advection operator $\mathbf{A}$ has been calculated using LAPACK (Anderson et al. 1999). Fig. 3.3 shows examples of the eigenvalue spectrums for uniform and non-uniform meshes

at $P = 7$. While the eigenvalue distribution may show a predictable trend, as discussed above, we use the values calculated by LAPACK for implementing the CFL condition, computing for each numerical simulation the restriction on Δt as reported in Eq. (3.11).

In Fig. 3.3(b) we also show the stability region of the fourth-order Runge-Kutta scheme, scaled by $\Delta t_{\max}$ to minimally enclose the eigenvalue distribution of the spatial operator constructed on the non-uniform mesh in Fig. 3.1(b). As is apparent in the figure, the dominating eigenvalue λ_{dom} which is in closest proximity to the boundary of the rescaled stability region of the scheme may not necessarily be those having maximum modulus or real part, due to the shape of the stability region itself.



(a) Uniform Mesh (b) Non-uniform Mesh

Figure 3.3: Eigenvalues distributions with $P = 7$ for (a) the uniform mesh shown in Fig. 3.1(a) and (b) the non-uniform mesh shown in in Fig. 3.1(b). For the non-unifrom case the stability region for the fourth-order Runge-Kutta scheme is shown, scaled to encompass the eigenvalues distribution.

3.6 Error Model

In order to better understand which aspects of the spatial and temporal discretisation lead to errors in the solution, along with their relative contribution, we introduce the following model to describe the total error ε:

$$\varepsilon = \boldsymbol{f}(\overbrace{C_1(h,P)}^{\text{spatial}}, \overbrace{C_2(h,P)K(q,\Delta t,T)}^{\text{dispersion/diffusion}}, \overbrace{C_3(q,\Delta t,T)}^{\text{temporal truncation}}). \tag{3.14}$$

Eq. (3.14) is composed of three terms, denoting different sources of error, and the simulations outlined in the remainder of this section aim to assess the relative contributions of each of these throughout the parameter space. The first term, $\varepsilon_p = C_1(h, P)$, represents the projection error, that is the contribution due to the projection of the initial condition onto the discrete space. This term is time independent and occurs once at the beginning of the time integration; it is therefore only a function of the discretisation. The third term $\varepsilon_t = C_3(q, \Delta t, T)$ is the truncation error introduced when discretising the temporal derivative. This error is not directly dependent on the chosen spatial discretisation, but depends on the order of the time-integration scheme used (indicated by q), the time-step Δt, and the final time, T. The remaining term accounts for the dispersion/diffusion error of the method and numerical errors associated with multiple applications of the spatial operator. This term couples the spatial and the temporal discretisation, where $K(q, \Delta t, T)$ is the number of applications of the spatial operator, which may vary from scheme to scheme, as well as due to the size and number of time steps taken.

3.7 Results

We present results obtained through numerical experiments. The simulations have been run in serial on a 64-bit Mac Pro using a 2.26GHz Quad-Core Intel Xeon E5520 processor (8MB of L3 cache) and 16GB of RAM. The operating system was OSX with a 10.8 Darwin kernel. All tests were performed using the *Nektar++* spectral/*hp* element framework version 3.1.0 (Kirby & Sherwin 2006*b*). The Accelerate Framework provided with OSX was used for BLAS operations.

3.7.1 Projection Error ε_p

The first source of error in all tests is the projection error introduced when the infinite-dimensional initial function is projected onto the finite-dimensional discrete space through a discontinuous Galerkin approximation. This error is computed as $\varepsilon_p = ||\mathbf{u} - \mathbf{u}^\delta||_{L_2}$, where $\mathbf{u}$ and $\mathbf{u}^\delta$ denote the analytic function and discrete representation, respectively. This is depicted in Fig. 3.4 which shows the error for both uniform and non-uniform

meshes.

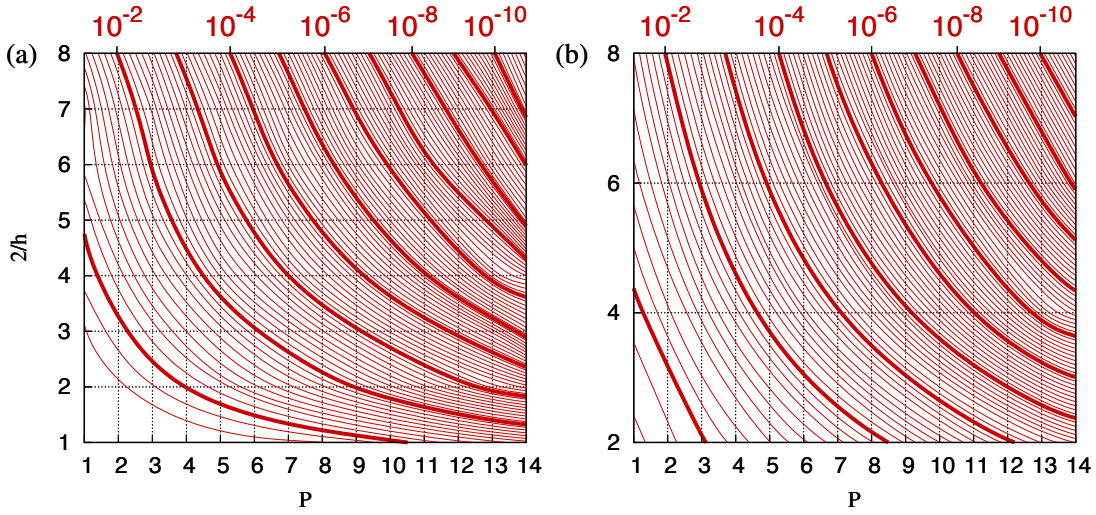
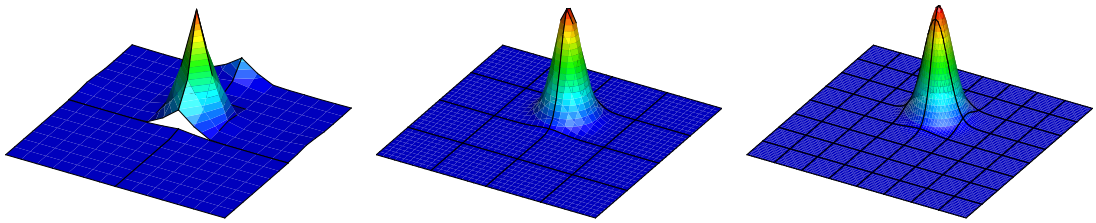


Figure 3.4: L^2 projection error, ε_p, of the initial Gaussian function onto spectral/*hp* element discretisations using (a) uniform meshes, and (b) non-uniform meshes. Gridline intersections indicate possible (h, P) discretisations.

The format of these plots shows increasing number of elements $2/h$ on the y-axis, with increasing polynomial order P of the expansion used on each element along the x-axis. Although the data are discrete, we plot them in a continuous form for the benefit of analysis. Here h corresponds to the size of each element in both coordinate directions. The isolines denote constant ε_p where bold lines denote orders of magnitude. This notation will be used throughout the remaining figures in this paper to represent constituents of the solution error. For highly refined discretisations, projection errors may be as low as $\varepsilon_p = 10^{-10}$. Below an error of 10^{-3} it can be seen that doubling the polynomial order decreases the error by a significantly greater magnitude than doubling the number of elements. This highlights the improved convergence properties of high-order discretisations.

For non-uniform meshes, $y - axis$ values are set to correspond to the uniform mesh they approximate. For example, a non-uniform mesh with 81 elements corresponds to a uniform mesh of 64 elements with the additional 17 elements arising from the narrow strips of elements in the centre of the mesh, and would be represented by $2/h = 8$ on the non-uniform plots, as can be seen in Fig. 3.1. The coarsest non-uniform mesh is that consisting of 9 elements, corresponding to the 4-element uniform mesh. There are few differences in the magnitude of the projection error on non-uniform meshes in comparison

to the uniform equivalents. The only notable difference is for few elements and low polynomial order where the narrow elements provide an increase in projection accuracy.



(a) $(2/h, P) = (2, 4)$ (b) $(2/h, P) = (4, 7)$ (c) $(2/h, P) = (8, 11)$

Figure 3.5: Qualitative representation of the 2D advection problem initial projection. The three examples are showing the gaussian approximation at different levels of accuracy for uniform meshes, where (a) $\varepsilon_p = 10^{-1}$ (b) $\varepsilon_p = 10^{-3}$ (c) $\varepsilon_p = 10^{-8}$.

3.7.2 Effects of Time-integration on the Total Error ε

We now investigate how the choice of time integration scheme affects the L_2-error. Additionally, for each of the three schemes, we will consider two durations of integration in order to help assess when the error introduced by a given scheme becomes important. Short time integration is understood to be integration to a final time of $T = 0.25$, corresponding to the Gaussian being advected for a quarter of a rotation around the domain and equivalent to a distance of approximately two widths of the bump. Long time integration equates to integration to a final time of $T = 4.00$, corresponding to four cycles around the domain and therefore approximately thirty-two wavelengths.

3.7.2.1 Uniform Meshes

Fig. 3.7 summarises these tests for uniform meshes using the local elemental matrix approach for operator evaluations. The left column of plots in this figure correspond to short time integration while the right column shows results for long time integration. The contours of error now correspond to the total error ε accumulated throughout the simulation. In addition, we overlay contours of CPU time. We measure only the time-integration

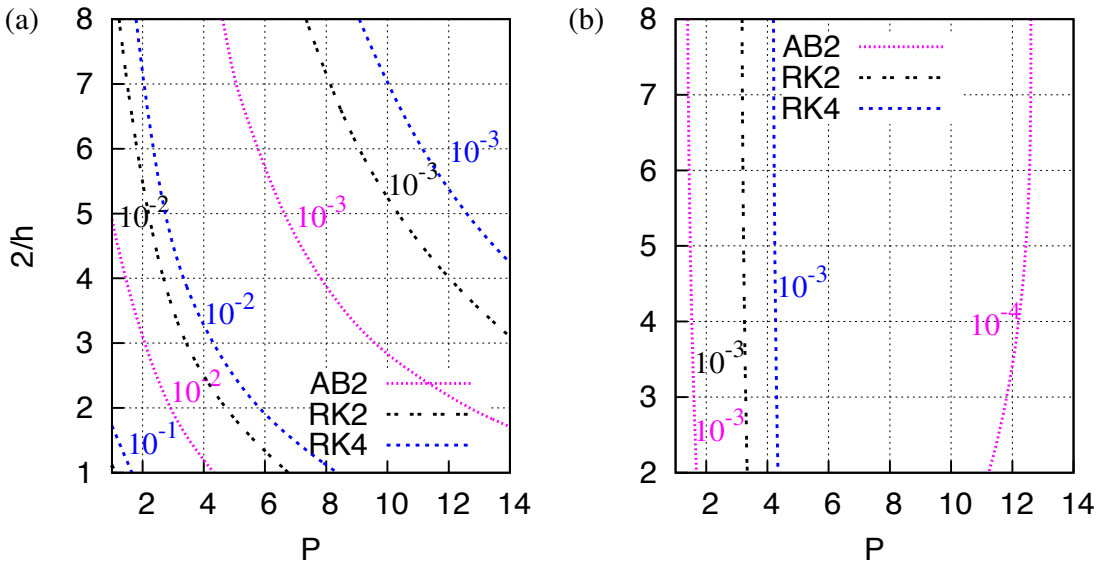


Figure 3.6: Maximum time-step (Δt_{max}) as dictated by the CFL constraint ($C = 1$) for (a) uniform and (b) non-uniform meshes using second-order Adams-Bashforth (AB2), second- and fourth-order Runge-Kutta (RK2 and RK4) schemes.

portion of the total execution, discounting setup costs and I/O. Given a prescribed error tolerance, one now seeks to find a discretisation which achieves this tolerance in the minimal CPU time. This corresponds precisely to the (h, P) combination of minimal runtime which lies on, or to the right of, the chosen error contour. Such minima are denoted by black connected circles, highlighting the optimal path to follow to reduce error at minimal computational cost.

The first observation is that while solution accuracy is comparable across all time integration schemes on coarse meshes, the fourth-order Runge-Kutta scheme achieves far greater accuracy on finer meshes than the second-order schemes. Integrating over long time periods leads to a greater relative increase in error for refined meshes than for coarse meshes across all time integration schemes. These two regimes correspond to where temporal and spatial errors dominate; this will be explored more precisely in section 3.7.4.

CPU time clearly increases with longer time integration. The time-step used in each test is chosen at the limit of the CFL condition, $C = 1$, and is reported in Fig. 3.6. The choice of C derives from the assumption that we do not have a priori knowledge of the initial condition and therefore all eigenvectors could potentially be energised. While

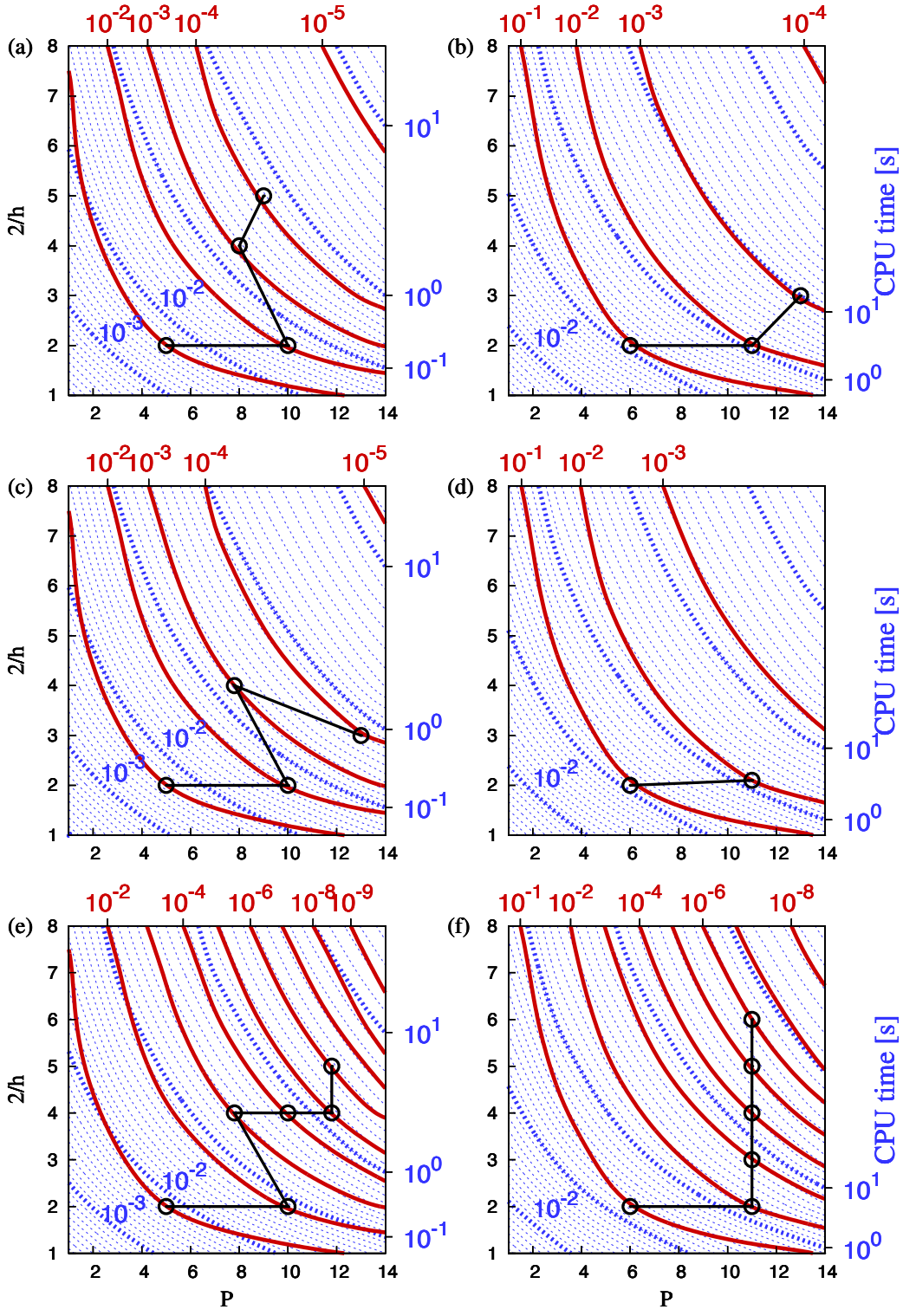


Figure 3.7: Isolines of L_2 error (solid red) and CPU time (dotted blue) for second-order Adams-Bashforth (a,b), second-order Runge-Kutta (c,d) and fourth-order Runge-Kutta (e,f), at times $T = 0.25$ (a,c,e) and $T = 4.00$ (b,d,f). All plots are for uniform meshes using the local matrix operator implementation. Black circles denote the optimal (h, P)-discretisation for the the contours of error where the minimum lies within the explored parameter space.

Fig. 3.6(a) shows that $\Delta t_{\max}$ clearly depends on both h and P for uniform meshes, Fig. 3.6(b) highlights that for non-uniform meshes the maximum timestep is almost independent of h for the parameter space considered. It is apparent that for uniform meshes Runge-Kutta schemes support a larger time-step than Adams-Bashforth. For example, for $P = 8$ and $2/h = 4$, the second-order Adams-Bashforth scheme requires a timestep $\approx 10^{-3}$ while the fourth-order Runge-Kutta scheme requires only $\approx 10^{-2.5}$. However, the fourth-order Runge-Kutta scheme supports only a slightly larger timestep than its second-order counterpart, particularly on coarse meshes

From the contours in Fig. 3.7 we note that for highly accurate solutions the only feasible strategy is to use a high-order discretisation and a high-order time integration scheme together to reduce projection and temporal truncation errors. Even if larger time-steps can be used with the fourth-order Runge-Kutta scheme, it remains slightly more computationally expensive overall than the second-order version since each step requires more work per time-step. Therefore, if we have a high tolerance of errors (for example, 10^{-1}) a second-order time integration scheme using a lower-order discretisation obtains the result in less time than a higher-order scheme, even for the long time period investigated.

We now highlight those (h, P)-combinations which achieve the lowest runtime for each order of magnitude in solution error. These optimal discretisations do not show a clear pattern, but in general to achieve a more accurate solution over long times with second-order time integration schemes the trend suggests that increasing polynomial order offers the most effective strategy. This makes sense, since dispersion errors from repeated application of the operators will decrease exponentially with increasing P. For short times, the total error has a lower temporal component so a more balanced increase in mesh refinement and polynomial order gives the best performance by reducing projection error (i.e. moving normal to the contours of ε_p). The fourth-order scheme suggests that for long time periods increasing mesh element density (h-refinement) is the best approach, but such a conclusion may be considered misleading since the CPU time and error contours are essentially parallel in this region of the parameter space.

3.7.2.2 Non-uniform Meshes

Introducing non-uniformity into the mesh has the most apparent effect on coarse meshes where the small elements impose a much stronger restriction on the CFL limit, and therefore the time step, than would otherwise be the case. This is shown in Fig. 3.8, where CPU time is significantly higher for coarse discretisations than in the equivalent plots for uniform meshes in Fig. 3.7. The increase is less pronounced on finer meshes since the disparity of element sizes is reduced. As a consequence of this change, the choice of optimal discretisation on non-uniform meshes is typically in the fine-mesh, low-order range. In contrast to the uniform case, to improve accuracy in the solution the best strategy for non-uniform meshes is to increase mesh refinement. For smaller error tolerances Fig. 3.8 suggests increasing P is the optimal strategy, however this is purely an artificial consequence of the finite bounds imposed on the parameter-space of this study. It should be noted that, even at $\varepsilon = 10^{-2}$, the discretisation giving minimum CPU time uses $P \geq 4$ which is significantly higher than most conventional finite element methods.

3.7.3 Operator Implementation

So far we have only assessed performance using the local elemental matrix approach for performing matrix-vector multiplications. In this case applications of the explicit matrix operators are performed using a block-diagonal matrix, where each block corresponds to the operator on a single element of the domain. In this section we present performance using the sum-factorisation technique (Orszag 1980). The local elemental matrix approach was shown to be efficient in the continuous Galerkin case for intermediate polynomial orders ($P \approx 4$ to $P \approx 7$) while at higher polynomial orders sum-factorisation is found to be more efficient (Vos et al. 2010, Cantwell et al. 2011*b*,*a*).

In both approaches the operations are performed elementally and afterward the elemental contributions are assembled throughout the assembly procedure described in section 2.1.3.2. In order to highlight the differences between the two techniques we consider the elemental operator $\boldsymbol{B}$, which reconstructs the physical representation of the variable $u(x, y)$ from the expansion basis coefficients $\hat{u}$, as

$$u(x, y) = \boldsymbol{B}\hat{u}. \tag{3.15}$$

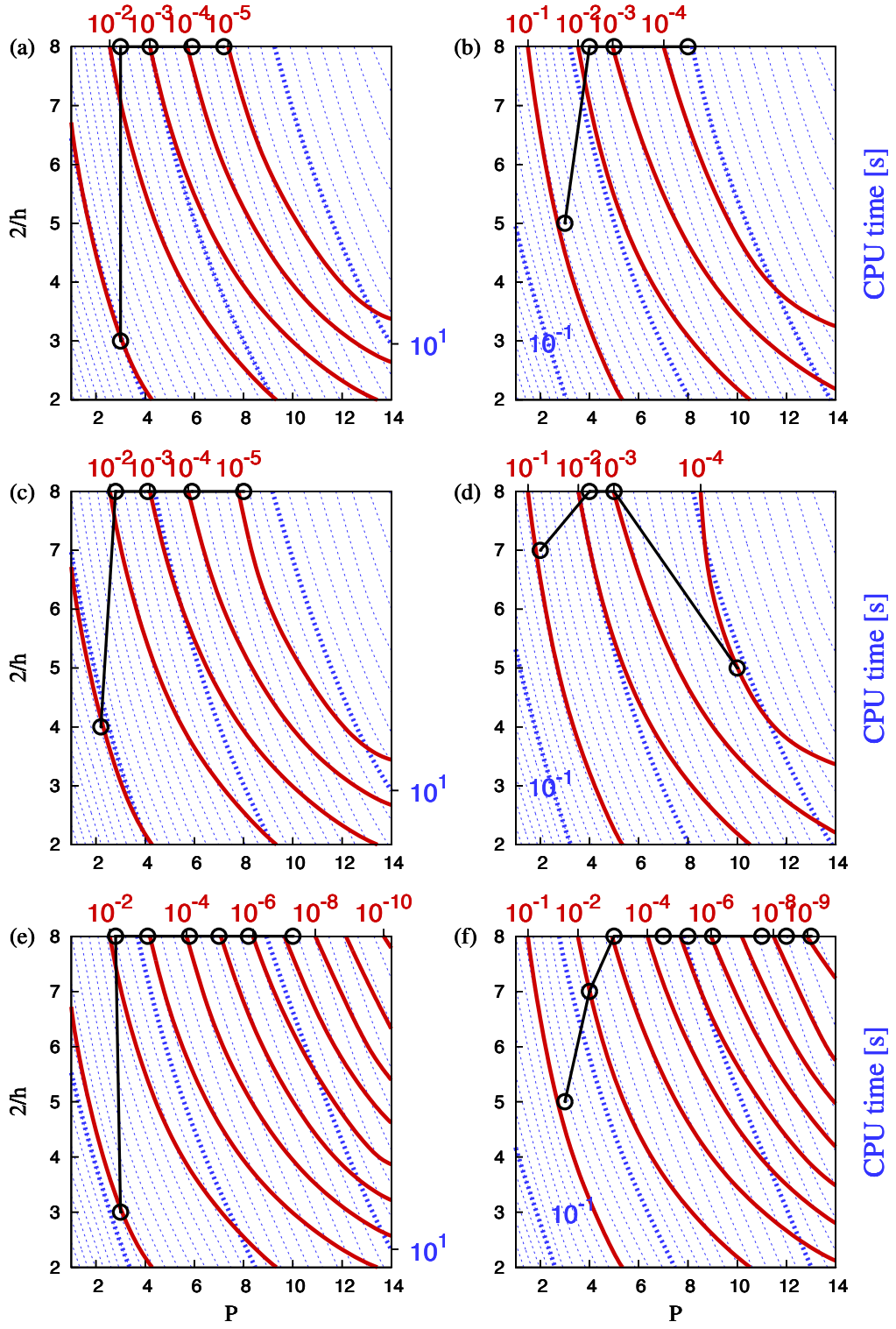


Figure 3.8: Isolines of L_2 error (solid red) and CPU time (dotted blue) for second-order Adams-Bashforth (a,b), second-order Runge-Kutta (c,d) and fourth-order Runge-Kutta (e,f), at times $T = 0.25$ (a,c,e) and $T = 4.00$ (b,d,f). All plots are for non-uniform meshes using the local matrix operator implementation. Black circles denote the optimal (h, P)-discretisation for the contours of error where the minimum lies within the explored parameter space.

In case we use a local elemental matrix approach, $\boldsymbol{B}$ is a precomputed matrix which we can directly apply to the vector $\hat{u}$. Disregarding the transformation from real to reference space for simplicity, this matrix can be conceptually precomputed as

$$\boldsymbol{B} = \begin{bmatrix} \phi_0(x_0, y_0) & \phi_1(x_0, y_0) & \cdots & \phi_n(x_0, y_0) \\ \phi_0(x_0, y_1) & \phi_1(x_0, y_1) & \cdots & \phi_n(x_0, y_1) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(x_Q, y_Q) & \phi_1(x_Q, y_Q) & \cdots & \phi_n(x_Q, y_Q) \end{bmatrix}, \tag{3.16}$$

where $\phi_n(x, y) = \phi_p(x)\phi_q(y)$ is the two-dimensional tensorial expansion basis described in section 2.1.3.4 and (x_Q, y_Q) is the two-dimensional quadrature points grid.

The result obtained by applying $\boldsymbol{B}$ to $\hat{u}$ can be equally achieved via the sum-factorisation technique. Following this approach we do not perform a matrix-vector multiplication, it is in fact a matrix-free technique. The sum-factorisation technique takes advantage of the tensorial nature of both the expansion basis and the quadrature point grid and the operator application translates in

$$u(x_i, y_j) = \sum_{p=0}^{P} \phi_p(x_i) \Big\{ \sum_{q=0}^{P} \phi_q(y_j) \hat{u}_{pq} \Big\} \tag{3.17}$$

where the right-side summation is performed first.

In Figure 3.9 we present timings for uniform meshes and the sum-factorisation technique. These confirm the findings in the literature are also valid for the discontinuous Galerkin case. Furthermore, the optimal discretisations for all error tolerances now lie in the coarse-mesh, high-order regime, since this is the parameter range in which the technique is most efficient. Discussion of this aspect is covered in the literature so we do not consider it further here.

3.7.4 Spatial/temporal dominance

To further understand the relative contributions of the remaining terms in Eq. (3.14), we measure the error (κ) in the solution when using a $Courant$ number of $C = 0.1$. This has the effect of reducing the time-step by an order of magnitude and consequently we can consider the truncation error, $\kappa_t = C_3(q, \Delta t, T)$ to be small or negligible. The remaining error arises from the projection error, $\kappa_p \equiv \varepsilon_p$, and the dispersion error, $\kappa_d \approx$

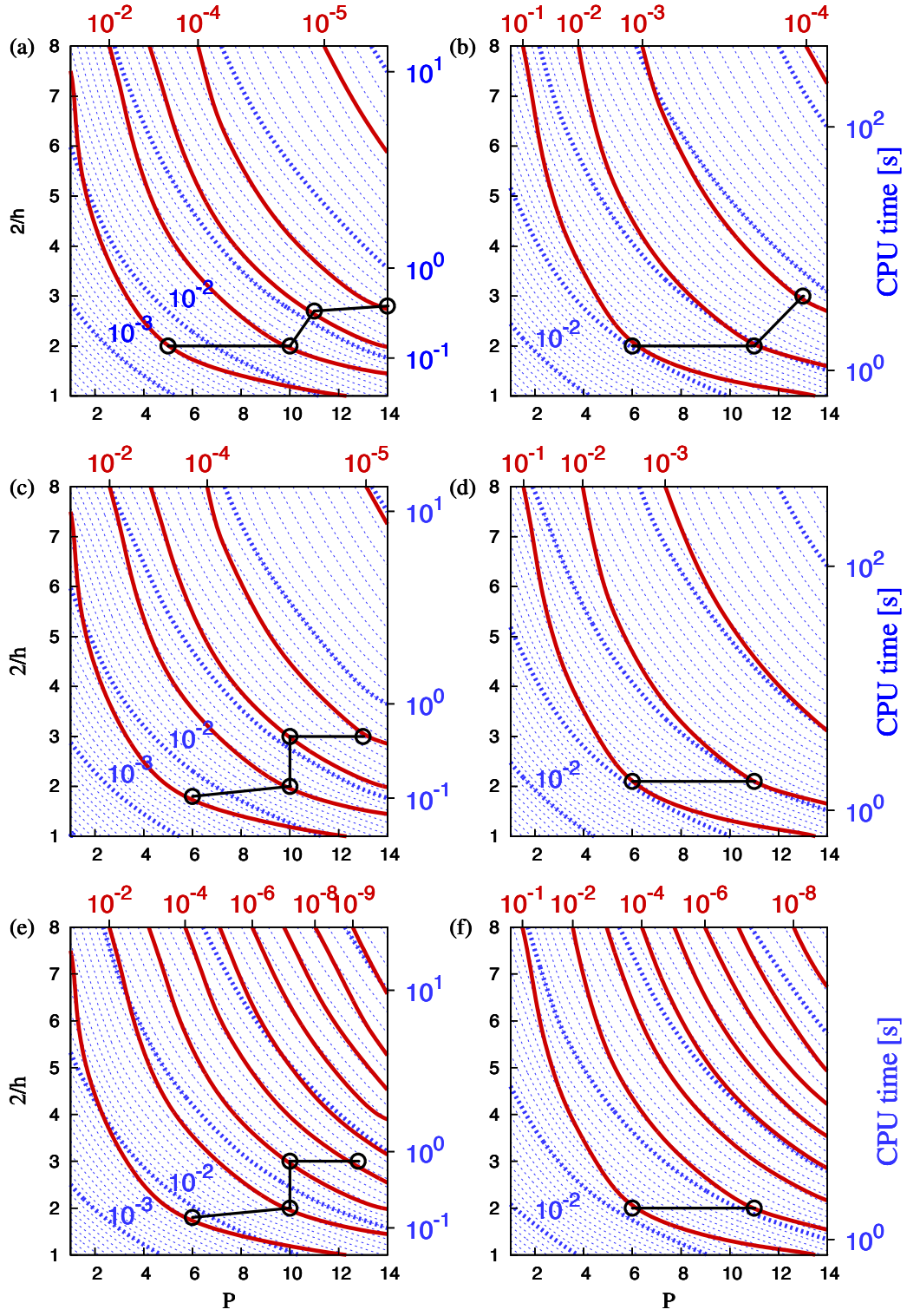


Figure 3.9: Isolines of L_2 error (solid red) and CPU time (dotted blue) for second-order Adams-Bashforth (a,b), second-order Runge-Kutta (c,d) and fourth-order Runge-Kutta (e,f), at times $T = 0.25$ (a,c,e) and $T = 4.00$ (b,d,f). All plots are for uniform meshes using the sum-factorisation technique. Black circles denote the optimal (h, P)-discretisation for the contours of error where the minimum lies within the explored parameter space.

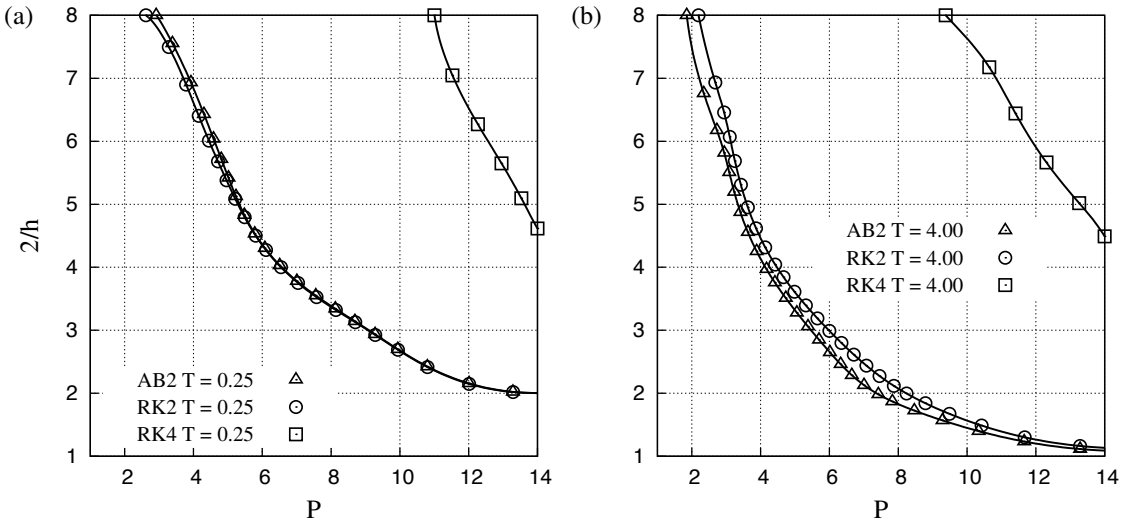


Figure 3.10: Influence zones for uniform meshes and the three time-integration schemes considered for (a) short time integration, and (b) long time integration. Lines indicate $\kappa/\varepsilon = 1$, where κ corresponds to the error when using $C = 0.1$. Discretisations where the spatial error dominates are to the lower-left of the line while to the upper-right temporal error dominates.

ε_d, introduced through the repeated application of the spatial operators. This enables us to identify for which discretisations the ratio of $\kappa/\varepsilon \approx 1$, where we recall that ε is the error with $C = 1$. For $\kappa/\varepsilon > 1$ we have that the spatial error is dominating and where $\kappa/\varepsilon < 1$ temporal errors dominate. Figure 3.10 summarises this data for the three time integration schemes. The lines indicate the boundary between the spatial and temporal error dominance. The region to the left of a given line indicates discretisations for which the dominant error is due to spatial inaccuracy, while the region to the right corresponds to temporal error dominating.

As expected, the error from using fourth-order Runge-Kutta is predominantly spatially dominant unless using refined high-order discretisations. This is consistent with the earlier analysis, indicating the one should increase P for optimal execution time given a desired accuracy. For both second-order schemes the break-even point occurs with much coarser discretisations. Over longer time integration the region of temporal dominance extends further towards coarser meshes and lower polynomial orders. This is a consequence of the additional dispersion error introduced by the order of magnitude increase in the number of time-steps taken to reach the same final time. Although the spatial/temporal dominance is qualitatively predictable, it is interesting to remark how those regions are

actually shaped in the (h, P)-plane and where their boundaries are located for the specific case.

3.7.5 Performance prediction

The ability to predict the time required for a simulation depends on the accuracy when forecasting the eigenvalues distribution of the weak advection operator, given that a direct calculation is often prohibitive in real applications. In order to enhance the understanding of the CFL restrictions which govern our simulations, we investigate the spectrum of $\mathbf{A}$ for regular meshes. Our intention is to recognise a trend in the growth rate of the eigenvalues with respect to (h, P) and then predict $\Delta t_{\max}$ using Eq.(3.11).

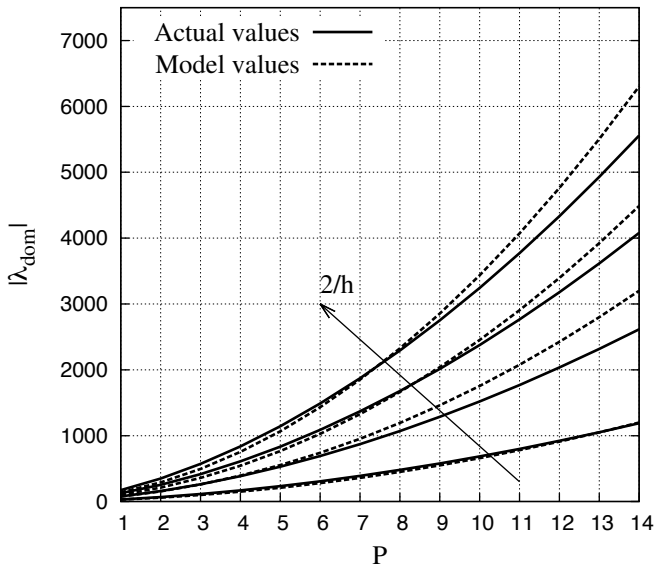


Figure 3.11: Dominant eigenvalue magnitude for uniform meshes. Actual values obtained using LAPACK (solid lines) are compared with the estimate (dashed lines) of Eq.(3.18).

We asses this by monitoring, during our numerical experiments, the actual magnitude $|\lambda_{dom}|$ of the eigenvalue which dominates the stability of the scheme. For regular meshes the eigenvalue which quantifies the CFL restriction appears to be the one showing the minimum real part, $i.e.$ $\theta_j = \pi$. We model $|\lambda_{dom}|$ growth rate as

$$|\lambda_{dom}| \approx B\Big(h^{-1/2}P^2 + h^{-1/4}P\Big) \approx \tilde{B}P^2. \tag{3.18}$$

Throughout a calibration process we extract $B = 9.6265$. Figure 3.11 shows a comparison

between the actual values of $|\lambda_{dom}|$ and the model predictions. Although Eq.(3.18) is a rough estimate of $|\lambda_{dom}|$, the discrepancies between the forecasted and actual values are always less than 20%. The maximum error appears for high values of P, where the model overestimates the eigenvalue magnitude.

The model reported in Eq.(3.18), although problem specific, is consistent with what is anticipated in (Karniadakis & Sherwin 2005), where $|\lambda_{dom}|$ growth rate was identified as being proportional to P^2 for a weak advection operator. Provided that $\Delta t_{\max}$ can be estimated using Eq.(3.18) and Eq.(3.11), we can know beforehand the CPU-time required for a specific (h, P, T) combination.

3.8 Discussion

In this chapter we have systematically assessed the relative importance of discretisation and time-integration scheme when targeting minimal runtime. The spatial discretisation and time integration schemes both impose restrictions on the overall accuracy of the solution, but their relative error contributions will vary depending on the exact choice of discretisation parameters chosen. All the results demonstrate there are substantial benefits for using high-order methods for transient problems, while also highlighting some of the subtleties in choosing optimal discretisations to minimise runtime.

For each time integration scheme and specific choices for the final time T, we have identified the region in the (h, P)-plane for which the error in the solution is primarily due to the underlying inaccuracy of the spatial discretisation rather than a consequence of time integration. Outside this region, typically for more refined discretisations, time integration errors are the dominant cause of solution error. These divisions naturally differ for the three time integration schemes with the spatially dominant zone extending to finer discretisations for high-order time integration schemes, compared with the lower-order counterparts. A consequence of this is that higher-order time integration schemes offer no advantage over their computationally less-expensive lower-order counterparts if the solution error at the chosen discretisation is spatially dominated under both schemes.

The choice of the time-integration scheme therefore requires careful consideration. In particular, we have noted that for short-time integration and for error tolerances down

to 10^{-3}, high-order time integration schemes, such as the fourth-order Runge-Kutta, are not competitive for our 2D advection test problem. Second-order Adams-Bashforth and second-order Runge-Kutta achieve the same solution accuracy in lower runtimes in these cases. However, achieving highly accurate solutions typically requires a high-order discretisation, and therefore a high-order time-integration scheme, in order to keep both the spatial and temporal error contributions sufficiently small. Furthermore, over long time-integration periods, the shift in the break-even point between spatially and temporally dominated zones dictates that high-order time integration schemes are more important for maintaining overall solution accuracy.

High-order methods offer exponential reduction in error with increasing polynomial order. Increasing P should therefore offer a more attractive approach to increasing the solution accuracy than refining the mesh. This is evident in some of the uniform mesh results, particularly for long time-integration periods. The second-order schemes show significant variation in CPU time along a given error contour and the path of minima is predominantly in the direction of increasing P.

In changing the elemental polynomial order, the choice of implementation strategy for matrix-vector operations requires consideration. For continuous Galerkin the literature highlights the use of a whole-domain global matrix approach for low polynomial orders, a local elemental block-matrix approach for intermediate orders and the local elemental sum-factorisation approach for higher orders. The exact break-even points between these different strategies is of course dependent on the element type and performance of the computational hardware, but general observations can be made. We confirm a similar trend is true for discontinuous Galerkin projections for the local elemental and sum-factorisation strategies.

We conclude with a discussion of the effect of element-size diversity on the time step and consequently the selection of spatial and temporal discretisation for optimal performance. Variation in size and advection velocity across mesh elements dictates the spread of the eigenspectrum of the spatial operator, with smaller size-to-velocity ratios leading to greater magnitude eigenvalues. This leads to a more restrictive time step in order to enclose the entire eigenspectrum inside the stability region of the time-integration scheme. In general, accuracy on uniform meshes can be best achieved using a high-order discreti-

sation, and best improved through further increasing the polynomial order. While high-order discretisations are still effective for the non-uniform meshes considered, the most efficient way to increase accuracy is to reduce the size of the larger elements, thereby essentially converging towards a uniform mesh. This aligns with the common wisdom that h-refinement is most appropriate on meshes where there is a significant disparity in element size. Furthermore it also raises the possibility of variable polynomial order usage which was not explored in this study.

Finally, there is a common understanding that high-order methods generally lead to stringent CFL limitations, due to the eigenvalues of the spatial operators growing as a polynomial power of P. However, we have shown that even for a coarse error tolerance on the solution, high-order methods often become the most efficient choice. This is due to the accuracy of the solution increasing faster than the stability requirements limit the time step. Consequently high-order methods offer substantial performance over their linear-order counterparts for transient simulations.

As we stated from the outset, the absolute numerical values presented in this chapter are code dependent and will also vary along with the nature of the problem, size of the problem and the machine used. However, the numerical experiments highlight some general trends and the results support the common wisdom that high-order methods are particularly important for long and accurate time integration.

Chapter 4

Parallelisation Strategies

The fast development of super-computers forces software designers to make a continuous effort to keep algorithms up-to-date and able to exploit all the benefits coming from hardware innovation. In the last decade investigations of numerical schemes, parallelisation paradigms and algorithm efficiency have been fundamental to push the limits forward. Parallelisation strategies usually depend on the applications (nature and size of the problem), the hardware and the numerical methods involved in the solution of the problem. Predicting the scalability and efficiency of an algorithm on a specific architecture is not trivial, as demonstrated by the seminal work of $Gruma\ et\ al.$ in 1993 (Grama et al. 1993). In this chapter we discuss how a sensible and flexible implementation can be used to achieve higher levels of parallelism and efficiency when solving the three-dimensional Navier-Stokes equations. Although the following results, techniques and considerations might be applied to other engineering problems, we focus on the solution of incompressible flows using the velocity-correction scheme described in section 2.3.

In section 4.1 we give a brief overview of the role played by parallel computing in

CFD applications. Subsequently, in section 4.1.1 we recall some relevant work on the topic, in particular those which are at the base of our studies. In section 4.1.2 we state the objectives and the investigation philosophy of our study. After these introductory sections we present, in section 4.2, the building blocks of the incompressible Navier-Stokes algorithm. In this part we describe, from an algorithmic point of view, the main steps required to advance in time the velocity correction scheme reported in section 2.3. In particular, we restrict our attention to the time-stepping cycle. In section 4.3 we introduce the actual parallelisation of the algorithm. Initially, we describe in detail the two basic parallelisation approaches mentioned in section 4.1.1, highlighting their practical issues. Afterward we show how to combine these techniques to attain a hybrid parallelisation approach. Section 4.4 describes briefly the test cases which will be used to perform numerical experiments and the machine specifications. In section 4.5 we present the steps to construct a scalability model for our hybrid parallel algorithm. Using this model we show how performance can be predicted in section 4.5.6, providing some guidelines on how to interpret the model outcomes. Finally, in section 4.6 we highlight some scalability tests performed on the Imperial College parallel cluster using $Nektar++$. In these tests we solve the 3D Navier-Stokes equations for two different flows, the turbulent pipe and channel reported in section 2.3.2. In section 4.7 we then summarise our findings and observations.

4.1 Application to Fluid Dynamics

A practical CFD application in which parallelisation is of critical importance is the DNS of a turbulent flows. When there is no turbulence model all the scales need to be resolved. Therefore, the spatial discretisation needs to be fine enough to capture also the smallest scale (Komolgorov scale), which is commonly denoted with η. The Kolmogorov dissipative scale can be related to the kinematic viscosity ν and to the rate of kinematic energy dissipation ε as

$$\eta = \left(\frac{\nu^3}{\varepsilon}\right)^{\frac{1}{4}}. \tag{4.1}$$

If we call h our spatial resolution, we need to select $h \leq \eta$ to capture the Kolmogorov scale. Given the integral scale L along a given direction (discretised with N points), to

satisfy the resolution requirements it must also be true that $Nh > L$. Combining the previous constraints with Eq. (4.1) and the approximation $\varepsilon \approx u'^3/L$, we obtain that, for a three-dimensional problem, the number of degrees of freedom N grows with Re as

$$N^3 \geq Re^{9/4} \tag{4.2}$$

where

$$Re = \frac{u'L}{\nu}. \tag{4.3}$$

The greater the number of degrees of freedom the greater the number of operations and the memory requirements. Hence the computational cost of a DNS solution grows exponentially as we increase the Reynolds number. Moreover, explicit time-integration schemes are often employed to time-step the Navier-Stokes equations. As reported in chapter 3, refining the spatial discretisation implies a reduction of the time-step for explicit methods (CFL condition). The obvious consequence is that to integrate until $\tau = L/u'$ we need a number of time-steps proportional to $L/C\eta$, where C is the *Courant* number. Deducing $L/\eta \sim Re^{3/4}$ from previous equations, we can conclude that the DNS of a turbulent flow requires a number of floating-points operations which exponentially grows with Re as

$$\text{FLOP} \approx Re^{9/4} Re^{3/4} \approx Re^3. \tag{4.4}$$

Practical Reynolds numbers are of the order of thousands to millions, making serial computations impossible, as Eq. (4.4) suggests.

4.1.1 Overview of Previous Works

In the last five decades, along with the development of faster and bigger super-computers (Meuer et al. 2013), CFD practitioners progressed their parallelisation approaches to take advantages of new hardware, performing simulations of more complex flows (in term of geometries) and higher Reynolds numbers. As a consequence, there is a vast literature on the topic. In this section we report the works which have been fundamental for our studies. The techniques which follow will constitute the building blocks of our implementation.

Karniadakis et al. focused on the solution of the 3D incompressible Navier-Stokes equations using the Fourier spectral/*hp* element method described in section 2.1.7. Between 1995 and 1996 they presented a series of studies in which they benchmarked and

modelled the scalability of their code $Prism$ (Evangelinos & Karniadakis 1996, Crawford et al. 1996). The parallelisation was implemented in the spectral direction, sending different Fourier modes to different processors, and using MPI $All - to - All$ calls to transpose data through the processes to perform operations (such as derivatives) in the third dimension. Under the assumption of a flat topology they also presented a scalability model able to predict the code behaviour. The maximum number of processors which can be used following this approach is equal to the number of Fourier modes adopted in the spectral discretisation. The upper bound in the number of employable processors is commonly referred as the natural bottleneck of the technique.

Also in the nineties $Fisher\ et\ al.$ approached the solution of turbulent incompressible flows, implementing parallel algorithms which apply an elemental decomposition of the spectral/*hp* element discretisation (Fischer 1990, Fischer & Rønquist 1994, Fischer 1994, Fischer & Patera 1994, Fischer 1997). In their studies 2D and 3D domains are discretised using a full spectral/*hp* element method. The natural bottleneck for this parallel technique is the number of elements from which the mesh is constructed. They implemented optimal decomposition techniques and focused on the speed-up of the linear system parallel solution. When using an iterative solver they centred on reducing the communication pattern which dominates each conjugate method iteration but also on the overall reduction of the iteration count introducing appropriate preconditioners. More recently they moved on the direct solution of liner systems in parallel achieving high levels of granularity (few elements per processor) and strong scaling up to thousands of CPUs (Tufo & Fischer 2001, Fischer et al. 2008).

The common practice is to optimise the algorithm and the communication pattern of a specific approach in order to achieve strong scalability up to the related bottleneck. Afterward the general enhancement is to gain benefits from a new architecture (larger number of CPUs) increasing the number of degrees of freedom, which allows the investigation of higher Reynolds numbers. Although this weak scalability is still very useful for turbulent simulations, it may not always be of practical interest. In fact the approaches just mentioned force users to study larger problems (in terms of degrees of freedom) in order to exploit the growing performances of super-computers.

In 2007 $Hamman\ et\ al.$ presented a study (Hamman et al. 2007) where both the

modal and elemental parallelisation were applied concurrently. They implemented an algorithm where a 1D spectral/*hp* element method combined with a 2D Fourier spectral expansion were used to solve a 3D turbulent channel flow. They utilised the same velocity correction scheme presented in section 2.3 and a hybrid parallel approach which consists in a mixed elemental-modal decomposition using a MPI cartesian communicator. Assuming a fixed number of iterations for the solution of the linear systems involved in the sub-steps of the schemes, they also produced a scalability model. The scalability performances presented in this work clearly suggest that a hybrid-algorithmic approach provides the tools to achieve strong scalability on specific architectures.

4.1.2 Motivations

In this study we investigate the issue from a different perspective: the design of the algorithm and its optimal usage. The aim is to identify possible algorithmic solutions which can extend the strong scalability range by mixing standard implementations, but also design a software environment able to adapt itself to various problem needs and hardware configurations. Combining the flexibility of an elemental tessellation with the accuracy of spectral approximations, *Nektar++* is equipped with various algorithmic components to solve 3D problems. A sensible usage of these components can be employed to tune the code to achieve higher levels of parallelism. Parallelisation approaches and numerical discretisation techniques are encapsulated within C++ classes in *Nektar++* (see Appendix A). This flexibility facilitates alternative selections of one or more combined algorithms, allowing the optimised use of the provided hardware characteristics for the solution of a specific problem. The ability to select the most appropriate numerical method or parallelisation approach plays a relevant role in the parallelisation efficiency, portability and, at the same time, provides a solid base for exploiting possible new hardware features. Typical variables characterising a super-computer such as latency, bandwidth, cache and available memory per node, can be pushed to their limits selecting the most suitable numerical techniques. An immediate consequence of this approach is an optimal usage of the computational resources, reducing computational time and costs.

A common scenario in many engineering applications is a 3D problem which needs

to be solved to a desired accuracy. Within a 3D spectral/*hp* element discretisation we can alter with various parameters to keep an optimal balance of computation, communication and memory usage. The number of degrees of freedom ($i.e.$ the resolution) is proportional to both the number of elements and the spectral expansion order. Proper combination of these two parameters can maximise the efficiency of a mesh decomposition technique on a given machine, preserving the desired accuracy. Applying a 3D hybrid discretisation as presented in section 2.1.7, we can also select which type of parallelisation we want to implement. Following the approach presented by $Kardiadakis$ (Evangelinos & Karniadakis 1996), we could solve each Fourier mode ($i.e.$ each pair of 2D spectral/*hp* element planes) on a different processor taking advantage of the Fourier basis orthogonality for linear operators. Communications are then required just when operations in the spectral direction are needed. On the other hand, we could decompose our operations across processors elementally, as efficiently performed by $Fisher$ (Fischer 1990, Fischer & Rønquist 1994, Fischer 1994, Fischer & Patera 1994, Fischer 1997) in his previously mentioned studies. In this case, communication is required while solving the linear systems. The former solution generally requires the transmission of fewer and bigger messages compared to the latter. Depending on the hardware features and on the ratio of the degrees of freedom between the elemental and spectral discretisation, one parallelisation approach may outperform the other. Furthermore, a sensible combination of them could be the most efficient choice and it can be used to increase the scalability limit.

As a variant/extension of $Hamman\ et\ al.$ (Hamman et al. 2007) study, we investigate the scalability of a similar algorithm where the 3D domain is built throughout a 2D spectral/*hp* element method and a 1D Fourier spectral expansion ($i.e.$ the Fourier spectral/*hp* element method described in section 2.1.7). As test cases for our investigations we re-use the turbulent simulations presented in section 2.3.2. These basic turbulent flows can be solved using harmonic expansions in two of the three dimensions (as accomplished in (Hamman et al. 2007)). However, we decide to remove the periodicity constraint in one of the dimensions. This approach will allow future studies of more complex geometries, such as flows over airfoils and stability analysis.

Our studies highlight how the two basic parallel approaches yield to different results in terms of efficiency and scalability. We also show that the combination of these two paral-

lel algorithms allows strong scalability beyond the natural bottlenecks given by the mesh size and the number of Fourier modes. An iterative approach coupled with a diagonal precoditioner has been used to solve the linear systems arising during the time integration. The selection of an appropriate preconditioner to reduce significantly the number of iterations in a conjugated gradient method is not trivial. In order to facilitate the construction of a scalability model we limit our investigation to the easily pre-computable diagonal preconditioner. We compare in the results section, when it is possible[1], the iterative approach with a direct solution of the linear system where preconditioning is not required.

4.2 Algorithm Overview

When solving the three-dimensional Navier-Stokes equations a sequence of operations (usually confined in dedicated routines) is required to reach the desired final condition. These operations can be interpreted as computational costs and expressed as the time required to perform them. A basic classification of these costs is:

- $input - output$ costs (also know as I/O costs);
- $set - up$ costs;
- $time - integration$ costs.

I/O costs consist of the time required to load all the problem details from the input file (mesh, boundary conditions, parameters, etc.) and the time to write the solution to file once the simulation is finished (or at some intermediate steps during the time-integration). An efficient parallelisation of I/O routines is the subject of current research and may influence the code performance and reliability (No et al. 2002). Although we have an operating parallel implementation of I/O routines we disregard their contribution in this study as they ideally appear just at the very beginning and the very end of a simulation. In the case of long time-integration, fairly common in turbulent scenarios, their cost can be

[1] A direct solution of the linear system is possible when we are using a pure spectral parallelisation, leaving two or more whole planes per processor.

considered negligible compared to the main computation. For the same reason we ignore also the set-up costs, $i.e$ the time required, once the problem is loaded, to build and to store all the required matrices, vectors and variables which are used during the simulation. These are common assumptions when investigating parallel efficiency (Hamman et al. 2007, Evangelinos & Karniadakis 1996). Therefore we are left with the time-integration costs, which we defined as all the costs required to actually time-step the initial flow condition to its final state throughout the velocity correction scheme steps indicated in section 2.3. Fig. 4.1 provides a diagrammatic representation of the algorithm where just

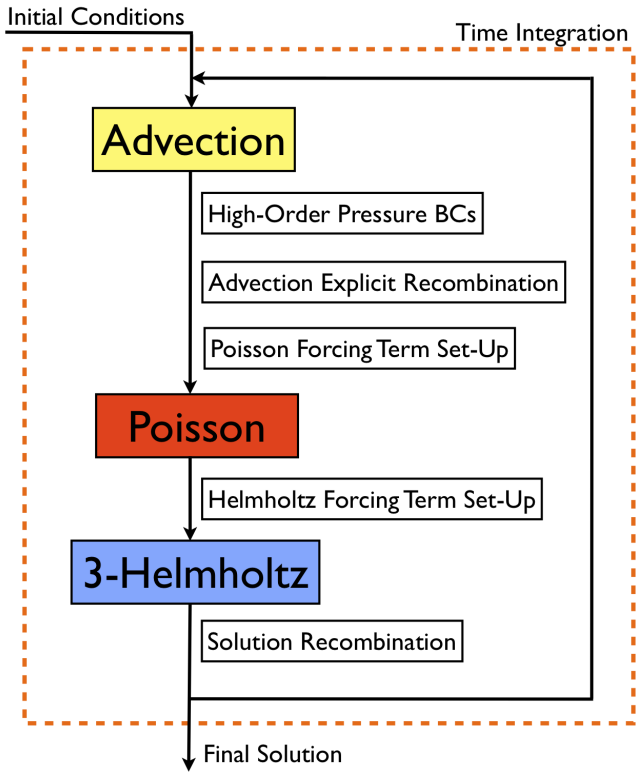


Figure 4.1: Incompressible Navier-Stokes solution algorithm. Details of the building blocks of the time-integration process. The most expensive routines are highlighted, $i.e$. the advection term calculation and the elliptic solvers for pressure and velocity (Poisson and Helmholtz).

the time-integration costs are presented. We can observe that, removing all the I/O and set-up routines, we end up applying repeatably a series of steps in the time-integration loop. In the following, we present a description of this procedural algorithm. Some ideas about parallelisation are introduced to supply a general picture of the problem but their detailed description is left to section 4.3.

The initial condition enter the time-integration loop and, after all the sub-steps are performed, it is advanced in time (by Δt). This is repeated as many times as required to reach the final state (in terms of final time or in terms of the number of steps). As Fig. 4.1 suggests some of the sub-steps in the velocity correction scheme are more computational demanding than others. Indeed, as we shall show in the results section, the total computational time required for a time-step is dominated (80% or more) by the advection term calculation and the elliptic solves.

The parallelisation, and thus the communication, plays a significant role in these three steps. To reduce and decouple the number of communications throughout the cycle we decide to advance our variables (pressure and velocity) in a semi-physical space, which corresponds to $\tilde{u}_k(x,y)$ for the generic variable u in Eq. (4.5). This means that the variables in physical space $u(x,y,z)$ will be Fourier transformed while time-stepping and moved back to physical space just when constructing the nine components of the advection term.

$$\overbrace{u(x,y,z)}^{\text{physical space}} = \sum_k \overbrace{\phi_k(z)}^{e^{i\beta_k z}} \overbrace{\tilde{u}_k(x,y)}^{\text{Fourier space}} = \sum_k \sum_{pq} \phi_{pqk}(x,y,z) \overbrace{\hat{u}_{pqk}}^{\text{coefficient space}}. \tag{4.5}$$

This approach allows us to reduce the number of $DFTs$ required during the computation but also decouples the two parallelisation approaches. The parallelisation along the Fourier expansion requires communication during the advection term calculation only, where the non-linear terms needs to be constructed in physical space in order to take the collocation inner products $N(u_i) = u_j \partial u_i / \partial x_j$ for $i,j = 0,1,2$. We will give an overall description of this parallelisation in section 4.3.1.

After the calculation of the advection term, the high-order pressure boundary conditions need to be evaluated. Fourier-transformed variables facilitate a local evaluation of Eq. (2.107e), provided that we have an even number of modes per processor. In fact the few derivatives in the Fourier space, required for this calculation, can be obtained through a multiplication by the wave number and a coefficient swap between the conjugated cosine/sine modes $\partial \tilde{u}_k(x,y)/\partial z = \boldsymbol{i}\beta_k \phi_k(z) \tilde{u}_k(x,y)$.

Once we have locally recombined the advection term we need to solve four linear systems. The elemental parallelisation requires a series of messages to be passed between

processors only when solving elliptic problems. The various chunks of mesh obtained by the partitioning algorithm need to communicate during the system solution. We will discuss in more detail the solution of a linear system in parallel in section 4.3.2.

All the forcing terms can be set-up locally and no Fourier transformations are required after the advection term. Some more communications and $DFTs$ may be required during the set-up of the problem. For example a first series of $DFTs$ are undertaken for the transformation of initial/boundary conditions and forcing terms. Also the distribution of the mesh partitions among processors will require a sequence of messages. As mentioned before, we disregard all these communications because they appear just at the beginning of the simulation and can be considered *una tantum* costs.

In order to have a consistent notation throughout this chapter we define in Table 4.1 the quantities we will use in the following to quantify the problem size, messages size and number of operations.

Table 4.1: List of quantities used to define operations and communications.

N_{el}^{plane}	number of elements in a 2D plane
N_Z	number of degrees of freedom in the spectral direction z (number of planes)
N_{el}	total number of elements $N_Z \times N_{el}^{plane}$
P	polynomial expansion order
N_{XY}	number of degrees of freedom in the $xy-$plane $N_{el}^{plane} \times (P+2)^2$
N_{TOT}	total number of degrees of freedom in the 3D domain $N_{XY} \times N_Z$
$\boldsymbol{P}_{XY}$	number of processors associated with the mesh partitioning
$\boldsymbol{P}_Z$	number of processors used for the DFT parallelisation
$\boldsymbol{P}_{TOT}$	total number processors $\boldsymbol{P}_Z \times \boldsymbol{P}_{XY}$
N_{el}^{loc}	$N_{el}/\boldsymbol{P}_{TOT}$ (elements on each processor)
N_{XY}^{loc}	$N_{el}^{loc} \times (P+2)^2$

4.3 Parallelisation Approaches

In this section we describe in detail the parallelisation approaches we have inserted in our implementation, $i.e.$ the parallelisation of the spectral component of the discretisation (modal parallelisation) and of the spectral/*hp* element component (elemental parallelisation). Section 4.3.1 highlights how the modal parallelisation has been inserted while in section 4.3.2 we focus on the mesh decomposition process and the elemental parallelisation. Finally in section 4.3.3 we show how to combine the two techniques to create a hybrid approach.

4.3.1 Modal Parallelisation

We now focus on the parallelisation of the spectral method discretisation following the approach presented by $Karniadakis\ et\ al.$ (Evangelinos & Karniadakis 1996, Crawford et al. 1996). As mentioned in section 4.2 the time-integration is carried out in Fourier space. The only step during the cycle in Figure 4.1 for which a physical representation of the variables is required is when the advection term is calculated. Knowing in advance that we have to transform the velocity components back to physical space, we can take advantage of this and keep available both the forms. If we pay attention at each step of the convective term routine in **Algorithm 4**, we can discern what form is the most convenient to use for each operation (the physical or the Fourier space). In the rest of the chapter we assume the advection term is evaluated in its convective form and no dealiasing routines are applied. Hence we call u_i with $i = 0, 1, 2$ the three velocity[2] components in physical space and $\tilde{u}_i$ the same variables in their Fourier transformed form. Following the same logic, we call $N(u_i)$ and $\tilde{N}(u_i)$ the three advection-term components in their physical and Fourier transformed state respectively. The sequence of operations required for the advection term is described in **Algorithm 4**. $IDFT$ and DFT are respectively the inverse and regular discrete Fourier transforms. Differentiation is achieved through a matrix-vector multiplication using the elemental derivative matrices $\boldsymbol{D_x}$ and $\boldsymbol{D_y}$ in the $xy-$plane. $\tilde{\boldsymbol{D}}_{\boldsymbol{z}}$ indicates a pseudo-matrix able to swap the cosine with the sine coefficient and multiply by the wave number. In practice this operation is performed without any

[2]In previous chapters we also used (u, v, w).

matrix application.

input : $\tilde{u}_0, \tilde{u}_1, \tilde{u}_2$

output: $\tilde{N}(u_0), \tilde{N}(u_1), \tilde{N}(u_2)$

`// Transformation back to physical space`

for $i = 0$ **to** 2 **do**

(1) $IDFT(\tilde{u}_i) = u_i$

end

for $i = 0$ **to** 2 **do**

`// Derivatives in the 2D spectral/hp element plane`

(2) $\partial u_i/\partial x = \boldsymbol{D_x} u_i \qquad \partial u_i/\partial y = \boldsymbol{D_y} u_i$

`// Derivatives in the spectral direction`

(3) $\partial \tilde{u}_i/\partial z = \tilde{\boldsymbol{D}}_{\boldsymbol{z}} \tilde{u}_i$

`// Transformation back to physical space`

(4) $IDFT(\partial \tilde{u}_i/\partial z) = \partial u_i/\partial z$

`// Construction of the` $i-th$ `advection component`

(5) $N(u_i) = u_0 \partial u_i/\partial x + u_1 \partial u_i/\partial y + u_2 \partial u_i/\partial z$

`// Transformation to Fourier space`

(6) $\tilde{N}(u_i) = DFT(N(u_i))$

end

Algorithm 4: Non-linear advection term procedure.

Step (1) of **Algorithm 4** consists of performing a series of 1D $IDFTs$ for each one of the velocity components. We perform N_{XY} 1D $IDFTs$ (in serial) for each component, $i.e.$ one $IDFT$ for each quadrature point of the 2D discretisation. At this stage we have both the physical and the Fourier transformed representations of the velocity. During steps (2) and (3) we decide to use the physical space for derivatives in the $xy-$ plane and the Fourier space for derivatives in $z-$direction, reducing in this way the number of $DFTs$ required. The three derivatives in the $z-$direction will be in Fourier space and they require a set of $IDFTs$ in step (4) to be consistently transformed and ready to be

multiplied and added in the fifth step. The last step (6) highlights the final transformation of the advection term components $N(u_i)$ into Fourier space in order to be processed, without any other transformations, in the remaining part of the cycle. From **Algorithm 4** we can deduce that the total number of $DFTs$ has been reduced to nine. Each one of these nine global $DFTs$ actually refer to a series of N_{XY} 1D $DFTs$.

4.3.1.1 FFT Parallelisation

Parallelising the spectral method translates to performing the $DFTs$ in parallel, since the transformation from Fourier to physical space is the only global operation, $i.e.$ requiring all the information in the $z-$direction. The DFT of a set of N data requires $\Theta(N^2)$ operations (N is a power of 2). In practice the 'naive' version of a DFT becomes slow as the number of DOFs increases (N). Therefore we use an FFT algorithm to perform such transformations. Since the algorithm of Cooley and Tukey appeared (1965), many other variants of the FFT algorithm have been created (an example is the $\mathrm{Temperton}$ variant published in 1983). In order to have a deeper appreciation of the operations required during an FFT we give a brief description of it in the following. The original[3] FFT algorithm reduces the number of operations from $\Theta(N^2)$ to $\Theta(N \log N)$ where N is a power of 2. The basic idea can be shown by considering the one-dimensional, unordered, radix-2 FFT $Y = (Y[0], Y[1], \ldots, Y[N-1])$ of a sequence $X = (X[0], X[1], \ldots, X[N-1])$,

$$Y[i] = \sum_{k=0}^{N-1} X[k]\omega^{ki} \qquad with \qquad 0 \leq i \leq N-1 \tag{4.6}$$

where $\omega = e^{2j\pi/N}$ and $j = \sqrt{-1}$. The powers of ω are also known as *twiddle factors*. Assuming that N is a power of 2, the FFT algorithm permits the $N-\mathrm{points}$ of the DFT to be split in two $(N/2)-\mathrm{points}$ $DFTs$ as:

$$Y[i] = \sum_{k=0}^{(N/2)-1} X[2k]\omega^{2ki} + \sum_{k=0}^{(N/2)-1} X[2k+1]\omega^{(2k+1)i} \tag{4.7a}$$

$$Y[i] = \sum_{k=0}^{(N/2)-1} X[2k]\tilde{\omega}^{ki} + \omega^{i} \sum_{k=0}^{(N/2)-1} X[2k+1]\tilde{\omega}^{ki} \tag{4.7b}$$

$$\tilde{\omega} = e^{2j\pi/(N/2)} = \omega^2 \tag{4.7c}$$

[3]Other FFT variants allow N to be factorized in different ways, $i.e.$ N does not have to be a power of 2.

The parallelisation of the FFT algorithm in an efficient and scalable manner has been a research topic since parallel computing started. This is because the FFT is a very useful tool in many scientific fields such as time series, wave analysis, convolutions, signal processing, image filtering and the solution of partial differential equations. The two main techniques to parallelise the FFT algorithm are the *Binary-Exchange Algorithm* and the *Transposed Algorithm*. The work of $Gupta$ and $Kumar$ provides a guideline to understand the scalability of the two approaches (Gupta & Kumar 1993). Their analysis shows that the Binary-Exchange algorithm yields good performance on super-computers which have a high communication bandwidth compared to the processing speed of the CPUs. The transposed algorithm is the most widely used today because it provides a good scalability when the processing speed is very high (hundreds of thousands of fast CPUs) compared to the communication network bandwidth. During the last twenty years various authors have presented *ad hoc* implementations of the parallel FFT algorithm (optimised for their machines). In 2003 $Takahashi$ presented an algorithm to perform a 3D FFT using the transposed technique optimising the cache utilisation (Takahashi 2003). $Chan\ et\ al.$ implemented a 3D parallel FFT algorithm for flat cartesian meshes and tested it on a BlueGene/L system in 2008. The algorithm showed good scalability up to thousands of processors (16384 nodes) (Chan et al. 2008). $Ning$ and $Laizet$ developed a FORTRAN library which implements the transposed algorithm (May 2010). The library has been tested on HECToR and Jugene (the German BG/P), showing scalability up to hundreds of thousands of processors (Li & Laizet 2010).

The common understanding is that the transposed algorithm is the most efficient on the current facilities, and the main reasons are:

- it uses the 1D serial FFT algorithm. This routine is generally well-known and it is optimised on each architecture (processors type);
- there is a large number of open-source libraries that implement the 1D serial FFT and the data transposition;
- the application of the transposition method does not require many changes to the serial code. Indeed, from the mathematical point of view, the FFT is performed in a serial fashion;

- the data transposition procedure can be implemented at a different level. As a consequence, the routine that actually performs the serial 1D FFT can be switched (we can exploit the optimised FFT of the machine we are using);
- in a distributed memory architecture, after the communication process, the operations can be performed within a single node. It reduces the number of communications;
- it allows a flexible ratio between the message size and the number of messages, permitting a certain level of optimisation;
- it theoretically permits the overlap of communications and computations, reducing the computational time.

Overall, on the basis of these considerations, we decide to implement the parallelisation of the purely spectral part of the discretisation following the transposition philosophy. We will call this parallel approach *FFT Transposition* in the rest of the chapter.

4.3.1.2 Parallel Algorithm

In practice this approach consists of splitting the N_Z 2D planes (discretised with a spectral/*hp* element method) across a set of $\boldsymbol{P}_Z$ processors. While doing this splitting we have to keep in mind that we need the conjugated modes cosine/sine to be on the same processors to avoid communications when taking first order spectral derivatives (see section 2.1.7). This last remark translates into a simple constraint; $i.e$ we have to maintain an even number of planes per processor. Operations like derivatives and linear system solutions can be performed locally, plane by plane, using routines for 2D spectral/*hp* element discretisation and exploiting the nature of Fourier expansions. In fact the time-integration in Fourier space removes the necessity to transform our variables from physical space to the coefficient space for the harmonic component of the basis (required when solving linear systems or taking derivatives in $z-$direction). The transformation from Fourier to coefficient space can be performed locally for the polynomial part of the basis as Eq. (4.5). Moreover, the orthogonality of the Fourier basis changes 3D elliptic problems into

a series of 2D problems, as reported in section 2.1.7, allowing local solution of planes separately.

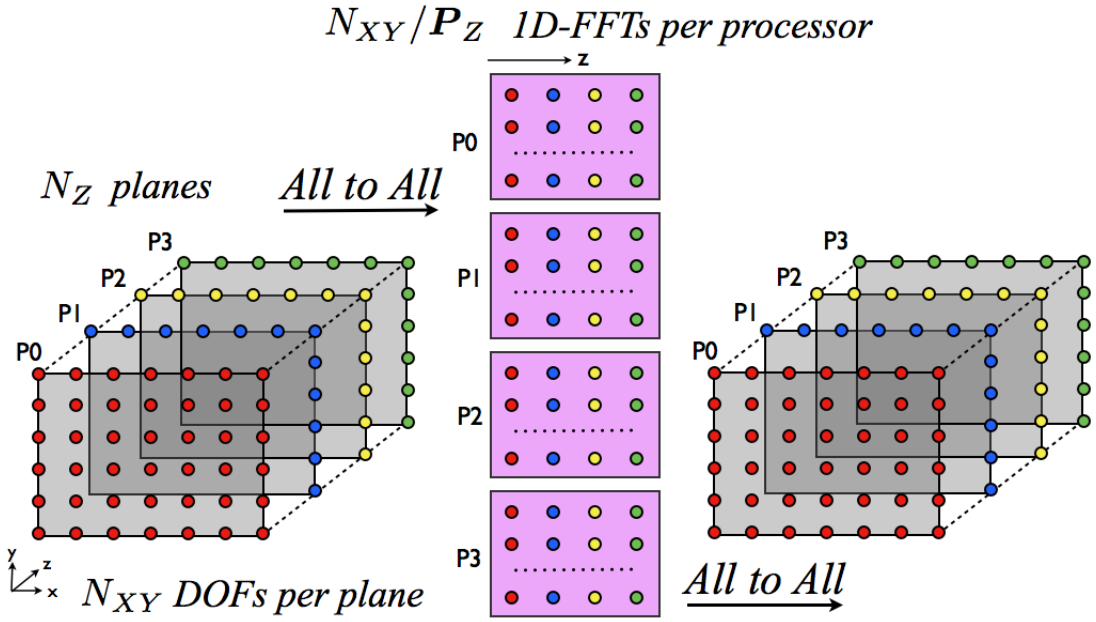


Figure 4.2: Graphical illustration of the FFT Transposition approach. The example considers 4 processors, $N_Z = 4$ and N_{XY} DOFs in the xy-plane. We force each processor to perform the same number of 1D serial FFTs, using padding vectors if necessary. We are not imposing the constraint about the number of planes per processor just for clarity of presentation.

In Fig. 4.2 we provide a general description of the *FFT Transposition* approach. In this representation we remove the constraint on the number of planes per processors, without affecting the main concepts. Actually the constraint is fundamental to reduce communications (transpositions) when the first order spectral derivative is required, as for the advection term. Otherwise, if the problem is purely elliptic, we can have just a single plane per processor because the second derivatives in the spectral direction does not require communications (cosine goes back to be a cosines after being differentiated twice). Starting from an $N_{XY}N_Z$ domain, the data are shuffled and shared as a collection of 'pencils'. A series of 1D serial $FFTs$ is performed in sequence on each group of pencils. The second step is to reshuffle the data among the processors back to the original ordering. It requires double the amount of memory to store the data in a proper order and an MPI $All-to-All$ communication to send data through the communication network. The number of 1D $FFTs$ per processor is kept balanced and equal, distributing $N_{XY}/\boldsymbol{P}_Z$

per processor, whatever the number of planes we initially have per processor. In case the subdivision $N_{XY}/\boldsymbol{P}_Z$ can not been done exactly some padding quadrature points are inserted to balance the computational load on the processors.

Defining the natural bottleneck $\mathcal{B}^{tran}$ as the maximum number of processors we can use, we would say that $\mathcal{B}^{tran} = \max(\boldsymbol{P}_Z) = N_Z$ in general and $\mathcal{B}^{tran} = \max(\boldsymbol{P}_Z) = N_Z/2$ in case of the Navier-Stokes solver (because of the two planes constraint). In the literature it is often found that $\mathcal{B}^{tran} = N$ for this technique also for the solution problems which are not purely elliptic, where N is the number of complex modes. The discrepancy lies in the definition of the basis. In our case we define the Fourier basis as a series of N_Z real modes with $N_Z/2$ different frequencies $k = 0, \ldots, N_Z/2 - 1$. This can be translated as a series of $N_Z/2$ complex modes, therefore matching the common definition of natural bottleneck for the transposed algorithm.

4.3.2 Elemental Parallelisation

The second type of parallelisation can be implemented using a mesh decomposition technique. Domains which have been discretised using an elemental approach naturally suggest a distribution of the computational load across a collection of processors by assigning operations on different elements to different CPUs. If we call N_{el} the number of elements in the mesh, the maximum number of processors we can use to parallelise operations obviously corresponds to N_{el}. In the specific case we are investigating, having series of 2D planes, the natural bottleneck traduces in $\mathcal{B}^{dec} = \max(\boldsymbol{P}_{XY}) = N_{el}^{plane}$. As mentioned in previous sections, the elemental parallelisation affects just the solution of linear systems, that is the Poisson and the three Helmholtz solvers highlighted in Figure 4.1. While basic operations, such as derivatives and physical/coefficient space transformations, can be performed elementally in a spectral/*hp* element discretisation, the solution of a linear system requires information about elements connectivity. Therefore, when the elliptic solvers appear in the time-integration loop and the elements are located on different processors, communication is required to impose the connectivity conditions associated with the spatial discretisation[4]. In this study we take into account basic continuous Galerkin

[4]Continuous and Discontinuous Galerkin projections have different connectivity rules.

projections, hence C^0 continuity between elements needs to be imposed. In the following we briefly describe the solution procedure of a linear system in *Nektar++* to highlight operations and communications.

The linear systems for the elliptic solvers are built in order to apply a static condensation approach (Karniadakis & Sherwin 2005). This technique is based on the boundary-interior decomposition of the polynomial basis we use in the spectral/*hp* element discretisation. Given a typical linear system $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{f}$, we reorder the DOFs $\boldsymbol{x}$ in such a way that we have the boundary degrees of freedom followed by the interior ones as

$$\begin{bmatrix} \boldsymbol{M}_b & \boldsymbol{M}_c \\ \boldsymbol{M}_c^T & \boldsymbol{M}_i \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_b \\ \boldsymbol{x}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_b \\ \boldsymbol{f}_i \end{bmatrix}, \tag{4.8}$$

where the subscripts b and i indicates boundary and interior degrees respectively. Observing the structure of the system, also depicted in Figure 4.3, it is observed that the matrix $\boldsymbol{M}_i$ is block-diagonal as a consequence of the non-overlapping nature of the interior modes. Pre-multiplying by a block matrix of the form

$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{M}_c\boldsymbol{M}_i^{-1} \\ 0 & \boldsymbol{I} \end{bmatrix}, \tag{4.9}$$

we obtain

$$\begin{bmatrix} \boldsymbol{M}_b - \boldsymbol{M}_c\boldsymbol{M}_i^{-1}\boldsymbol{M}_c^T & 0 \\ \boldsymbol{M}_c^T & \boldsymbol{M}_i \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_b \\ \boldsymbol{x}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_b - \boldsymbol{M}_c\boldsymbol{M}_i^{-1}\boldsymbol{f}_i \\ \boldsymbol{f}_i \end{bmatrix}. \tag{4.10}$$

```
// Calculate the new right hand-side vector
(1) g_b = f_b − M_c M_i^{-1} f_i
// Calculate the Schur complement
(2) S = M_b − M_c M_i^{-1} M_c^T
// Solve the linear system
(3) S x_b = g_b
// Calculate the right hand-side vector
(4) g_i = f_i − M_c^T x_b
// Solve the final linear system
(5) M_i x_i = g_i
```

Algorithm 5: Static condensation solution procedure.

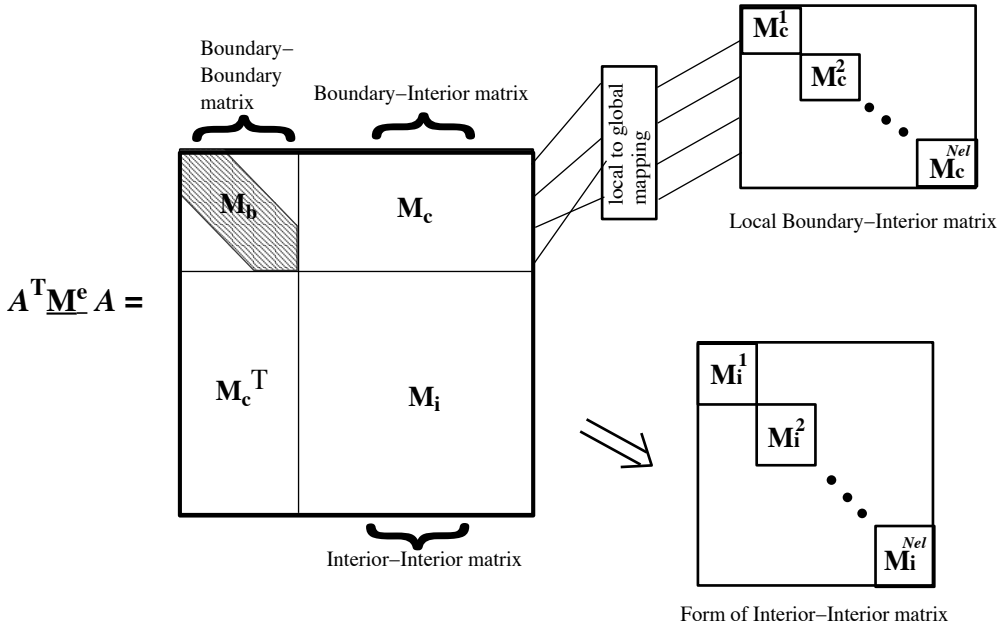


Figure 4.3: A general global matrix pattern after DOFs have been reordered to apply a static condensation approach. Courtesy of $Karniadakis$ and $Sherwin$ (Karniadakis & Sherwin 2005).

The solution of the reduced linear systems at steps (3) and (5) in **Algorithm 5** can be performed using a direct or an iterative method. The system in step (5) can be solved locally, using a direct approach, and it does not require communication. Mesh decomposition techniques play a relevant role in the solution of the boundary DOFs linear system, $i.e.$ step (3).

4.3.2.1 Mesh Decomposition

In the case that all the elements of a 2D plane are located on the same processor, hence we are not decomposing the mesh, the solution of each 2D plane is performed in serial fashion and connectivity between elements is naturally enforced during the global matrix construction. On the other hand, when the mesh is decomposed, communication is required between boundary DOFs lying on the partition boundaries, as shown in Figure 4.4.

Partitioning routines for elemental discretisations are quite common nowadays. The general issue is to optimise the mesh decomposition reducing the number of communicating edges. In *Nektar++* the mesh is partitioned using the open-source library *METIS*

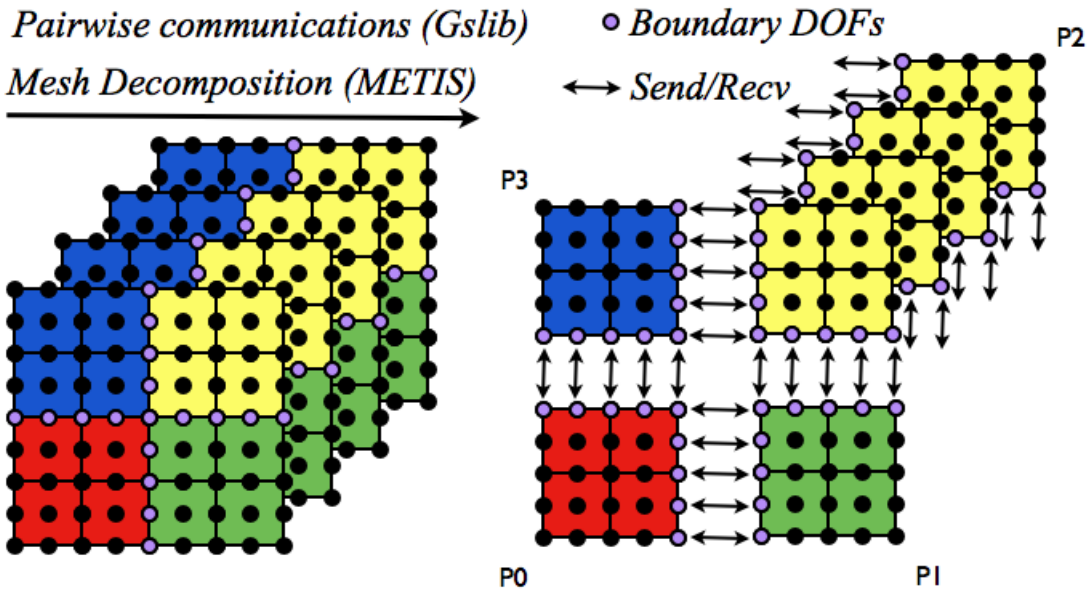


Figure 4.4: Schematic of a mesh decomposition approach. A regular mesh with 16 quadrilaterals is distributed across 4 processors. Pairwise communications are required between the DOFs on the partitions edges during matrix-vector multiplications in the linear systems solutions.

(Karypis 2013). Starting from a dual-graph containing all the elements, a tree-decomposition is applied using internal METIS routines which minimise the number of cuts, $i.e.$ the number of communications required. Once the mesh has been partitioned a universal numbering of the DOFs is used to coordinate messages between MPI processes. This task is achieved using the open-source library *Gslib* by $Fisher$ (Fischer et al. 2008). The selection of these task-specific libraries was dictated by their high efficiency and portability along with their practical implementation which fits into the *Nektar++* structure and layout. A complete description of these libraries is beyond the scope of this chapter, therefore we redirect the interested reader to the cited documentation for further details.

Assuming the mesh partitions are generated by minimising the number of DOFs on their edges and the communication patterns between processes is defined, we are left with the decision on the solution method to employ, $i.e.$ iterative or direct. Theoretically speaking there are no restrictions for this choice. Generally a direct approach is more expensive in terms of memory requirements, yielding to some possible memory constraints when the problem size increases. An iterative approach could require on the other hand more operations, due to its recursive nature, which can be however reduce by tuning the conjugated gradient algorithm tolerance. In this study we consider just an iterative approach for the

parallel solution of the elliptic problems appearing in the time-stepping cycle. Nevertheless, if we are not inserting an elemental decomposition in the 2D planes we compare the iterative approach with a direct solution of the system. In this case each plane is entirely on a single processor and then we solve the system using a Cholesky factorisation coupled with a reverse Cuthill-McKee algorithm (LAPACK) (Anderson et al. 1999).

4.3.2.2 Parallel Algorithm

Since the direct solution of the system is applied just when communications are not required between elements, we describe in the following the iterative algorithm in order to highlight where communications play a role at this level of parallelism. The iterative algorithm to solve a generic linear system in *Nektar++* is based on $Demmel\ et\ al.$ Preconditioned Conjugated Gradient Method (PCGM) (Demmel et al. 1993).

The pseudo-code in **Algorithm 6** illustrates the sequence of operations for the solution of a linear system $\boldsymbol{Ax} = \boldsymbol{b}$. We report the basic steps of this algorithm in order to highlight where the parallelism is introduced. We consider a diagonal preconditioner $\boldsymbol{K}$ and a maximum number of iterations $\mathcal{N}_{iter}^{MAX}$. The iterative procedure stops if the maximum number of iterations is reached or the residual is small enough (below a specified tolerance). We disregard the initial operations, considered as set-up costs and we focus on the **for** loop.

Step (1) to (4) can be performed locally on each processor, hence they do not require communication. A exchange of data is required to perform the reduction of the inner products in step (7) and to evaluate the **if** statement. In these cases the partial inner products are performed locally and then an MPI $All - Reduce$ summation is required to attain the total value of the products. Step (5) would require communication just in case of non-diagonal preconditioners. In our analysis we limit our implementation to the use of diagonal preconditioners to facilitate the modelling of the communications pattern. Most of the communication appears in step (6), where we need to perform a matrix-vector multiplication. At this step messages are required between boundary degrees of freedom lying on the partition boundaries at each iteration, as evident from Fig. 4.4. A detailed description of the communication pattern will be provided in section 4.5.

input : initial guess $\boldsymbol{x}_0$

output: final solution $\boldsymbol{x}$

`// calculate initial residual` $\boldsymbol{r}_0$

$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$

`// solve for` $\boldsymbol{w}_0$ `where` $\boldsymbol{K}$ `is the preconditioner`

$\boldsymbol{K}\boldsymbol{w}_0 = \boldsymbol{r}_0$

`// set parameters`

$\boldsymbol{q}_{-1} = \boldsymbol{p}_{-1} = 0 \qquad \beta_{-1} = 0 \qquad \boldsymbol{s}_0 = \boldsymbol{A}\boldsymbol{w}_0$

$\rho_0 = (\boldsymbol{r}_0, \boldsymbol{w}_0) \qquad \mu_0 = (\boldsymbol{s}_0, \boldsymbol{w}_0) \qquad \alpha_0 = \rho_0/\mu_0$

for $i = 0$ **to** $\mathcal{N}_{iter}^{MAX}$ **do**

(1) $\boldsymbol{p}_i = \boldsymbol{w}_i + \beta_{i-1}\boldsymbol{p}_{i-1}$

(2) $\boldsymbol{q}_i = \boldsymbol{s}_i + \beta_{i-1}\boldsymbol{q}_{i-1}$

(3) $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i\boldsymbol{p}_i$

(4) $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha_i\boldsymbol{q}_i$

if $(\boldsymbol{r}_{i+1}, \boldsymbol{r}_{i+1}) < tolerance$ **then**

break

end

(5) Solve for $\boldsymbol{w}_{i+1}$ the system $\boldsymbol{K}\boldsymbol{w}_{i+1} = \boldsymbol{r}_{i+1}$

(6) $\boldsymbol{s}_{i+1} = \boldsymbol{A}\boldsymbol{w}_{i+1}$

(7) $\rho_{i+1} = (\boldsymbol{r}_{i+1}, \boldsymbol{w}_{i+1}) \qquad \mu_{i+1} = (\boldsymbol{s}_{i+1}, \boldsymbol{w}_{i+1})$

(8) $\beta_i = \rho_{i+1}/\rho_i$

(9) $\alpha_{i+1} = \rho_{i+1}/(\mu_{i+1} - \rho_{i+1}\beta_i/\alpha_i)$

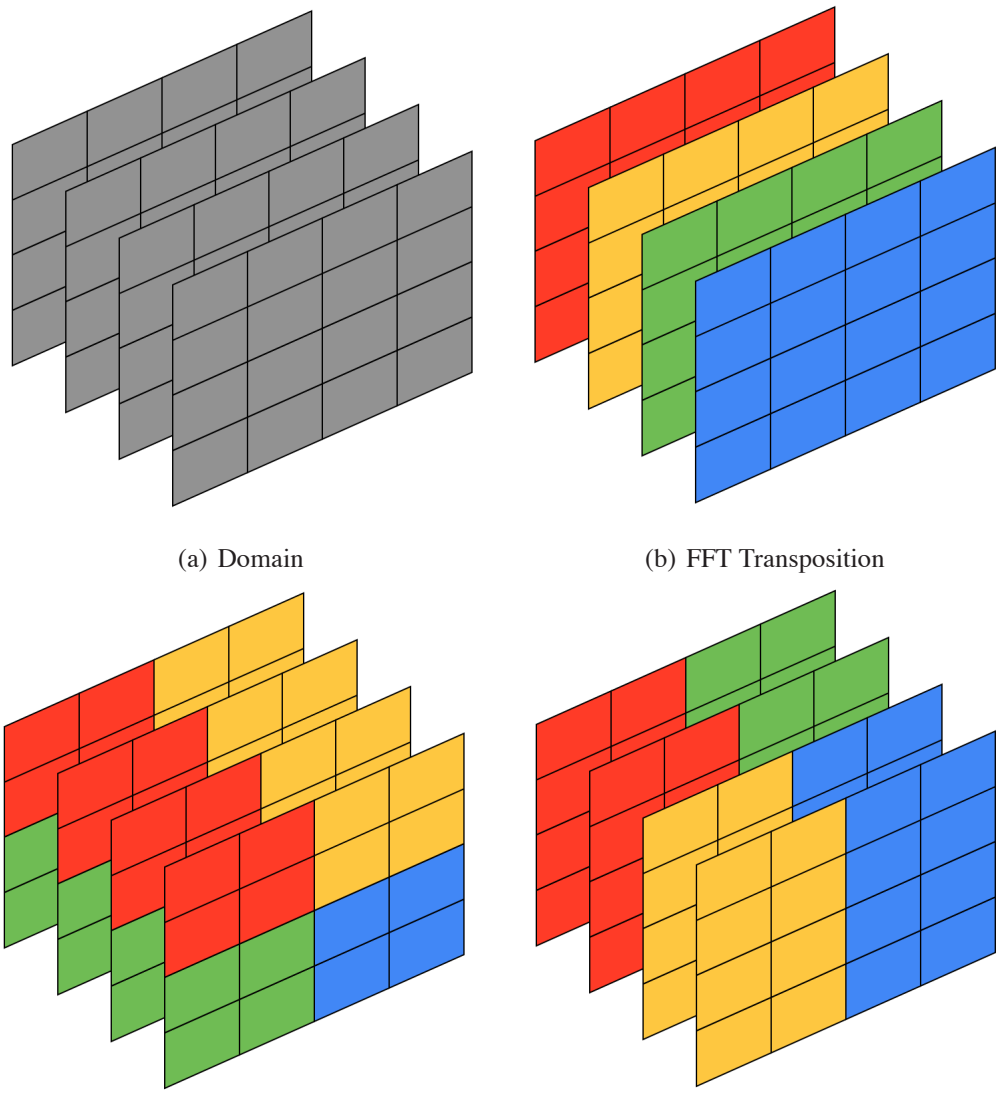
end

Algorithm 6: Preconditioned Conjugated Gradient Method. $Demmel\ et\ al.$ (Demmel et al. 1993).

4.3.3 Hybrid Parallelisation

The natural consequence of having the two previously discussed approaches implemented is to use them concurrently. As reported by $Hamman\ et\ al.$ (Hamman et al. 2007),

combinations of parallelisation approaches can extend the limit in the number of processors that can be adopted. In fact the total bottleneck of the combined techniques is $\mathcal{B}^{tot} = \mathcal{B}^{tran}\mathcal{B}^{dec}$. In Fig. 4.5 we can appreciate how the same initial domain can be decomposed using the modal parallelisation approach (*FFT Transposition*), the elemental parallelisation approach (*Mesh Decomposition*) or both of them at the same time (*Hybrid*).



(a) Domain (b) FFT Transposition

(c) Mesh Decomposition (d) Hybrid

Figure 4.5: Parallelisation strategies visualisation over four processes. The Fourier spectral/*hp* element domain reported in (a) can be decomposed according to the Fourier modes (b) or as an arbitrary decomposition of the 2D mesh (c). A third option is a combined approach (d).

From the implementation point of view, the problem translates into encapsulating in C++ classes the concept of parallelisation in order to facilitate the usage of both techniques. In *Nektar++* the task is achieved through two layers:

- encapsulation of the MPI communicators;
- appropriate usage of those communicators within derived classes.

All communicators derive from an abstract class containing the required methods to transmit data between processors. In the case of parallel execution those methods will be replaced by MPI routines, otherwise by their serial counterparts. The top level communicator can be decomposed, creating a MPI virtual topology (Gabriel et al. 2004). The cartesian virtual topology (rows and columns) shown in Fig. 4.6 facilitates the conceptual handling of the two parallel techniques. The global communicator named m_comm con-

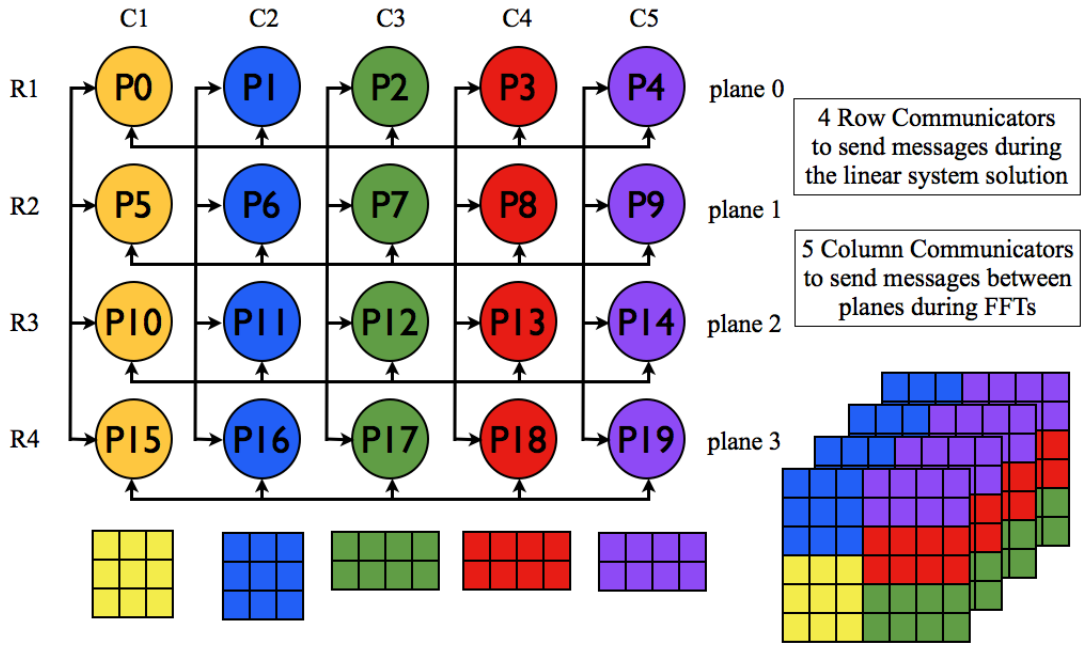


Figure 4.6: Structure of the MPI cartesian communicator for an hybrid parallelisation approach. In this example 20 MPI processes are used to parallelise a Fourier spectral/*hp* element discretisation with 42 elements per plane and 4 planes.

tains all the information about parallelisation and can be easily transmitted/instantiated in derived classes. In the case of the creation of a cartesian MPI communicator, m_comm contains two sub-communicators equipped with the same general methods.

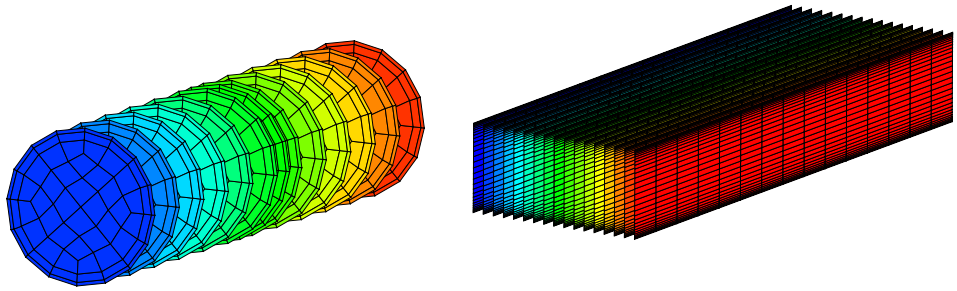
```
m_comm->GetRank()
m_comm->SendRecv(Proc,input,output)
m_comm->GetRowComm()->GetRank()
m_comm->GetRowComm()->Send(Proc,input)
m_comm->GetColumnComm()->GetRank()
m_comm->GetColumnComm()->AlltoAll(input,output)
```

As can be observed in the graphical illustration of the communicator, the mesh partitions are distributed across columns, requiring a specific communicator for each row. The decomposition of the 2D plane is accomplished on the root process just once, $i.e.$ P0 in Fig. 4.6. Subsequently, each partition will be sent to the appropriate process. For example the first partition, the yellow one, will be sent to P5, P10 and P15. The second partition, the blue one, will be sent to P1, P6, P11,P16 and so on. When solving the linear systems arising during the algorithm steps, each row can be considered as a standalone problem, because of the modal decoupling between planes, hence requiring communication just through its row communicator. It also appears from this particular example that P1 is not required to communicate with P2, since the respective partitions do not share DOFs (actually in each row the blue process does not need to communicate with the green one). The composition of all the partitions on a row produces a full 2D plane, but each row refers to a different plane (or set of consecutive planes in the most general case). Therefore operations across planes require another set of communicators, $i.e.$ the column communicators. Before and after an FFT, data needs to be reordered. Following the procedure described in Fig. 4.2. Given that the partitions are identical across a column, data shuffling is bounded within each column communicator.

4.4 Test Cases

In order to study the performance of the different parallelisation approaches discussed in section 4.3 we need to select some tests cases. We focus on turbulent simulations making use of the turbulent flow verifications presented in section 2.3.2. Since we are now interested in monitoring the parallel computational time when running different approaches, we simplify the problem set up. Initial conditions are applied via a restart file containing

the already converged turbulent states we reached in previous simulations. This choice removes an initial overhead of PCGM iterations due to the white noise effect on the residual. No stabilisation techniques are applied, since the simulations do not require them once they have passed the transition phase. The advection term is treated in its convective form and a first order IMEX scheme is used to time-step the incompressible Navier-Stokes equations. In Fig. 4.7 an overview of the domain discretisation is provided, where the plane and element distribution is highlighted. As mentioned in section 2.3.2 the fluid flows in the $z-$direction in the pipe and in the $x-$ direction for the channel. In Table 4.2 we concentrate on the features of both discretisations in terms of DOFs and bottlenecks.



(a) Pipe (b) Channel

Figure 4.7: Domain discretisation structure of the turbulent test cases. Pipe flow discretisation (a) and channel flow discretisation (b).

Table 4.2: Turbulent test case discretisation features. The total bottleneck $\mathcal{B}^{tot} = \mathcal{B}^{tran}\mathcal{B}^{dec}$.

Test case	P	N_{el}^{plane}	N_Z	N_{el}	N_{XY}	N_{TOT}	$\mathcal{B}^{tran}$	$\mathcal{B}^{dec}$	$\mathcal{B}^{tot}$
Pipe	7	64	128	8192	5184	663552	64	64	4096
Channel	6	450	64	28800	28800	1843200	32	450	14400

We present in the following results obtained through numerical experiments. All the simulations have been run on the Imperial College cluster CX2, an SGI Altix ICE 8200 EX system. Among the possible choices of node type, we selected the 8-core nodes, which allows the utilisation of 512 cores in total (64 nodes). Each one of the 8 cores in a node is a Nehalem CPU running at 2.93 GHz. The memory per node is 24 GBytes (fast

memory) and the system has dual-rail *Infiniband* interconnect. The operating system is Linux with kernel version 3.0.58-0.6.6.

4.5 Scalability Model

Creating a scalability model consists of quantifying the number of operations and communications involved in the execution of a parallel algorithm and, based on the hardware, predicting the computational time required to solve a specific problem. The first distinction when building a scalability model is between the time spent in performing operations and the time spent for communication. While it is important to avoid operation duplication among processes, the real challenge is to minimise the communication costs, since this is often the cause of poor scaling. If we denote by T_{Oi} the time required to perform the $i-th$ operation in the algorithm we want to model and T_{Cj} the time required for the $j-th$ communication, we can define the total time T required for the parallel execution as

$$T = \sum_i T_{Oi} + \sum_j T_{Cj}. \tag{4.11}$$

The modelling of T_{Oi} consists of the operation count throughout the algorithm execution. Such a type of quantification can be achieved commonly by measuring the number of operations associated with basic pieces of code, $e.g.$ matrix-vector multiplications, inner products, vector-vector summations etc. In the following we will evaluate operations at the elemental level. The DOFs associated with a 2D element can be easily calculated based on the polynomial expansion employed in the discretisation. Practically, each element is defined with $(P+2)^2$ quadrature points and $(P+1)^2$ DOFs (expansion coefficients). While DOFs indicate the number of coefficients of the polynomial expansion, hence the problem unknowns, quadrature points are more indicative of the problem size for operations in physical space. Vector-vector summations and multiplications are based on the variables physical representation within the advection operator (quadrature points) and on their DOFs counterpart during the linear system solution. The matrix-vector multiplication in *Nektar++* can be obtained throughout a series of algorithmic variants, as also mentioned in the previous chapter. For this study we force all

matrix-vector multiplications to take advantage of the sum-factorisation technique, which requires $(4P^3 + 18P^2 + 26P + 12)$ operations per element as reported by $Vos\ et\ al.$ (Vos 2010). The number of operations then need to be scaled with respect to the time required on a specific machine to perform an operation. We name this time t_O and it is quantified as seconds to perform an operation.

Communication costs are generally more complex to model. The first consideration is that they strongly depend from the hardware configuration. The cluster topology[5] plays a relevant role in the data exchange between processes. Possible layouts such as a mesh-topology, a hypercube-topology or a ring-topology require *ad hoc* modelling to take into account the physical message path. In this study we follow the most common approach when estimating communication costs (Hamman et al. 2007, Evangelinos & Karniadakis 1996, Tufo & Fischer 2001), $i.e.$ assuming a "flat" topology with no contention between messages. Under this hypothesis, the time T_C for a set of messages to be transmitted and received can be modelled as

$$T_C = Number\ of\ messages\ \times \Big[\tau_L + Message\ Size\ \times \tau_B\Big] \tag{4.12}$$

where we call τ_L the latency, measured in seconds $[s]$ and τ_B the inverse of the bandwidth. Citing what is reported on most MPI manuals we define

Latency (τ_L)

The overhead associated with sending a zero-byte message between two MPI tasks. Total latency is a combination of both hardware and software factors, with the software contribution generally being much greater than that of the hardware. It is usually measured in milli/microseconds.

Bandwidth $(1/\tau_B)$

The rate at which data can be transmitted between two MPI tasks. Like latency, bandwidth is a combination of both hardware and software factors. It is usually measured in bytes/megabytes per second.

[5] A network topology is defined as the arrangement of computers in a network. Practically it defines how the computers, or nodes, are connected to each other.

We suppose each DOF is to be represented by a double in C++, hence an 8-byte information packet. Therefore τ_B, instead of being quantified using its canonical unit of measure $[s/Mbytes]$, will be translated in $[s/DOFs]$ to facilitate substitutions during the modelling phase. In order to map the features of the employed machine, we sample latency and bandwidth via the MPI benchmarking application **IBM-MP1**. Running a set of basic MPI message-passing routines, **IBM-MP1** samples the latency (the time required to send/receive a 0-byte message) and the bandwidth, monitoring the time required to conduct a communication for various message sizes. The timings are performed on various routines like $Alltoall$, $Allgather$ and $Sendrecv$. The typical output of an **IBM-MP1** sampling for the $Sendrecv$ routine on 16 processes is reported below. The latency is the average time for the 0-byte message and bandwidth is calculated for messages of size 1 byte to 4194304 bytes. The difference in bandwidth for the different message sizes (especially after 1024 bytes), suggests that the packet-size in which the total message is decomposed for transmission is not always optimal.

```
#---------------------------------------------------------------------------
# Benchmarking Sendrecv
# #processes = 16
#---------------------------------------------------------------------------
       #bytes #repetitions  t_min[usec]  t_max[usec]  t_avg[usec]   Mbytes/sec
            0         1000         0.85         0.86         0.85         0.00
            1         1000         0.85         0.85         0.85         2.23
            2         1000         0.84         0.84         0.84         4.54
            4         1000         0.89         0.89         0.89         8.53
            8         1000         0.89         0.89         0.89        17.11
           16         1000         0.83         0.83         0.83        36.63
           32         1000         0.85         0.86         0.85        71.37
           64         1000         1.01         1.02         1.01       120.24
            .            .            .            .            .            .
            .            .            .            .            .            .
        65536          640        40.10        40.31        40.21      3100.66
       131072          320        68.18        68.84        68.51      3631.71
       262144          160       123.69       125.93       124.82      3970.42
       524288           80       228.95       238.12       233.57      4199.50
      1048576           40       455.65       513.20       482.10      3897.10
      2097152           20      1068.75      1250.00      1175.17      3199.99
      4194304           10      2198.60      2834.49      2498.70      2822.38
#---------------------------------------------------------------------------
```

For practical usage we unify the values of latency and bandwidth by their arithmetic means spanning different messages size and routines. The average value for the bandwidth is $1.64 \cdot 10^3\ MB/s$ which, after manipulation, translates into $\tau_B = 4.87 \cdot 10^{-9}\ s/DOFs$. For the latency we obtain $\tau_L = 2.09 \cdot 10^{-6}\ s$.

4.5.1 Advection Term Modelling

We start the modelling of the advection term costs by quantifying the time required to perform all the $FFTs$. As stated in section 4.3.1, overall nine $FFTs$ are required. Each one of these $FFTs$ is composed of a set of $1D-FFTs$, one for each pencil. The number of pencils on each column communicator is defined by the number of quadrature points associated with the local mesh partition. Assuming the mesh is evenly partitioned, we can quantify the number of pencils on each column communicator as $(P+2)^2 N_{el}^{plane}/\boldsymbol{P}_{XY}$. Given that the number of pencils will be balanced among the $\boldsymbol{P}_Z$ processes and that a single FFT costs $N_Z log_2(N_Z)$, we can summarise the cost of the 9 $FFTs$ as

$$T_{O1}^A = t_O \cdot 9\Big[\frac{N_{el}/N_Z}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}}(P+2)^2 N_Z log_2(N_Z)\Big]. \tag{4.13}$$

The elemental cost of calculating the physical derivatives will be proportional to the cost of executing a matrix-vector multiplication, where the matrix is a general derivative matrix. Therefore, the number of operations for an elemental matrix-vector multiplication will be scaled with respect to the number of elements on the process, yielding to

$$T_{O2}^A = t_O \cdot 9\Big[\frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}}(4P^3 + 18P^2 + 26P + 12)\Big]. \tag{4.14}$$

Each advection component then requires 9 vector-vector multiplications to calculate the $u_j \partial u_i/\partial x_j$ factors. This term can be modelled starting from the elemental size of each vector, hence the number of quadrature points $(P+2)^2$, and then by multiplying it for the number of elements on each process as

$$T_{O3}^A = t_O \cdot 9\Big[\frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}}(P+2)^2\Big]. \tag{4.15}$$

Finally we need to take in account the 6 vector-vector summations required to build the three advection term components $N(u_i)$. For each component i we have to sum the three $u_j \partial u_i / \partial x_j$ (with $j = 0, 1, 2$), therefore two summations per component. Applying the same approach we described for the previous term, where the number of operations is directly proportional to the vectors size, we obtain

$$T^A_{O4} = t_O \cdot 6 \Big[\frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}} (P+2)^2 \Big]. \tag{4.16}$$

Once we have quantified the number of operations and their computational cost, we are left with the communication costs modelling. Within the advection term routines, communication is required during the 9 $FFTs$ only, for shuffling and reshuffling data between processes belonging to the same column communicator. We apply the communication model described in Eq. (4.12). For each FFT two MPI $Alltoall$ calls are needed (shuffling and reshuffling) which formally require $(\boldsymbol{P}_Z - 1)$ messages, as also stated in (Hamman et al. 2007, Evangelinos & Karniadakis 1996, Tufo & Fischer 2001). The messages size can be calculated by recalling that the number of pencils will be balanced across the $\boldsymbol{P}_Z$ process, yielding to

$$T^A_{C1} = 18(\boldsymbol{P}_Z - 1) \Big[\tau_L + \frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}} (P+2)^2 \tau_B \Big]. \tag{4.17}$$

The very last step is to put together all the contributions mentioned above to create a full model for the advection term calculation cost as

$$T^A = t_O \cdot \sum_i T^A_{Oi} + \sum_j T^A_{Cj} = t_O \cdot T^A_O + T^A_C, \tag{4.18}$$

where $T^A_C = T^A_{C1}$. In addition, neglecting the terms not depending on P or N_Z, we have

$$T^A_O = \frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}} \Big(36P^3 + 176P^2 + 290P + 9P^2 log_2(N_Z) + 18P log_2(N_Z) + 36 log_2(N_Z) \Big). \tag{4.19}$$

4.5.2 Elliptic Solver Modelling

In contrast with the advection routine, the vector operations during the linear system solution are performed in coefficient space, hence $(P+1)^2$ is the typical vector dimension. All operations are carried out elementally, therefore each process deals with the assigned N_{el}^{loc}. Steps (1), (2), (3) and (4) in **Algorithm 6** can be modelled using the same logic applied in the previous section. At each step we have one scalar-vector multiplication and one vector-vector summation yielding

$$T_{O1}^{E} = t_O \cdot 8\Big[\frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}}(P+1)^2\Big]. \tag{4.20}$$

The two inner products appearing at steps (7) of **Algorithm 6** and the inner product required during the evaluation of the stopping criteria consist of a vector-vector multiplication and a sum reduction. The vector-vector multiplication requires $(P+1)^2$ operations while the sum reduction $(P+1)^2-1$. In order to simplify the model we assume that both components of the inner product routine require $(P+1)^2$ operations, leading to

$$T_{O2}^{E} = t_O \cdot 6\Big[\frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}}(P+1)^2\Big]. \tag{4.21}$$

The diagonal preconditoner can be considered a vector-vector multiplication and, as a consequence, step (5) of **Algorithm 6** can be modelled as

$$T_{O3}^{E} = t_O \cdot \Big[\frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}}(P+1)^2\Big]. \tag{4.22}$$

The most expensive step of **Algorithm 6** is the application of the matrix system (6). In this case we reuse the sum-factorisation operation count defined earlier in this section and we quantify the number of operations as

$$T_{O4}^{E} = t_O \cdot \Big[\frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}}(4P^3 + 18P^2 + 26P + 12)\Big]. \tag{4.23}$$

Communications appear during the inner product reductions and during the matrix-vector

multiplication. The inner product reduction can be modelled using the $Allgather$ model also used in (Hamman et al. 2007). The number of messages is $(\boldsymbol{P}_{XY} - 1)$ for each inner product, since the local reductions need to be composed into a global reduction, which happens on one processor. The message size is simply one DOF because the results of the local reductions is a number which needs to be sent to the process taking care of the final summation. This results in the communication time

$$T^E_{C1} = 3(\boldsymbol{P}_{XY} - 1)\Big[\tau_L + \tau_B\Big]. \tag{4.24}$$

Communications during matrix-vector multiplication are more complex. The number of messages required between mesh partitions can be quantified as $2log_2(\boldsymbol{P}_{XY})$. This communications count, typically considered when determining the number of messages for mesh decomposition approaches (Hamman et al. 2007, Tufo & Fischer 2001), derives from the tree-graph nature of common mesh partitioners. Given that each mesh partition translates into a graph node connected to other nodes via edges and given that minimisation of connections is usually obtained via recursive bisection algorithms, we can easily infer that the number of connections will be proportional to $log_2(\boldsymbol{P}_{XY})$.

On the other hand forecasting the message size is not really possible. In fact each edge of the graph can be comprised of a varying number of DOFs and it will depend on the domain nature. To formalise a prediction we assume that each partition will have the maximum possible number of communicating edges. In Fig. 4.8 we show that the maximum number of edges for a N^{loc}_{el} partition is $\propto 2(N^{loc}_{el} + 1)$; under the assumption that the partition is the middle of the mesh, therefore none of its edges are on the domain boundaries. Observing in Fig. 4.8 the possible shapes of a partition as we increase the number of elements we can simply count the number of edges on the boundaries. Collecting together previous considerations, we can apply the communication model descried in Eq. (4.12) obtaining

$$T^E_{C2} = 2log_2(\boldsymbol{P}_{XY})\Big[\tau_L + 2\Big(\frac{N_{el}}{\boldsymbol{P}_Z\boldsymbol{P}_{XY}} + 1\Big)(P+1)\tau_B\Big]. \tag{4.25}$$

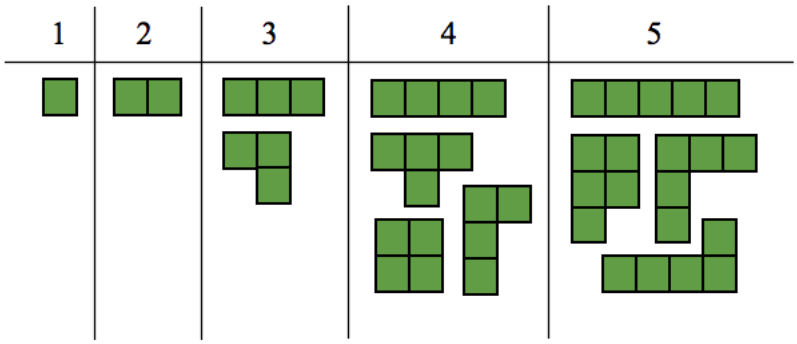


Figure 4.8: Overview of how a partition containing N_{el}^{loc} can be cast. The different groupings suggest that the maximum number of edges which may require communication is $\propto 2(N_{el}^{loc}+1)$

The very last step, as before, is to put together all the contributions mentioned above to create a full model for the elliptic solver cost as

$$T^E = t_O \cdot \sum_i T_{Oi}^E + \sum_j T_{Cj}^E = t_O \cdot T_O^E + T_C^E, \tag{4.26}$$

where $T_C^E = T_{C1}^E + T_{C2}^E$. Neglecting the terms not depending on P or N_Z T_O^E is

$$T_O^E = \frac{N_{el}}{\boldsymbol{P}_Z \boldsymbol{P}_{XY}} \Big(4P^3 + 33P^2 + 56P \Big). \tag{4.27}$$

4.5.3 Incompressible Navier-Stokes Model

Recalling the Navier-Stokes solver algorithm described in Fig. 4.1, we can build up the total cost T^{NS} of one step of the solution cycle as

$$T^{NS} = a \cdot T^A + b \cdot \mathcal{N}_{iter}^{Poisson} \cdot T^E + 3 \cdot c \cdot \mathcal{N}_{iter}^{Helmholtz} \cdot T^E \tag{4.28}$$

The coefficients $a,\ b,\ c$ will be set in next section via a calibration process. Those coefficients concentrate and encapsulate all the unpredictable issues, such as memory contentions, missing details from the model and machine specific features. Calibration will then be required every time a different machine is used. The number of iterations $\mathcal{N}_{iter}$ for the elliptic solvers will vary depending on the problem nature and between solutions of Helmholtz and Poisson equations. The preconditioner also plays a fundamental role in

the reduction of $\mathcal{N}_{iter}$. Basic diagonal preconditioners, as the one accounted in this study, generally show poor performance in terms of reducing the number of iterations. For the presented turbulent simulations, typical values are $\mathcal{N}_{iter}^{Poisson} \sim 80$ and $\mathcal{N}_{iter}^{Helmholtz} \sim 10$. Introduction of suitable preconditioners would certainly speed-up the simulations, making the following results an upper bound on practically achievable performances.

4.5.4 Calibration

As anticipated in section 4.5.3, calibration consists in assigning a value to those coefficients which, within the model, take into account all the issues that can not be predicted or strictly modelled. In Eq. (4.28) we introduced a first estimate for the incompressible Navier-Stokes solver scalability model, which is however not suited for a quick and efficient calibration. Given that the scalability model should generally be easy to use and to calibrate, we decide to simplify what is reported in Eq. (4.28) as

$$T_c^{NS} = a_1 \cdot t_O \cdot T_O^A + a_2 \cdot T_C^A + b_1 \cdot t_O \cdot T_O^E + b_2 \cdot T_C^E. \tag{4.29}$$

where we collapse the 3-Helmholtz and the Poisson elliptic solver contributions into two terms, where computations and communications are highlighted.

Calibration is then performed by running timing tests where the two parallelisation techniques are applied separately. Monitoring the computational time of these simulations we calculate the required coefficients, which are

$$t_O = 0.9 \cdot 10^{-6} \quad a_1 = 0.5 \quad a_2 = 0.2 \quad b_1 = 3.5 \tag{4.30}$$

and

$$b_2 = \begin{cases} 400 & if\ N_{el}^{plane}/\boldsymbol{P}_{XY} < 4 \\ 10 & otherwise \end{cases} \tag{4.31}$$

where b_2 has two different values because the actual bottleneck for the mesh decomposition techniques is appearing before the theoretical one. In fact, the current implementation can reach a level of granularity of four elements per processor, and not less. This limitation is associated with the libraries selected to handle the the mesh decomposition

technique (METIS and *Gslib*) and the lack of overlap between communication and computation during the linear system solution.

Application of the calibrated model to the turbulent simulations of section 4.4 suggests a good prediction capability. Fig. 4.9 shows a comparison between the actual time required to run one step of the turbulent pipe flow simulation using various approaches (black lines) and what is predicted by the model (red solid line), suggesting good agreement between the model and the real data.

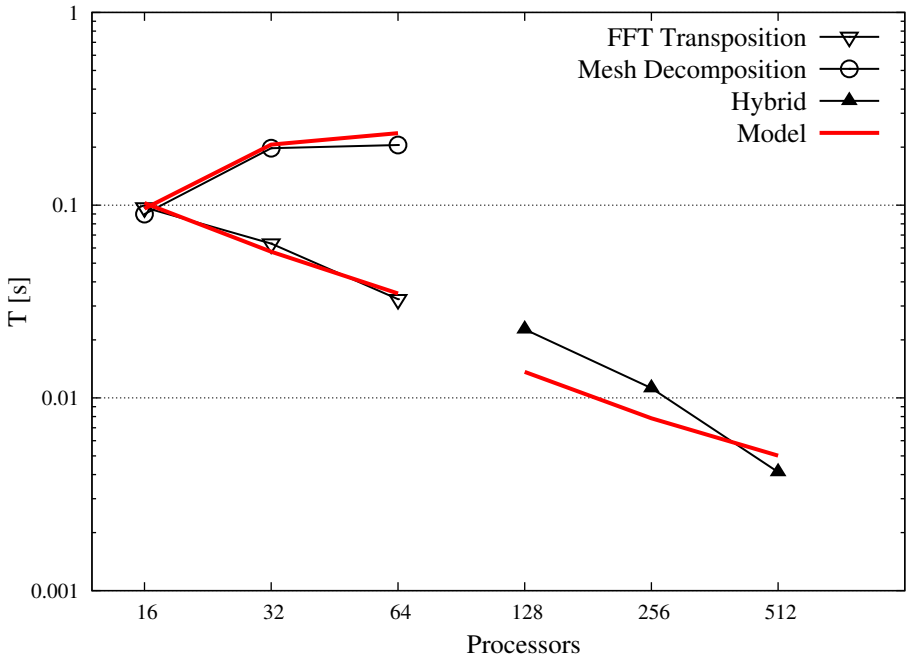


Figure 4.9: Scalability model calibration. On the y-axis the time required to perform one cycle of the solution process reported in Fig. 4.1. The model of Eq. (4.29) after calibration (red solid line) is compared with the measured times for the turbulent pipe flow test case.

4.5.5 Limitations

The scalability model presented in this section, and that will be used to predict performance in section 4.5.6, is affected by a series of limitations. As remarked above, we have disregarded some pieces of the algorithm in order to focus on the two main routines, namely the advection and the elliptic operators. This is a typical approach when creating a scalability model (Hamman et al. 2007), although it may introduce some errors. Furthermore, we have removed the number of iterations $\mathcal{N}_{iter}$ from the model to simplify.

The calibration has been carried out by monitoring the solution time on the SGI Altix ICE 8200 EX system described in section 4.4, therefore the current coefficients must be considered specific for that machine. Series of 1000 samples has been taken for each test case during calibration and numerical tests. The values presented are based on the arithmetic mean of those samples.

Finally, we would like to stress that the model, and therefore the considerations deriving from it, is specific to *Nektar++* . In fact the lack of accuracy when predicting hybrid parallelisation solution mainly derives from possibly non-optimal code implementations and from external library selection ($e.g.$ MPI version, METIS, etc.). However, despite the strong dependence of the results on our implementation, the study provides some generic understanding and suggests overall guidelines on how to address typical issues arising when a specific problem needs to run efficiently on a parallel machine.

4.5.6 Performance Prediction

We recall that $\boldsymbol{P}_{XY}$ and $\boldsymbol{P}_Z$ correspond to the number of processes employed for the elemental and the modal parallelisation respectively. In the following we denote pairs of these values with the notation $(\boldsymbol{P}_{XY}, \boldsymbol{P}_Z)$, since they act as cartesian coordinates in the following graphs. In Fig. 4.10 and Fig. 4.11 we can observe the computational time (expressed in seconds) for one step of the cycle. The surfaces indicate the time will be reduced using different combinations of parallel approaches. In fact both figures suggest that a hybrid approach can reduce the computational time and potentially extend the bottlenecks of standalone approaches. The practical mesh decomposition bottleneck, appearing when we send less than four elements per processor, can be observed for both the pipe and channel simulation. In fact we can observe in Fig. 4.10 and Fig. 4.11 a jump at $\boldsymbol{P}_{XY} = 16$ (pipe case) and $\boldsymbol{P}_{XY} \approx 128$ (channel case) respectively. We would like to recall that the current mesh decomposition bottleneck does not match the expected theoretical performance (one element per processor) as highlighted in section 4.5.4. While Fig. 4.10 shows a steeper time-reduction in the $\boldsymbol{P}_Z$ direction, Fig. 4.11 suggests that, for problems with an high number of elements per plane, increasing $\boldsymbol{P}_{XY}$ is the fastest path to reach higher levels of scalability.

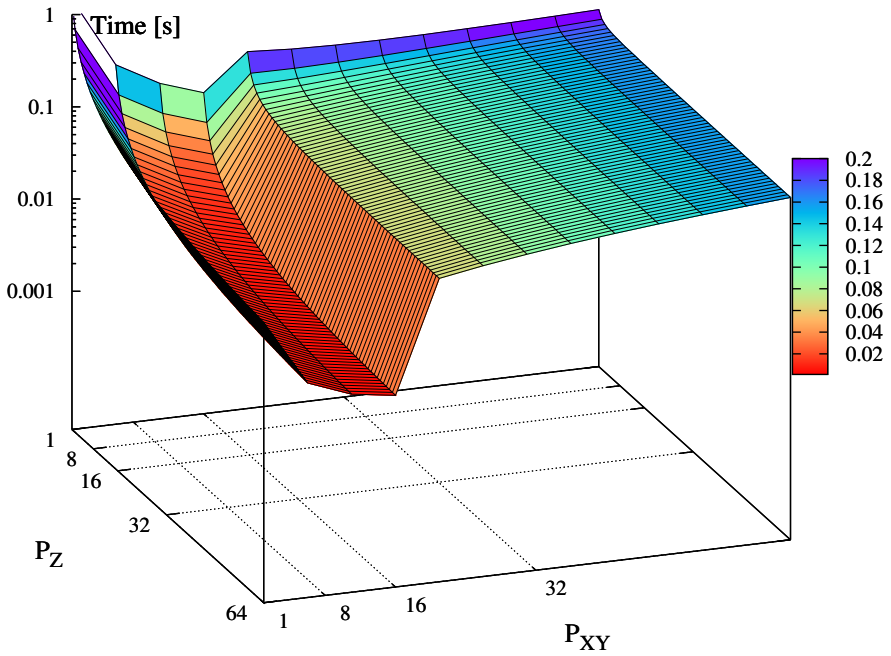


Figure 4.10: Computational time prediction for the turbulent pipe flow using Eq. (4.29). Practical bottleneck for the mesh decomposition technique is clearly visible at $\boldsymbol{P}_{XY} = 16$.

These trends are consistent with the intuitive understanding that having few elements per plane and a lot of Fourier modes promotes the usage of an FFT transposition routine (pipe example). On the other hand, when we have a lot of elements per plane and few planes (channel example), a mesh decomposition technique may be preferred. When applying an FFT transposition to a problem with a lot of elements per plane (and eventually also an high polynomial expansion) we need to consider the number of DOFs which require transposition. The number of pencils, described in Fig. 4.2, is proportional to the number of elements and the polynomial order. As a consequence the amount of data communicated can increase drastically as N_{el}^{plane} and P increase, possibly saturating the system bandwidth.

The three-dimensional representations of the time required for one step of the cycle provide qualitative indications of the algorithm behaviour. In Fig. 4.12 we propose the speed-up maps based on the model prediction as evidence of the parallel performance in a more quantitative style. Speed-up is defined as the ratio between the time required to perform a step of the cycle using one processor and the time required using a combination of $(\boldsymbol{P}_{XY}, \boldsymbol{P}_Z)$. These maps reinforce the overall understanding that a mesh with few ele-

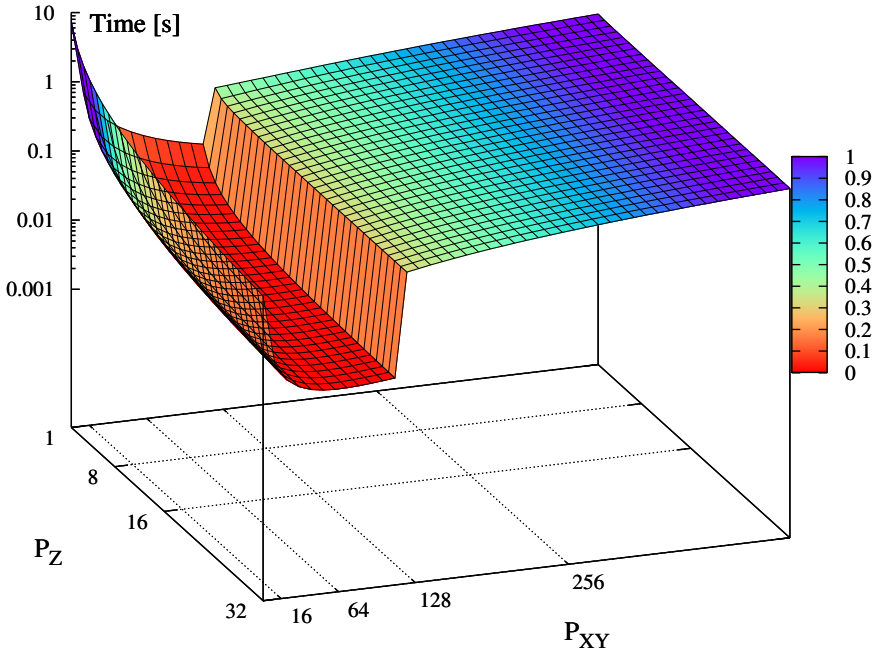


Figure 4.11: Computational time prediction for the turbulent channel flow using Eq. (4.29). Practical bottleneck for the mesh decomposition technique is clearly visible at $\boldsymbol{P}_{XY} \approx 128$.

ments per plane and a lot of planes scales better if we use an FFT transposition approach. In fact an inspection of Fig. 4.12 denotes that the iso-speed-up lines (black solid) are more deformed in the $\boldsymbol{P}_Z$ direction for the pipe simulation than for the channel. Moreover, observing Fig. 4.12(a), the isolines suggest the a mesh decomposition technique can not reach on its own the same level of speed-up of the FFT transposition approach for this test case.

Fig. 4.12 suggests another consideration, $i.e.$ using a hybrid approach is generally more convenient than using a single parallelisation technique, even within the scalability limits of the single parallel approaches. This conclusion derives from observations of the iso-speed-up lines. We can analyse, for example, the isoline in Fig. 4.12(a) highlighted using black dots (it starts at coordinates $(\boldsymbol{P}_{XY} = 16, \boldsymbol{P}_Z = 2)$, corresponding to $\boldsymbol{P}_{TOT} = 32$). If we follow this line down to where it intersects the $\boldsymbol{P}_Z$-axis we appreciate how a pure FFT transposition approach would require $\boldsymbol{P}_Z = \boldsymbol{P}_{TOT} > 32$ processors to achieve the same level of speed-up. This result is plausible under the hypothesis we made for the scalability model. However it is purely an artefact of the model, since communications are assumed to be contention-free. In reality communications are not

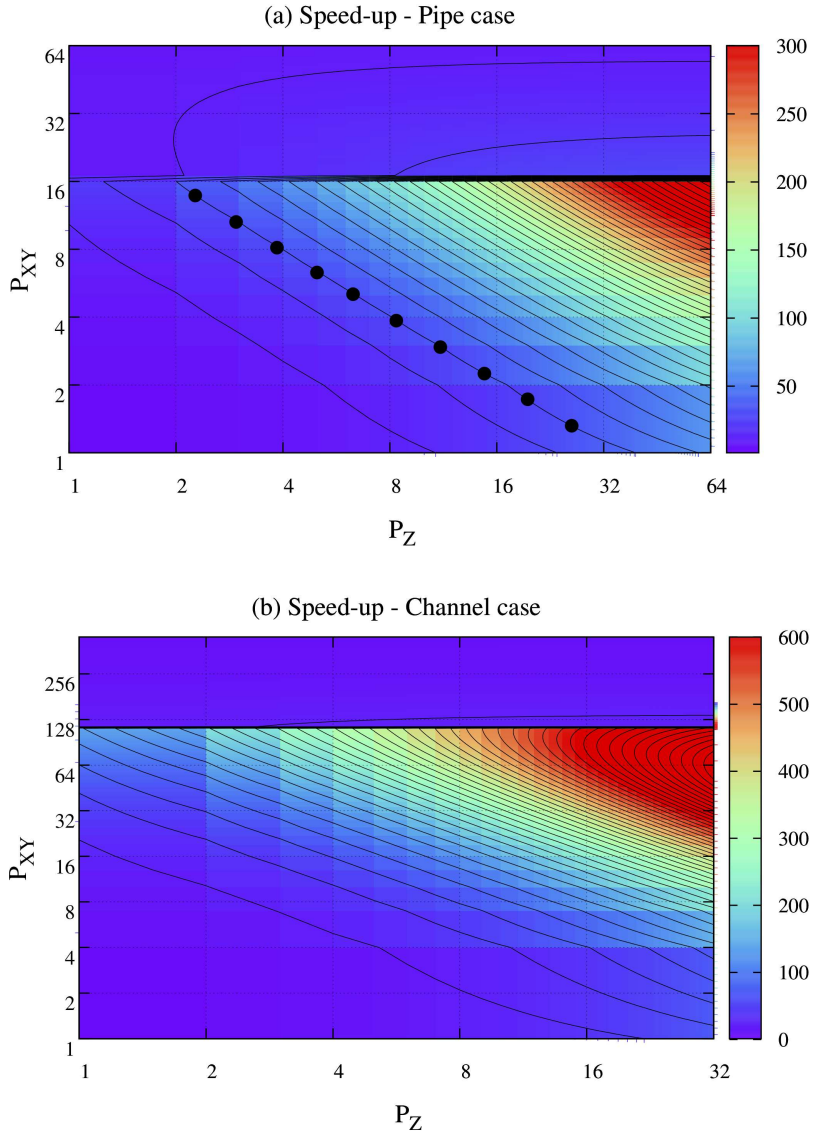


Figure 4.12: Speed-up prediction for the turbulent pipe flow (a) and turbulent channel flow (b) using the model described in Eq. (4.29). Black solid lines indicate points with same speed-up (iso-speed-up lines). Speed-up is defined as $S = T_c^{NS}(\boldsymbol{P}_{XY} = 1, \boldsymbol{P}_Z = 1)/T_c^{NS}(\boldsymbol{P}_{XY}, \boldsymbol{P}_Z)$.

contention-free and the physical outline of the machine plays a role in delaying/blocking messages between nodes. Mixing parallel approaches when it is not necessary generally yields to performance deprecation compared to the most appropriate standalone parallel implementation. The intuitive consequence is that a hybrid approach will have a speed-up which is always in between the speed-up of the standalone techniques.

4.6 Numerical Experiments

In this section we present some numerical experiments we performed on the SGI Altix ICE 8200 EX system described in section 4.4 for both the turbulent pipe flow and the turbulent channel flow. The aim of these experiments is to prove the validity of the assumptions we made in previous sections. The results presented are obtained using *Nektar++* version 3.3.0 (Kirby & Sherwin 2006*b*), averaging, in each experiment, 1000 samples of the solution cycle timing. We compare, in the following, the parallel solution using both an iterative and a direct method for the pure FFT transposition approach. The different strategies are identified as:

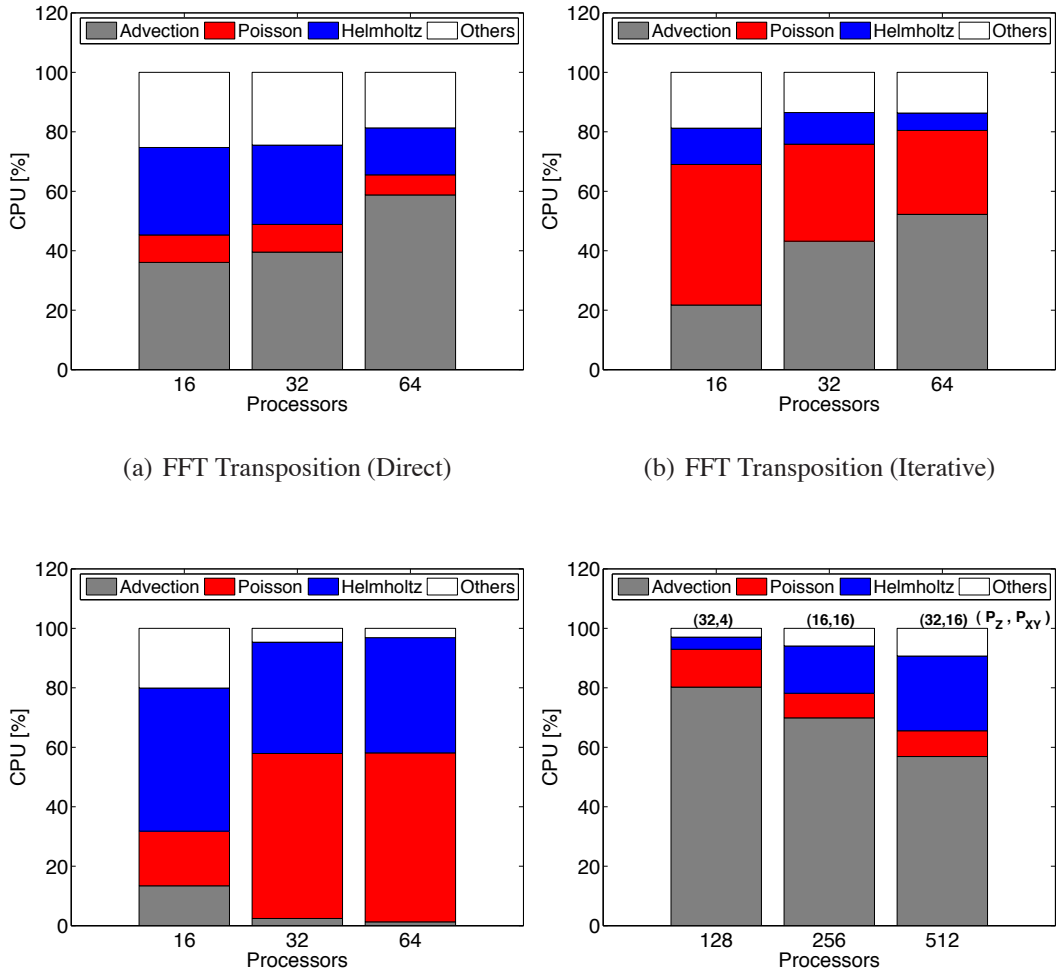
- FFT Transposition (Iterative), when solving the linear system using an iterative method;
- FFT Transposition (Direct), when solving the linear system using an direct method (LAPACK);
- Mesh Decomposition (Iterative), since we do not consider direct solution of the linear system when the mesh is decomposed over multiple partitions;
- Hybrid, when we combine FFT Transposition (Iterative) and Mesh Decomposition (Iterative).

The speed-up calculations are scaled, in each experiment, by the 16-core run using the FFT Transposition (Iterative) approach.

4.6.1 Turbulent Pipe

The first example is the solution of the turbulent pipe flow presented in section 2.3.2. In Fig. 4.13 we show the average percentage of time spent in the routines composing the Navier-Stokes algorithm of Fig. 4.1. As anticipated, most of the time is employed for the advection term calculation and the solution of the linear systems (75% or more).

In Fig. 4.13(a) and (b) we observe how the total computational time is distributed across the routines when solving the linear systems (a) directly or (b) iteratively. The parallelisation approach in these cases is identical but the linear systems arising from the



(a) FFT Transposition (Direct)

(b) FFT Transposition (Iterative)

(c) Mesh Decomposition (Iterative)

(d) Hybrid

Figure 4.13: Turbulent pipe flow parallel simulation - CPU usage of the algorithm steps on a cluster of 8-core nodes. The histograms show the percentage of time spent in the three main routines using different parallel approaches.

solution of the Poisson and Helmholtz equations are solved directly using a Cholesky factorisation in the first case, or via **Algorithm 6** in the second case. For both simulations the FFT Transposition approach is applied and the extra time required to solve the system iteratively is purely due to the slow convergence of the PCG method, especially for the Poisson equation. A moderate reduction of the number of iterations can be attained via the introduction of more suitable preconditioners or constraining the PCGM residual tolerance. However, it is unlikely to obtain the same performance as a direct solution. In addition to previous considerations, we note that the advection term calculation is the one

generally dominating the FFT Transposition approach. This result is a consequence of the communication appearing during the FFT parallel algorithm. On the other hand Fig. 4.13(c) highlights how communication plays a relevant role during the linear system solution in the case that we are decomposing the mesh. In fact at each iteration of **Algorithm 6** a set of mesages must be exchanged between mesh partitions. The direct consequence is a complete dominance of the linear system solution on the overall computational time. We report also some hybrid combinations of the two parallel approaches. As can be noted in Fig. 4.13(d) those simulations are dominated by the advection calculation.

In Fig. 4.14 we present the speed-up S of different simulations associated with the standalone approaches and the hybrid simulations, alomg with the ideal linear speed-up (red solid line). While the FFT Transposition approach is scaling linearly up to its theoretical bottleneck, we note that, as expected, the Mesh Decomposition scalability is reduced, limiting the number of elements per process to four, and not less.

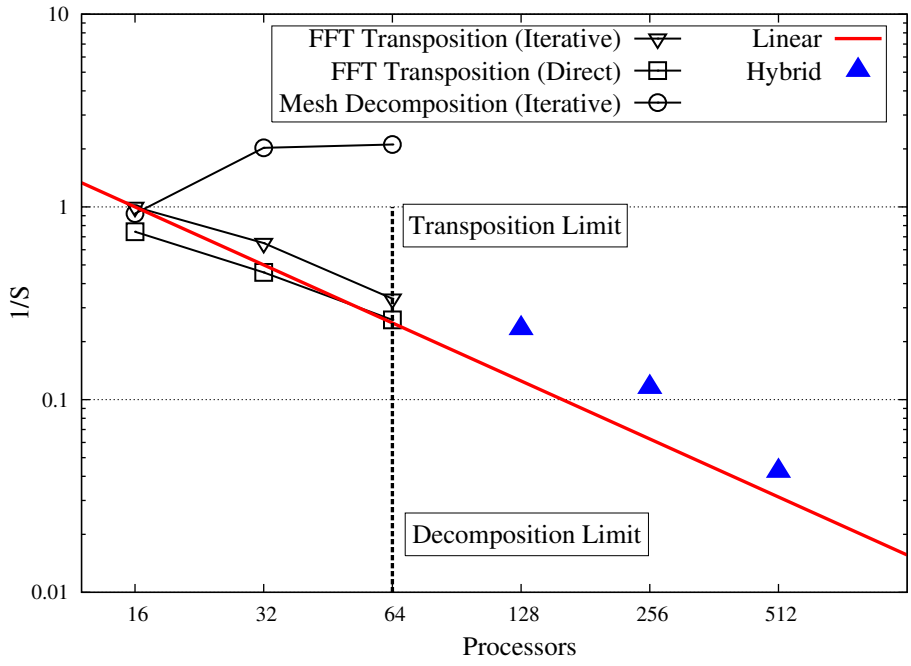


Figure 4.14: Turbulent pipe flow parallel simulation - scaling features on a cluster of 8-core nodes. The red solid line indicates the theoretical linear speed-up based on the 16-core (2 nodes) run using the FFT Transposition (iterative) approach. The Transposition and Decomposition bottlenecks are marked with a vertical black dashed line.

Comparing the FFT Transposition variants, $i.e.$ (Iterative) and (Direct), we also can

appreciate that the direct approach is more efficient. Some hybrid combinations of the two parallel approaches are marked in Fig. 4.14 with blue triangles, showing that the classical scalability limits can be overcome with the help of a flexible implementation. Although the hybrid approaches scale well, the speed-up does not perfectly match the ideal one, but it is slightly sub-linear. Another interesting remark concerns the performance comparison between hybrid approaches and the FFT Transposition (Direct). In fact it can be observed that the 128-core hybrid simulation is characterised by a speed-up very similar to the 64-core FFT Transposition (Direct) case. This result, even if code dependent, suggests that selecting the most appropriate numerical approach can improve performance and reduce the overall computational time. Efficiency of numerical simulations from an energy performnace point of view is becoming increasingly important. Optimal approaches should minimise both the computational time and the energy consumption, therefore getting the same performance using half the number of processors is the most efficient choice.

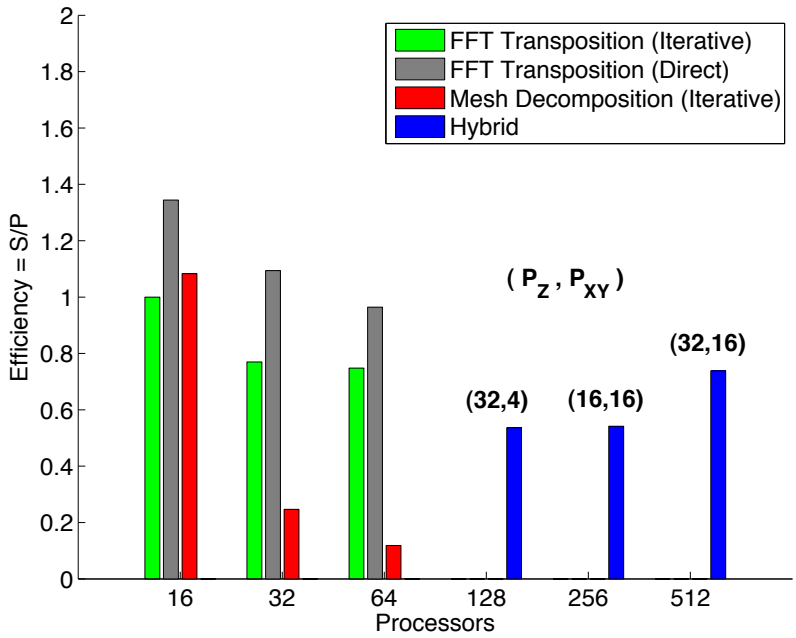


Figure 4.15: Turbulent pipe flow parallel simulation - efficiency of parallelisation approaches on a cluster of 8-core nodes. The histograms show the efficiency E of different parallel simulations defined as $E = S/P$ where S is the speed-up and P is the total number of processors used for the simulation. The speed-up is based on the 16-core (2 nodes) run using the FFT Transposition (iterative) approach.

A final quantification of the investigated techniques is given in Fig. 4.15, where the parallel efficiency $E = S/\boldsymbol{P}_{TOT}$ is illustrated. Again the reference run is the 16-core FFT Transposition (Iterative) simulation, hence $E = 1$ in that case. The efficiency bars depicted in this figure reinforce the basic considerations we derived previously, and that we can summarise as:

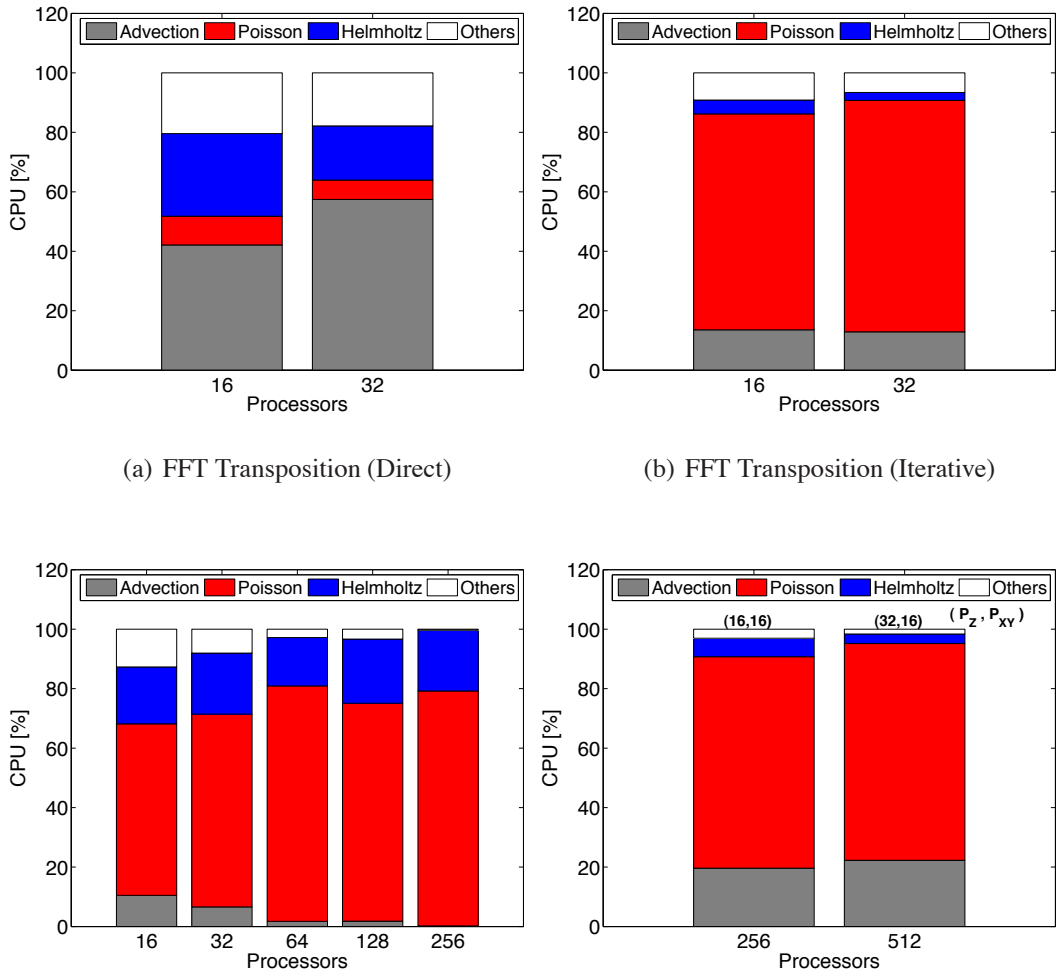
- when using an FFT Transposition approach a direct solution of the arising linear systems is the most effective choice;
- hybrid approaches recover efficiency permitting scalability beyond the theoretical bottleneck;
- when a problem is Fourier-dominated (many plane but few elements per plane), the FFT Transposition approach is the most efficient choice.

Finally, we remark on a point which can be noticed both in Fig. 4.14 and Fig. 4.15, $i.e.$ within its scalability limits, a Mesh Decomposition (Iterative) technique is more efficient than its FFT Transposition counterpart (Iterative). This consideration is intuitively valid when most of the communications between partitions are inside the same node, therefore with $\tau_L \sim 0$. We observed the same behaviour in some preliminary tests we have not reported here, when we tested the parallelisation approaches on a singe node. The Mesh Decomposition approach is generally more efficient on shared memory machine, where the latency is very low and sending a lot of small messages becomes the most attractive method. Moreover recent MPI libraries take advantage of the physical memory layout and, given that the memory is shared between processors, do not physically send messages, but point processes to the right locations in memory.

4.6.2 Turbulent Channel

As we have just considered for the turbulent pipe, in Fig. 4.16 we present the CPU time percentage spent within the different routines of the incompressible Navier-Stokes solver. Compared to the pipe simulations the turbulent channel flow discretisation is dominated by the number of elements. Consequently, most of the time is spent in solving the linear systems, apart from the FFT Transposition (Direct) approach. We note the

solution of the Poisson equation when using an iterative solver is the dominant routine, due to the high number of iterations required. The hybrid parallel simulations show in this case a dominance of the elliptic solver over the advection routine, as a consequence of the elevated number of DOFs in the $xy-$plane. Although we are using roughly the same number of $\boldsymbol{P}_Z$ processors as we applied for the pipe hybrid approaches, in this case the number of points in $z-$direction are reduced, bounding the communication overhead associated with the FFT transposition routine.



(a) FFT Transposition (Direct)

(b) FFT Transposition (Iterative)

(c) Mesh Decomposition (Iterative)

(d) Hybrid

Figure 4.16: Turbulent channel flow parallel simulation - CPU usage of the algorithm steps on a cluster of 8-core nodes. The histograms show the percentage of time spent in the three main routines using different parallel approaches.

Fig. 4.17 shows the scalability features we observed while testing the channel flow

case. Within the FFT Transposition bottleneck, significantly reduced in this case, we note that the FFT Transposition (Direct) method performs better than the other two approaches, confirming that an iterative method without a suitable preconditioner may become critical as we increase the linear system size. In addition we observe that the Mesh Decomposition (Iterative) approach is performing better than the FFT Transposition (Iterative), confirming the considerations we have reported in the final part of the previous section.

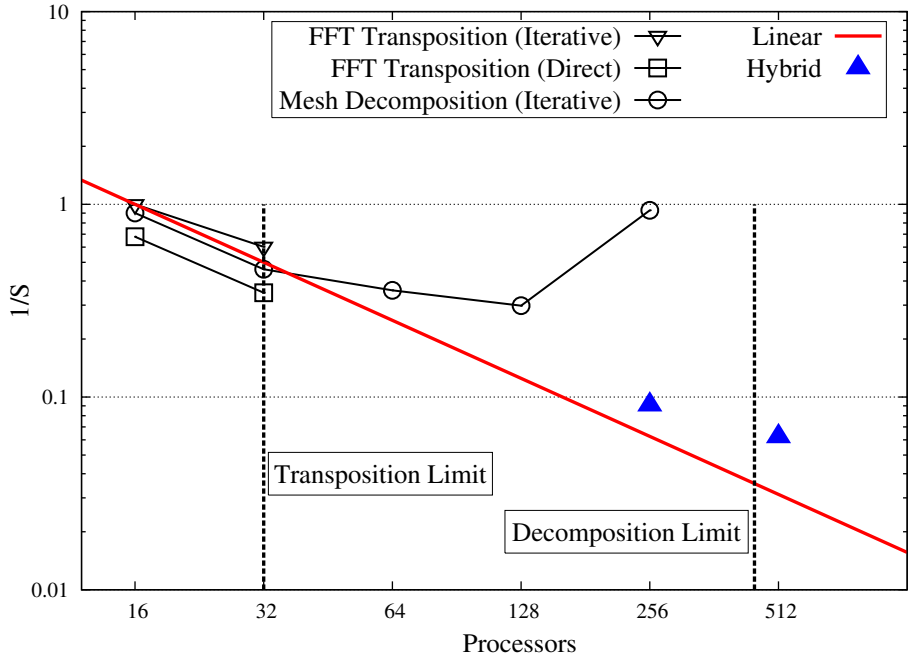


Figure 4.17: Turbulent channel flow parallel simulation - scaling features on a cluster of 8-core nodes. The red solid line indicates the theoretical linear speed-up based on the 16-core (2 nodes) run using the FFT Transposition (iterative) approach. The Transposition and Decomposition bottlenecks are marked with a vertical black dashed line.

As before, the practical bottleneck of the Mesh Decomposition technique is reducing the overall efficiency of the approach, which shows a sub-linear scaling. However, if we try to extrapolate visually the possible ideal speed-up of the Mesh Decomposition approach up to 64 processors, we can conclude that a direct solution of the linear system could provide the same performance using fewer processors. Therefore, even in this case, the choice which optimises CPU-time and energy consumption is the 32 processor approach using the FFT Transposition (Direct) method. The hybrid approaches can be used to extend the scalability limits for this problem too, or used to recover scalability as for

the 256 processors run, where the Mesh Decomposition approach stops scaling. All the observations we just made are summarised in the efficiency plot in Fig. 4.18.

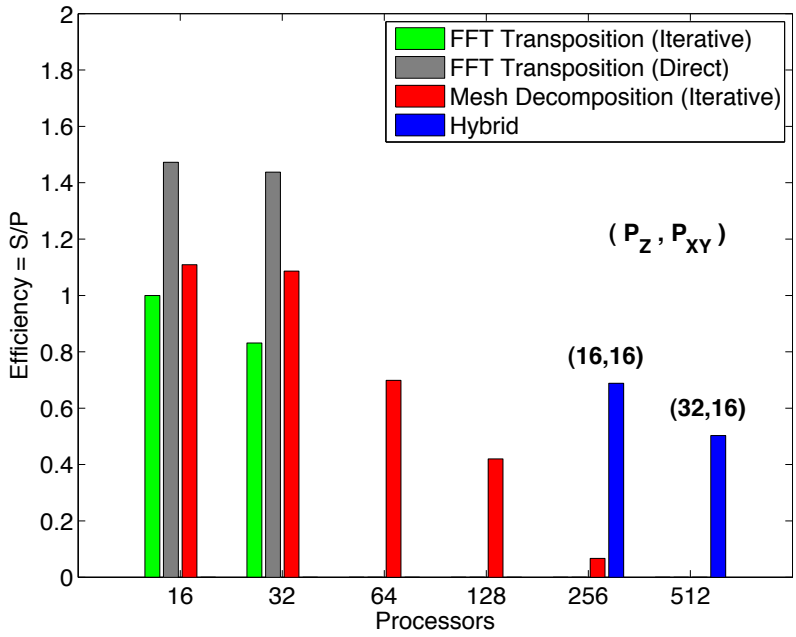


Figure 4.18: Turbulent channel flow parallel simulation - efficiency of parallelisation approaches on a cluster of 8-core nodes. The histograms show the efficiency E of different parallel simulations defined as $E = S/P$ where S is the speed-up and P is the total number of processors used for the simulation. The speed-up is based on the 16-core (2 nodes) run using the FFT Transposition (iterative) approach.

4.7 Discussion

In this chapter we presented a methodology to parallelise a 3D incompressible Navier-Stokes algorithm based on discretisation flexibility. Despite the results and guidelines deriving from this study depending on the *Nektar++* specific implementation, we can identify some general behaviours and trends. A priori choices in the decision-making process while developing a CFD software may therefore benefit from the systematic investigations we have reported.

Initially we described how, given a Fourier spectral/*hp* element method, we can approach the task of parallelisation. Careful consideration of the implied numerical meth-

ods led to the recognition of two canonical parallelisation possibilities. One discretisation component allows an elemental decomposition of the domain and the other a modal decoupling of the Fourier modes. Although experience and intuition can generally suggest which approach is more suitable for a specific scenario, in general mesh problems this task becomes difficult, especially when moving from one machine to another or when we have to tackle, with the same algorithm, a range of different physical problems. Even from a purely theoretical perspective, it is straightforward to understand how a standalone parallel approach can not be optimal in all situations.

Acknowledging the factors which affect parallel efficiency is a fundamental step to understanding the optimal choices. In the context of a Fourier spectral/*hp* element method, we highlighted how the ratio between the DOFs in the $xy-$plane and the number of modes in the periodic dimension requires special attention. In fact we recognised that problems with a higher number of modes compared to the number of elements per plane appears to benefit from the FFT Transposition approach. On the other hand, a domain discretisation which is element-dominated generally prefers a Mesh Decomposition technique. Keeping in mind this overall guideline, we also need to recall the role played by actual vector sizes moved to and processed by different CPUs. Fitting the local cache and optimising the effect of latency and bandwidth on the communication pattern is essential. Parallelisation approaches requiring an increased number of messages, such as the Mesh Decomposition, can suffer from performance reductions if the latency is high. On the contrary, we observed that data locality removes this limitation making such type of parallel techniques optimal on shared memory machines.

While an *ad hoc* algorithm design can address some of the efficiency issues, an unpredictable role is played by the employed libraries. Recent MPI and FFTW versions are characterised by a certain level of optimisation, as a consequence of their high portability and popularity. For example, MPI applies message decomposition to maximise the bandwidth usage. Therefore, selection of the most appropriate libraries has been identified as one of the key aspects for an efficient algorithm design.

Once we have acknowledged the principal quantities which impact the efficiency and portability of the two parallel algorithms considered, we presented a hybrid parallel solution. We illustrated an implementation procedure where, encapsulating the concept of

parallelisation, we are able to introduce one or more parallel technique concurrently. By mixing the efficiency properties of the two parallel algorithms a series of benefits have been identified and they are:

1. a flexible implementation, which allows an easy switch between parallel techniques, provides users with direct and accessible tools to tune the parallel efficiency of their simulations;

2. hybrid parallel solutions extend the strong scalability limits, promoting the usage of larger machines without modifying the code.

In turn we demonstrated that a substantial effort during the implementation process to increase algorithm flexibility is generally beneficial. Moreover, by implementing both the commonly used parallel approaches, we removed the uncertainties appearing when deciding which approach to implement. As we stressed before, the DOFs in $xy-$plane and the number of Fourier modes are first indicators of which technique is more appropriate. Generally speaking we want to tackle the solution of CFD problems where those quantities can vary, reaching also extreme values. Without the possibility of quantifying in advance those variables, the implementation framework we presented is a possible way to address the drawback.

In monitoring computational times for different scenarios we realised that hybrid parallel solutions should mainly be used to extend scalability limits. In fact, within the bottlenecks of the two parallel techniques we investigated, the optimal parallel approach was always the FFT Transposition or the Mesh Decomposition. Combining parallel approaches in this case returns performances which can be considered as an average between the two limiting techniques. However we can not totally exclude that a hybrid parallel solution may be the optimal approach in some peculiar scenarios (problem size, machine, libraries, etc.). Although we do not have numerical evidence supporting efficiency portability across architectures, we can speculate that in general relative optimums can be found in each scenario, or at least that we can preserve a certain level of efficiency for our parallel simulations also on different machines. In addition, the flexibility of choosing between different techniques allows for the possibility of compensating for a reduced efficiency in the implementation. An example of this last remark is the poor scaling of the

Mesh Decomposition approach reported in Fig. 4.17. As can be seen, the 128-processor run stops scaling before the theoretical bottleneck (which is one element per processor). That is because of an absence of computation/communication overlap and possible inefficiencies associated with external libraries. However, we observe that, although the current implementation may require refinements and improvements, a hybrid approach is able to recover efficiency, allowing effective parallel simulations even in the development phase.

The extension of the strong scalability limits we achieved throughout the hybrid parallel implementation proposes a further consideration concerning CFD simulations. As we stated in the initial part of this chapter, a common philosophy is to exploit a new machines potentials by investigating larger problems (and/or larger Reynolds numbers), hence capitalising on weak scalability. Our view is that good weak scalability follows from good strong scalability. Therefore when we force an algorithm to extend its strong scalability limits we secure the chance of running our simulations faster on larger machines (strong scalability) and at the same time we maintain the progression ratio between problems size and machines size (weak scalability).

Assuming the implementation reaches an optimal level, where all implementation inefficiencies are removed, predicting performance as a function of the machine and problem features is feasible. As a standard approach when investigating parallel efficiency we introduced a scalability model, specifying all the details of its construction. We stressed that modelling requires assumptions when building relations between machine, algorithm and problem size and that these assumptions introduce errors which can be misleading. The model we proposed furnishes sensible guidelines on what is the best approach when parallelising a specific problem. The model does not take into consideration the physical layout of the computer (interconnect, processors per node, etc.), therefore it must be considered indicative and not quantitative. A specialised scalability model could be coupled with a pre-run optimisation routine, which queries the machine and analyses the problem details to work out directly the optimal combination of strategies to reduce computational time and energy consumption.

Application on fluid dynamics have been considered in this chapter, namely a turbulent pipe and turbulent channel case. In both cases we demonstrated that a hybrid parallel approach can be utilised to extend the bottlenecks of standalone parallel techniques. We

want to emphasise that not all the hybrid parallel combinations we tried actually showed good scalability properties. Depending on the grouping of elements and Fourier modes across processors, some combinations may not perform efficiently. However, using those test cases as a benchmark, we identified a systematic methodology to investigate and achieve an efficient parallelisation. Furthermore, we showed that suitable numerical techniques may result in reducing the computational time without increasing the number of processors. This is the case of the FFT Transposition (Direct) approach, which generally performs as well as a Mesh Decomposition method with twice the number of CPUs. Minimising the energy consumption when running a simulation is a point of interest in current research about high performance computing. Again, implementation flexibility plays a relevant role in addressing these goals.

Chapter 5

Conclusions

As anticipated in the introduction chapter, there are many parameters which influence the efficiency of a CFD simulation. The research presented in this thesis was philosophically driven by our belief in the existence of potential benefits coming from implementation flexibility. We demonstrated that effective improvements of CFD algorithms can be addressed at various levels; from a conscious selection of the numerical methods involved in the problem approximation, to the implementation of *ad hoc* computational strategies. This thesis, in particular, considered investigations of optimal approaches when explicitly time-stepping partial differential equations and efficient parallelisation methodologies for CFD simulations.

In next section we will provide a detailed summary of the work reported in this thesis. However, we would like to recall briefly here our main findings:

- Computational efficiency investigations when time-stepping a basic PDE using explicit time-stepping schemes and the spectral/*hp* element method have shown that:

    1. A spectral/*hp* element method with an intermediate polynomial order ($4 \leq P \leq 8$) is generally the most efficient computation choice to get a desired accuracy on the solution.
    2. Short-time integration can be performed more efficiently using a low-order multi-step scheme.
    3. High-order multi-stage schemes are generally useful to preserve accuracy on long-time integrations.

4. When we have small elements in the mesh it is generally more efficient to improve accuracy refining the mesh size (h-refinement).

- Computational efficiency investigations on the parallelisation front have shown that:

1. The modal and the elemental decomposition approaches can be efficiently coupled, for a Fourier spectral/*hp* element method, using MPI virtual topologies.
2. A flexible implementation can be very useful to tune our software to the most convenient approach depending on the problem of interest.
3. Sensible combinations of the two parallel approaches exist and they can be used to extend the strong scalability limits and to recover parallel efficiency.
4. A proper selection of the algorithms involved in the problem solution, $e.g.$ iterative vs direct linear system solvers, can promote efficiency from an energy consumption perspective.

In this thesis we tried to improve the level of understanding on a relevant topic such as the computational efficiency for the solution of fluid dynamics problems. Although we mainly focused on the spectral/*hp* element methods, some of our considerations and practical recipes can be useful to many CFD practitioners, regardless the type of spatial discretisation they are adopting. In the last few years researchers oriented their investigations to parallel computational efficiency. Some examples of tailored numerical methods to improve parallel efficiency are still available in literature (Kim & Sandberg 2012). However, the most relevant improvements derive from the combination of the message passing model and the shared memory model (Lusk & Chan 2008), in order to exploit the features of new super-computers. In fact, new architectures are characterised by many multi-core nodes that need to communicate, but a good amount of shared memory is available on each node. While in 2008 this approach was still in a development stage, nowadays is becoming more common and efficient also for CFD applications (Mininni et al. 2011, Hoefler et al. 2013, Friedley et al. 2013).

5.1 Summary

In Chapter 2 we illustrated the overall framework, in terms of numerical methods, which has been used to practically conduct the investigations of efficiency. We started by presenting the spatial discretisation techniques. Focusing on high-order methods, we showed the capability of such approximations to reach high levels of accuracy while recalling their widespread use by CFD practitioners over the last few decades (Karniadakis & Sherwin 2005). The approach presented by $Karniadakis$ (Karniadakis 1990) in the 1990's has been introduced in our implementation, combining the spectral/*hp* element method with a pseudospectral method to create a 3D discretisation. We stressed the advantages of this discretisation methodology, which provides a modal decoupling of the approximation, reducing a 3D problem into a series of 2D discretisations, promoting efficiency and reducing memory usage. An encapsulation of this technique in C++ classes has also been illustrated. In this context we highlighted how code reutilisation deriving from object-oriented programming can be very beneficial in facilitating numerical methods implementation.

Having described the approach we followed in spatially discretising a 3D domain, we introduced a unified technique to time-step partial differential equations. Based on $Butcher$'s General Linear Method (GLM) (Butcher 2006), the framework we implemented allows a universal treatment of different time-stepping schemes which allows one to easily select from a wide variety of time-stepping schemes. We also described in more detail some specific aspects of this implementation, the ones directly connected to the our final application, $i.e.$ incompressible flow simulations. Namely we extended our discussions to the encapsulation of implicit-explicit (IMEX) time-stepping schemes into a GLM prototype and to the practical enforcement of strongly-imposed time-dependent boundary conditions. This translated to a decoupling of the scheme-related coefficients/methods and we showed how it translates in a GLM formulation. The latter issue has been solved demonstrating that by applying a similar GLM approach to the Dirichlet set of degrees of freedom, we can eliminate the dependence of the time-derivative of the boundary conditions from the general formulation.

Finally, in chapter 2, we presented the approach taken to solve the 3D incompressible Navier-Stokes equations. Taking advantage of the spatial and temporal discretisation

abilities in *Nektar++* , we introduced a splitting scheme algorithm for the solution of 2D and 3D incompressible flows (Karniadakis et al. 1991). We also reported and validated some canonical test cases, in particular the turbulent flow in a pipe and a channel.

In Chapter 3 we focused on the computational efficiency of low and high-order methods when implied in the solution of an unsteady problems. Given the complexity of the issues we limited ourself to a simplified test case, namely a 2D unsteady linear advection equation. Although simple, this can be seen as a reduced model for many applications in CFD for both compressible and incompressible flows. Indeed, the explicit integrated part of the velocity correction scheme corresponds to the solution of a non-linear unsteady advection equation. In addition, when a sub-stepping algorithm is introduced during the splitting scheme, a discontinuous Galerkin projection is applied also for incompressible flows.

Acknowledging the relevance of the selected test case, we investigated the efficiency of *Nektar++* when solving this type of equation. As a practical example we employed a Gaussian function rotating around the centre of a square domain convected by a time-independent, but spatially dependent, advective field. We assumed the CPU-time required to integrate the equation as an indicator of the global algorithm efficiency, and fixed the level of accuracy desired on the final solution. Recalling that the accuracy in numerical terms translates into the error deriving from the spatial and temporal discretisation, we performed a series of parametric simulations systematically varying the parameters which affect accuracy, $i.e.$ the polynomial expansion order, the mesh, the time-stepping scheme and the final time. Furthermore, a relevant role is played by numerical stability constraints deriving from the well-known CFL condition. Although not directly, restrictions on the applicable time-step influence accuracy throughout the temporal accumulation error, which is a monotonically increasing function of the number of required steps and therefore of the final time. Associated with each combination of those parameters we can recognise a definite numerical error on the solution. The final goal of this study was to understand the optimal parameters combinations to achieve a desired level of accuracy on the final solution for a selected final time, minimising the CPU-time.

The main conclusion of this study contradicted the general understanding regarding the utilisation of high-order methods to spatially discretise time-dependent problems. The

common idea is that high-order methods tend to limit practical efficiency because CFL restrictions grow algebraically with the polynomial expansion order. We demonstrated that, despite being true that the eigenvalues dictating stability grow algebraically with P, the accuracy of the final solution improves even faster (in an exponential manner for smooth problems). Therefore, given a desired level of accuracy, high-order methods tend to be, computationally speaking, the faster route to reach the required results. Although this last remark was clearly observable in the presented graphs for the uniform mesh family, we could not demonstrate the same conclusion for non-uniform meshes. For those set of meshes we recognised that an h refinement was more beneficial within our parameters range, promoting the utilisation of low order methods. However, observing the results obtained in those simulations, we can speculate that possible absolute optimums may be found for polynomial orders higher than the ones we considered. Another general remark is about the efficiency of time-stepping schemes. In this context we noted that for short-time integrations, low order time-stepping schemes (AB2, RK2) turn out to be computationally faster than the widely used high-order RK4. However, in case high accuracy is required ($\sim 10^{-9}$), higher-order time-stepping schemes appeared to be the only possible approach.

In Chapter 4 we introduced a methodic approach, based again on implementation flexibility, to undertake the parallelisation of our CFD algorithm. We initially recalled the relevance of parallel computing for CFD applications, reporting some of the parallel solutions adopted by various authors. Recognising the fundamental role of algorithms portability across architectures we stressed the importance of implementation strategies as a tool to optimise code efficiency. While super-computers continue to improve their capabilities, CFD practitioners need to keep their algorithms up-to-date in order to utilise these resources efficiently. Consequently, the ability to easily switch to the most appropriate approach, as a function of the problem nature and the hardware features, dictates the need to tune our solution methodology to the most convenient one for each specific scenario. Following these considerations, we presented an hybrid parallelisation approach to parallelise the velocity correction scheme algorithm. We took advantage of the natural predisposition of the Fourier spectral/*hp* element method to be parallelised in different

ways. The common procedure when parallelising this type of discretisation is to apply either an elemental decomposition of the domain or a modal decomposition of the harmonic expansion. Using the turbulent flows described in Chapter 2 as test cases, we combined these two approaches. This hybridisation provides, together with the opportunity to select the optimal method, the option to apply the parallel approaches concurrently to extend strong scalability limits. Although weak scalability remains a useful aspect for CFD applications, we believe that any effort to amplify strong scalability naturally promotes an efficient execution of our algorithms as the hardware performance increases. Moreover, achieving weak scalability is not always of practical interest, especially when we simply desire to run our simulations faster on a bigger machine, without enlarging the problem size.

Parallel speed-up and efficiency observations gave rise to a series of considerations and conclusions. First we demonstrated that the combination of parallel approaches can be directly used to extend the strong scalability limits or to recover efficiency when the scenario, or the implementation itself, is not optimal. In addition we noted that hybrid approaches were not beneficial within the scalability limits of the standalone techniques, at least for the cases we investigated. Therefore they should be used just to enforce strong scalability. The CFD problems we investigated were characterised by different discretisation features. In particular we observed that the parallel speed-up of spectral dominated problems (pipe flow) is promoted by a modal decomposition (FFT transposition). On the other hand, when we have a lot of elements, a mesh decomposition approach is preferable. However, we recognised that low latency machines ($e.g.$ shared memory environment such as a single node) can take advantages of mesh decomposition technique in any case. The reason is that the communication overhead deriving from the latency of many small messages is almost eliminated. Comparing the speed-up of an iterative method and a direct approach when solving a linear system suggested a further consideration. In fact we noted that a sensible usage of a direct method can yield the same speed-up levels with fewer cores. In conclusion, implementation flexibility can be also used to optimise our simulations from an energy consumption point of view.

5.2 Final remarks

Throughout a series of practical applications we showed how implementation flexibility can be used to tune our simulations parameters to achieve a greater level of efficiency. Despite the fact that results were obtained using a single code, *Nektar++* , we can state that in general, variety in the selectable computational strategies helps to fit a wide range of needs. Although the implementation effort may appear prohibitive, we demonstrated it yields appreciable benefits. While seeking for greater performance, we gained a deeper understanding about some problems related to CFD algorithms, such as the efficiency of high-order methods when utilised in an explicit time integration.

As often happens in research, the presented work, while producing some understandings, inspired other possible investigations. On the time-integration front we find it could be of interest a comparison between the presented explicit time-integration schemes and their implicit counterparts. This would provide a deeper understanding of how efficiency is affected by CFL restrictions compared to the higher operation count per time-step typical of implicit methods.

The parallelisation study naturally suggests some further numerical experiments and implementation extensions. First we find the implementation of a parallelisation optimiser of practical interest. Indeed, we intend to refine and consequently use the scalability model presented in this thesis to realise a top level algorithm, which, after collecting the required parameters from the machine and the problem, can automatically detect the most efficient parallel approach in a given situation. When a mesh decomposition was applied, we took into account diagonally preconditioned iterative methods only for solving the arising linear systems. In this context we are planning to extend our studies to encompass other preconditioners and also direct parallel methods for the solution of linear systems. Moreover, it could be of interest to explore the benefits coming from a multilevel static condensation approach. Hybrid parallelisation paradigms, $i.e.$ multithreading and GPU usage combined with MPI, gained great popularity in the last five years, especially since the development of new hybrid hardware architectures, which utilise both CPUs and GPUs. The next step in terms of implementation improvements would be to make our software able to take advantage of this new technology and investigate the real

benefits in a variety of parameters settings.

One of the consequences of this thesis is that with the development of the many numerical techniques, *Nektar++* has been extended to enable the study of a wide range of practical CFD problems. Indeed, we plan to use the developed tools to investigate the vortex shedding behind a vibrating cylinder, in order explore the wake topology.

Bibliography

Ainsworth, M. (2004*a*), 'Discrete dispersion relation for hp-version finite element approximation at high wave number', *SIAM J. Numer. Anal.* **42**(2), 553–575.

Ainsworth, M. (2004*b*), 'Dispersive and dissipative behaviour of high order discontinuous Galerkin finite element methods', *J. Comput. Phys.* **198**(1), 106–130.

Ainsworth, M., Monk, P. & Muniz, W. (2006), 'Dispersive and dissipative properties of discontinuous Galerkin finite element methods for the second-order wave equation', *Journal of Scientific Computing* **27**(1), 5–40.

Alastruey, J., Khir, A., Matthys, K., Segers, P., Sherwin, S., Verdonck, P., Parker, K. & Peiro, J. (2011), 'Pulse wave propagation in a model human arterial network: Assessment of 1-D visco-elastic simulations against in vitro measurements', *Journal of Biomechanics* **44**, 2250–2258.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. & Sorensen, D. (1999), *LAPACK Users' Guide*, third edn, Society for Industrial and Applied Mathematics, Philadelphia, PA.

Antonietti, P., Mazzieri, I., Quarteroni, A. & Rapetti, F. (2012), 'Non-conforming high order approximations of the elastodynamics equation', *Computer Methods in Applied Mechanics and Engineering* **209–212**, 212–238.

Ascher, U., Ruuth, S. & Wetton, B. (1995), 'Implicit-Explicit Methods for Time-Dependent Partial Differential Equations', *SIAM Journal on Numerical Analysis* **32**(3), 797–823.

Barkley, D., Blackburn, H. & Sherwin, S. (2007), 'Direct optimal growth analysis for timesteppers', *International Journal for Numerical Methods in Fluids* **231**, 1–20.

Blackburn, H. & Sherwin, S. (2004), 'Formulation of a Galerkin spectral element–Fourier method for three-dimensional incompressible flows in cylindrical geometries', *J. Comput. Phys.* **197**(2), 759–778.

Blaisdell, G., Spyropoulos, E. & Qin, J. (1996), 'The effect of the formulation of nonlinear terms on aliasing errors in spectral methods', *Applied Numerical Mathematics* **21**(3), 207–219.

Bolis, A., Cantwell, C., Kirby, R. & Sherwin, S. (2013), 'From h to p efficiently: Optimal implementation strategies for explicit time-dependent problems using the spectral/hp element method', *Submitted to Internation Journal for Numerical Methods in Fluids* .

Bowman, J. & Roberts, M. (2011), 'Efficient dealiased convolutions without padding', *SIAM J. Sci. Comput.* **33**(1), 386–406.

Boyd, J. (2001), *Chebyshev and Fourier spectral methods*, second edn, Dover Publications, Inc.

Butcher, J. (1987), *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*, Wiley, Chichester.

Butcher, J. (2006), 'General linear methods', *Acta Numerica* **15**, 157–256.

Butcher, J. (2009), 'General linear methods for ordinary differential equations', *Mathematics and Computers in Simulation* **79**(6), 1834–1845.

Canstonguay, P., Williams, D., Vincent, P., Lopez, M. & Jameson, A. (2011), On the Development of a High-Order, Multi-GPU Enabled, Compressible Viscous Flow Solver for Mixed Unstructured Grids, *in* '20th AIAA Computational Fluid Dynamics Conference', number AIAA-2011-3229, Honolulu, Hawaii, USA.

Cantwell, C., Sherwin, S., Kirby, R. & Kelly, P. (2011*a*), 'From h to p efficiently: selecting the optimal spectral/hp discretisation in three dimensions', *Mathematical Modelling of Natural Phenomena* **6**(3), 84–96.

Cantwell, C., Sherwin, S., Kirby, R. & Kelly, P. (2011*b*), 'From h to p efficiently: Strategy selection for operator evaluation on hexahedral and tetrahedral elements', *Computers & Fluids* **43**(1), 23–28.

Canuto, C., Hussaini, M., Quarteroni, A. & Zang, T. (2006), *Spectral Methods: Fundamental in Single Domain*, Scientific Computing, Springer, New York.

Canuto, C., Hussaini, M., Quarteroni, A. & Zang, T. (2007), *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics*, Scientific Computing, Springer, New York.

Carmo, B., Sherwin, S., Bearman, P. & Willden, R. (2011), 'Flow-induced vibration of a circular cylinder subjected to wake interference at low Reynolds number', *Journal of Fluids and Structures* **27**, 503–522.

Chan, A., Balaji, P., Gropp, W. & Thakur, R. (2008), 'Communication analysis of parallel 3D FFT for flat cartesian meshes on large Blue Gene systems', *High Performance Computing-HiPC 2008* **5374**, 350–364.

Chapman, B., Jost, G. & Pas, R. v. d. (2007), *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*, The MIT Press.

Cockburn, B. & Shu, C. (1998), 'The Local Discontinuous Galerkin Method for Time-Dependent Convection-Diffusion Systems', *SIAM J. Numer. Anal.* **35**(6), 2440–2463.

Cockburn, B. & Shu, C. (2001), 'Runge–Kutta discontinuous Galerkin methods for convection-dominated problems', *Journal of Scientific Computing* **16**(3), 173–261.

Collatz, L. (1966), *The Numerical Treatment of Differential Equations*, Springer Verlag, New York.

Crawford, C., Evangelinos, C., Newman, D. & Karniadakis, G. (1996), 'Parallel benchmarks of turbulence in complex geometries', *Computers & Fluids* **25**(7), 677–698.

De Basabe, J. & Sen, M. (2010), 'Stability of the high-order finite elements for acoustic or elastic wave propagation with high-order time stepping', *Geophys. J. Int.* **181**, 577–590.

De Basabe, J., Sen, M. & Wheeler, M. (2008), 'The interior penalty discontinuous Galerkin method for elastic wave propagation: grid dispersion', *Geophys. J. Int.* **175**, 83–93.

Demmel, J., Heat, M. & van der Vorst, H. (1993), 'Parallel numerical linear algebra', *Acta Numerica* **2**, 111–197.

den Toonder, J. & Nieuwstadt, F. (1997), 'Reynolds number effects in a turbulent pipe flow for low to moderate Re', *Physics of Fluids* **9**(11), 3398–3409.

Dumbser, M., Käser, M. & Toro, E. (2007), 'An arbitrary high-order Discontinuous Galerkin method for elastic waves on unstructured meshes – V. Local time stepping and p-adaptivity', *Geophys. J. Int.* **171**(2), 695–717.

Eskilsson, C. (2005), Spectral/hp Discontinuous Galerkin Methods for Computational Hydraulics, PhD thesis, Chalmers University of Technology.

Evangelinos, C. & Karniadakis, G. (1996), Communication Performance Models in Prism: A Spectral Element-Fourier Parallel Navier-Stokes Solver, *in* 'Proceedings of the 1996 ACM/IEEE Conference on Supercomputing (SC'96)'.

Feitelson, D. (1999), 'On the interpretation of top500 data', *International Journal of High Performance Computing Applications* **13**(2), 146–153.

Fischer, P. (1990), 'Analysis and Application of a Parallel Spectral Element Method for the Solution of the Navier-Stokes Equations', *Computer Methods in Applied Mechanics and Engineering* **80**, 483–491.

Fischer, P. (1994), 'Parallel domain decomposition for incompressible fluid dynamics', *Contemporary Mathematics* **157 AMS**, 313–322.

Fischer, P. (1997), 'An overlapping Schwarz method for spectral element solution of the incompressible Navier-Stokes equations', *J. Comput. Phys.* **133**, 84–101.

Fischer, P., Lottes, J., Pointer, D. & Siegel, A. (2008), 'Petascale algorithms for reactor hydrodynamics', *Journal of Physics: Conference Series* **125**(1), 012076.

Fischer, P. & Patera, A. (1994), 'Parallel Simulation of Viscous Incompressible Flows', *Ann. Rev. Fluid Mech.* **26**, 483–528.

Fischer, P. & Rønquist, E. (1994), 'Spectral element methods for large scale parallel Navier-Stokes calculations', *Computer Methods in Applied Mechanics and Engineering* **116**(1–4), 69–76.

Fornberg, B. (1996), *A Practical Guide to Pseudospectral Methods*, Cambridge University Press.

Friedley, A., Bronevetsky, G., Lumsdaine, A. & Hoefler, T. (2013), Hybrid MPI: Efficient Message Passing for Multi-core Systems, *in* 'accepted at IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC13)', Denver, Colorado, USA.

Frigo, M. & Johnson, S. (2005), 'The Design and Implementation of FFTW3', *Proceedings of the IEEE* **93**(2), 216–231.

Gabriel, E., Fagg, G., Bosilca, G., Angskun, T., Dongarra, J., Squyres, J., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R., Daniel, D., Graham, R. & Woodall, T. (2004), 'Open MPI: Goals, concept, and design of a next generation MPI implementation', *Recent Advances in Parallel Virtual Machine and Message Passing Interface* **3421**, 97–104.

Gottlieb, D. & Orszag, S. (1977), *Numerical analysis of spectral methods: theory and applications*, CBMS-NSF, Society for Industrial and Applied Mathematics, Philadelphia.

Gottlieb, S., Shu, C. & Tadmor, E. (2001), 'Strong stability-preserving high-order time discretization methods', *SIAM Review* **43**(1), 89–112.

Grama, A., Gupta, A. & Kumar, V. (1993), 'Isoefficiency: measuring the scalability of parallel algorithms and architectures', *IEEE Parallel Distributed Technology Systems Applications* **1**(3), 12–21.

Guermond, J. & Minev, J. (2006), 'An overview of projection methods for incompressible

flows', *Computer Methods in Applied Mechanics and Engineering* **195**(44–47), 6011–6045.

Gupta, A. & Kumar, V. (1993), 'The scalability of FFT on parallel computers', *IEEE Transactions on Parallel and Distributed Systems* **4**(8), 1–27.

Hamman, C., Kirby, R. & Berzin, M. (2007), 'Parallelization and scalability of a spectral element channel flow solver for incompressible Navier–Stokes equations', *Concurrency and Computation: Practice and Experience* **19**(10), 1403–1422.

Hesthaven, J. & Warburton, T. (2008), *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*, Springer Texts in Applied Mathematics 54, Springer Verlag, New York.

Hirsch, C. (2007), *Numerical computation of internal and external flows: Introduction to the fundamentals of CFD*, Butterworth-Heinemann, Oxford.

Hoefler, T., Dinan, J., Buntinas, D., Balaji, P., Barrett, B., Brightwell, R., Gropp, W., Kale, V. & Thakur, R. (2013), 'MPI + MPI: A New Hybrid Approach to Parallel Programming with MPI Plus Shared Memory', *Journal of Computing* (DOI 10.1007/s00607-013-0324-2).

Hu, F. & Atkins, H. (2002), 'Eigensolution analysis of the discontinuous Galerkin method with nonuniform grids. Part I: One space dimension', *J. Comput. Phys.* **182**, 516–545.

Hussaini, M. & Zang, T. (1987), 'Spectral methods in fluid dynamics', *Annual review of Fluid Mechanics* **19**, 339–367.

Karniadakis, G. (1990), 'Spectral Element-Fourier Methods for Incompressible Turbulent Flows', *Computer Methods in Applied Mechanics and Engineering* **80**, 367–380.

Karniadakis, G., Israeli, M. & Orszag, S. (1991), 'High-Order Splitting Methods for the Incompressible Navier-Stokes Equations', *J. Comput. Phys.* **97**, 414–443.

Karniadakis, G. & Kirby, R. (2003), *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation*, Cambridge University Press.

Karniadakis, G. & Sherwin, S. (2005), *Spectral/hp element methods for computational fluid dynamics*, Numer. Math. Sci. Comp., second edn, Oxford University Press, Oxford.

Karypis, G. (2013), *METIS's Manual*, 5.1.0 edn, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455.

Khronos OpenCL Working Group (2008), *The OpenCL Specification, version 1.0.29*. **URL:** *http://khronos.org/registry/cl/specs/opencl-1.0.29.pdf*

Kim, J. (1989), 'On the structure of pressure fluctuations in simulated turbulent channel flow', *Journal of Fluid Mechanics* **205**, 421–451.

Kim, J., Moin, P. & Moser, R. (1987), 'Turbulence statistics in fully developed channel flow at low Reynolds number', *Journal of Fluid Mechanics* **177**, 133–166.

Kim, J. & Sandberg, R. (2012), 'Efficient parallel computing with a compact finite difference scheme', *Computers & Fluids* **58**(1), 70–87.

Kirby, R. & Sherwin, S. (2006*a*), 'Stabilisation of spectral/hp element methods through spectral vanishing viscosity: Application to fluid mechanics modelling', *Computer Methods in Applied Mechanics and Engineering* **195**(23–24), 3128–3144.

Kirby, R. & Sherwin, S. (2006*b*), 'The Nektar++ project'. http://www.nektar.info.

Koberg, H. (2007), Turbulence control for drag reduction with active wall deformation, PhD thesis, Imperial College London.

Li, N. & Laizet, S. (2010), $2DECOMP\&FFT$ - a highly scalable 2D decomposition library and FFT interface, *in* 'Cray User Group 2010 conference'.

Liang, C., Cox, C. & Plesniak, M. (2013), 'A comparison of computational efficiencies of spectral difference method and correction procedure via reconstruction', *Journal of Computational Physics* **239**, 138–146.

Lörcher, F., Gassner, G. & Munz, C. (2008), 'An explicit discontinuous Galerkin scheme with local time-stepping for general unsteady diffusion equations', *Journal of Computational Physics* **227**(11), 5649–5670.

Lusk, E. & Chan, A. (2008), 'Early Experiments with the OpenMP/MPI Hybrid Programming Model', *Lecture Notes in Computer Science* **5004**, 36–47.

Markall, G., Slemmer, A., Ham, D., Kelly, P., Cantwell, C. & Sherwin, S. (2013), 'Finite element assembly strategies on multi-core and many-core architectures', *International Journal for Numerical Methods in Fluids* **71**(1), 80–97.

McIver, D., Blackburn, H. & Nathan, G. (2000), 'Spectral element-Fourier methods applied to simulation of turbulent pipe flow', *ANZIAM Journal* **42**, 954–977.

Meuer, H., Strohmaier, E., Dongarra, J. & Simon, H. (2013), 'Top500 supercomputer sites', http://www.top500.org.

Mininni, P., Rosenberg, D., Reddy, R. & Pouquet, A. (2011), 'A hybrid MPI–OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence', *Parallel Computing* **37**(6–7), 316–326.

Moin, P. & Kim, J. (1982), 'Numerical investigation of turbulent channel flow', *Journal of Fluid Mechanics* **118**, 341–377.

Moser, R., Kim, J. & Mansour, N. (1999), 'Direct numerical simulation of turbulent channel flow up to $\mathrm{Re}_\tau = 590$', *Physics of Fluids* **11**(4), 943–945.

Nichols, B., Buttlar, D. & Farrell, J. (1996), *Pthreads programming*, O'Reilly and Associates, CA, USA.

No, J., Park, S., Perez, J. & Choudhary, A. (2002), 'Design and Implementation of a Parallel I/O Runtime System for Irregular Applications', *Journal of Parallel and Distributed Computing* **62**(2), 193 – 220.

Nogueira Jr., A. & Bittencourt, M. (2007), 'Spectral/HP finite elements applied to linear and non-linear structural elastic problems', *Latin American Journal of Solids and Structures* **4**(1), 61.

Orszag, S. (1980), 'Spectral methods for problems in complex geometries', *J. Comput. Phys.* **37**(1), 70–92.

Orszag, S. & Israeli, M. (1974), 'Numerical simulation of viscous incompressible flows', *Annual Review of Fluid Mechanics* **6**, 281–318.

Patera, A. (1984), 'A spectral element method for fluid dynamics: Laminar flow in a channel expansion', *Journal of Computational Physics* **54**, 468–488.

Patterson, G. & Orszag, S. (1971), 'Spectral calculations of isotropic turbulence: Efficient removal of aliasing interactions', *Physics of Fluids* **14**, 2538–2541.

Peterson, T. (1991), 'A note on the convergence of the discontinuous Galerkin method for a scalar hyperbolic equation', *SIAM J. Numer. Anal.* **28**, 133–140.

Pope, S. (2000), *Turbulent Flows*, Cambridge University Press.

Reed, W. & Hill, T. (1973), Triangular Mesh Methods for the Neutron Transport Equation, Technical report, Los Alamos Scientific Laboratory, Los Alamos, NM.

Roberts, M. & Bowman, J. (2011), 'Dealiased convolutions for pseudospectral simulations', *Journal of Physics: Conference Series* **318**, 072037.

Rogallo, R. (1981), Numerical experiments in homogeneous tubulence, Technical Memorandum 81315, NASA, Ames Research Center.

Rønquist, E. (1988), Optimal spectral element methods for the Unsteady three-dimensional incompressible Navier-Stokes equations, PhD thesis, Massachusetts Institute of Technology.

Sanders, J. & Kandrot, E. (2010), *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st edn, Addison-Wesley Professional.

Sharma, A., Abdessemed, N., Sherwin, S. & Theofilis, V. (2011), 'Transient growth mechanisms of low Reynolds number flow over a low-pressure turbine blade', *Theoretical and Computational Fluid Dynamics* **25**(1–4), 19–30.

Sherwin, S. (2000), 'Dispersion Analysis of the Continuous and Discontinuous Galerkin Formulations', *Lecture Notes in Computational Science and Engineering* .

Sherwin, S. & Karniadakis, G. (1995), 'A Triangular Spectral Element Method; Applications to the Incompressible Navier-Stokes Equations', *Computer Methods in Applied Mechanics and Engineering* **123**, 189–229.

Stroud, A. (1966), *Gaussian quadrature formula*, first edn, Prentice-Hall.

Szabó, B. & Babuška, I. (1991), *Finite Element Analysis*, Wiley, New York.

Takahashi, D. (2003), 'Efficient implementation of parallel three-dimensional FFT on clusters of PCs', *Computer Physics Communications* **152**(2), 144–150.

Tufo, H. & Fischer, P. (2001), 'Fast Parallel Direct Solvers For Coarse Grid Problems', *J. Par. & Dist. Comput.* **61**, 151–177.

Vos, P. (2010), From h to p Efficiently: Optimising the Implementation of Spectral/hp Element Methods, PhD thesis, Imperial College London.

Vos, P., Eskilsson, C., Bolis, A., Chun, S., Kirby, R. & Sherwin, S. (2011), 'A generic framework for time-stepping partial differential equations (PDEs): general linear methods, object-oriented implementation and application to fluid problems', *International Journal of Computational Fluid Dynamics* **25**(3), 107–125.

Vos, P., Sherwin, S. & Kirby, R. (2010), 'From h to p efficiently: Implementing finite and spectral/hp element methods to achieve optimal performance for low-and high-order discretisations', *Journal of Computational Physics* **229**(13), 5161–5181.

Warburton, T. (1999), Spectral/hp methods on polymorphic multi-domains: algorithms and applications, PhD thesis, Brown University.

Warburton, T. & Hagstrom, T. (2008), 'Taming the CFL number for Discontinuos Galerkin Methods on Structured Meshes', *SIAM Journal on Numerical Analysis* **46**(6), 3151–3180.

Warburton, T., Lomtev, I., Du, Y., Sherwin, S. & Karniadakis, G. (1999), 'Galerkin and discontinuous Galerkin spectral/hp methods', *Computer Methods in Applied Mechanics and Engineering* **175**, 343–359.

Williams, D., Canstonguay, P., Vincent, P. & Jameson, A. (2013), 'Energy stable flux reconstruction schemes for advection-diffusion problems on triangles', *J. Comput. Phys.* **250**, 53–76.

Wood, W. (1990), *Practical Time-stepping Schemes*, Clarendon Press.

Zhang, Q. & Shu, C. (2010), 'Stability analysis and a priori error estimates to the third order explicit Runge-Kutta discontinuous Galerkin Method for scalar conservation laws', *SIAM Journal on Numerical Analysis* **48**(3), 1038–1063.

Zienkiewicz, O., Taylor, R., Sherwin, S. & Peiro, J. (2003), 'On Discontinuous Galerkin Methods', *International Journal for Numerical Methods and Engineering* **58**, 1119–1148.

Appendix A

Nektar++

In this appendix we present *Nektar++* structure and we give a brief overview of all its sub-libraries. The development and the extension of *Nektar++* implementation was part of the research project, as well as the creation of a substantial documentation of the code for future users and/or developers[1]. The encapsulation of key concepts and the design of flexible algorithms is a key point for a piece of software which intends to implement a spectral/*hp* element method in a maintainable and coherent manner.

Nektar++ ++ implementation philosophy is based on the isolation of the spectral/*hp* element method fundamental building-blocks. Once the mathematical (or geometrical) concepts have been isolated, they can be encapsulated in a C++ virtual object (or a cascade of them). This yields an almost perfect decoupling of the various implementation aspects promoting code reusability, modular development and high maintainability. Fig. A.1 shows a schematic representation of *Nektar++* where the following libraries are highlighted:

- the supporting utilities sub-library (LibUtilities),
- the standard elemental region sub-library (StdRegions),
- the parametric mapping sub-library (SpatialDomains),
- the local elemental region sub-library (LocalRegions),

[1]Part of what is reported in this appendix and further information can be found on *Nektar++* website (Kirby & Sherwin 2006*b*).

- the global region sub-library (MultiRegions).

This structure can also be related to the formulation of a global spectral/*hp* element expansion as

$$u(\boldsymbol{x}) = \overbrace{\sum_{e\in\mathcal{E}} \underbrace{\sum_{n\in\mathcal{N}} \phi_n^e(\boldsymbol{x})\hat{u}_n^e}_{\text{LocalRegions library}}}^{\text{MultiRegions library}} = \sum_{e\in\mathcal{E}} \underbrace{\sum_{n\in\mathcal{N}} \phi_n^{std} \overbrace{\left([\chi^e]^{-1}(\boldsymbol{x})\right)}^{\text{SpatialDomains library}} \hat{u}_n^e}_{\text{StdRegions library}} \tag{A.1}$$

where e indicates the element index within the $\mathcal{E}$ element collection in which the physical domain Ω has been subdivided. As mentioned in Chapter 2, ϕ_n^e is one of the $\mathcal{N}$ polynomials used to approximate the solution on the element e and $\hat{u}_n^e$ is the associated coefficient (degree of freedom). The second part of Eq. (A.1) highlights the mapping $[\chi^e]^{-1}$ between the real domains and the standard domain where all basic operations are performed.

In addition to the libraries layout, the *Solvers* and *SolvUtilities* blocks are reported in Fig. A.1. They are meant to provide a global view of the software, where the basic sub-libraries are used to solve PDE systems.

A.1 *LibUtilities* Sub-library

This is the most basic sub-library, where all the generic implementations which are not spectral/*hp* element specific are collected. We can find here:

- *BasicConst*: definition of all the constants used in *Nektar++* and the values precision to facilitate portability across architectures.

- *BasicUtils*: external libraries interface (Boost, Metis, TinyXml, etc) and basic algorithms to generalise operations between vectors and scalars. It also contains classes to read and parse input files.

- *Communication*: it is the parallelisation abstraction of *Nektar++* ++. It contains a cascade of virtual objects which generalise all the operations involved in a parallel execution.

- *FFT*: it is a cascade of classes meant to wrap external FFT libraries. It allows the

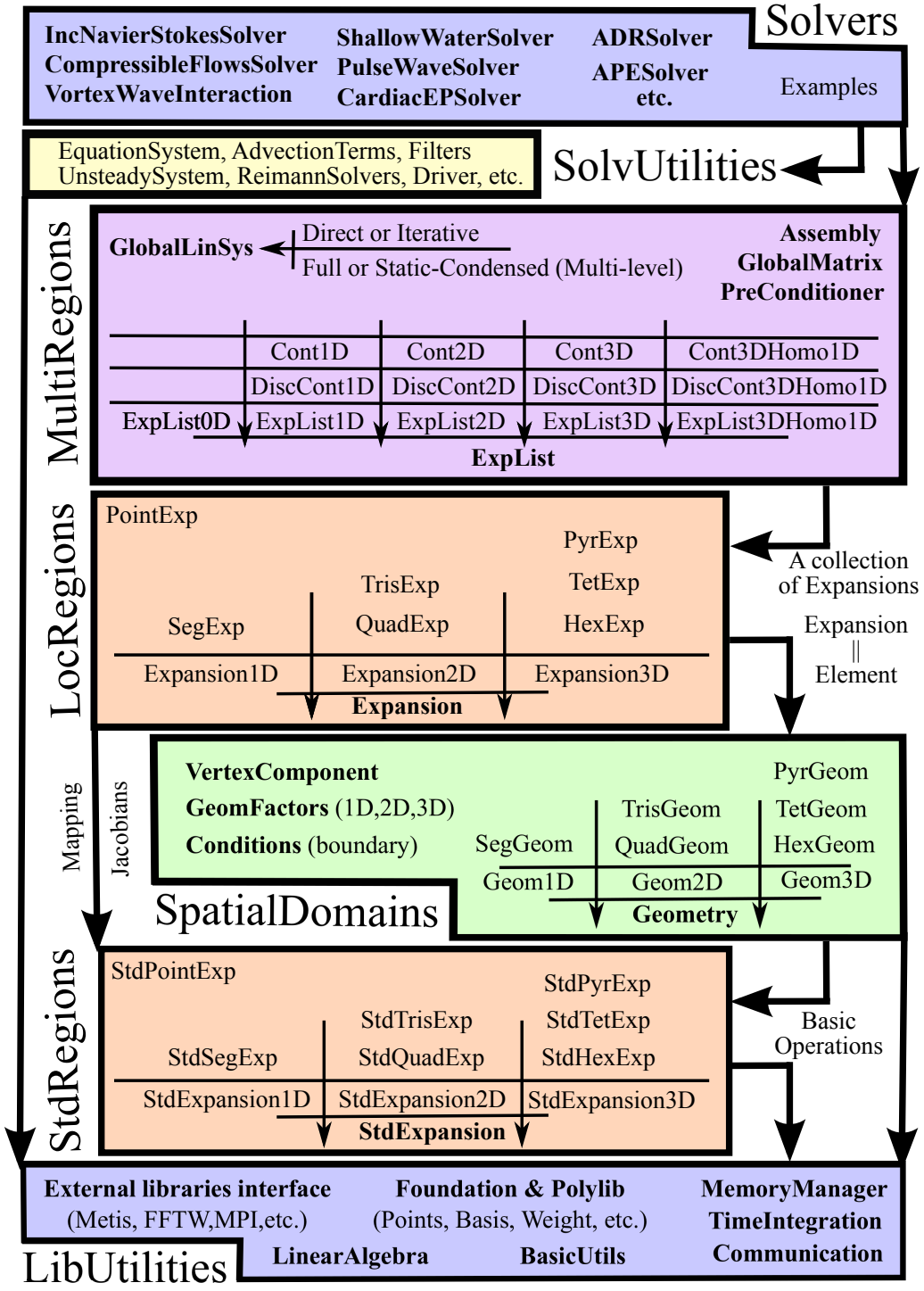


Figure A.1: *Nektar++* framework. Structure of the of the sub-libraries and contents. Arrows indicate some of the inheritance paths.

usage of any FFT library, although only a virtual object to interface FFTW has been implemented so far.

- *Foundations*: it contains series of C++ objects representing all the types of quadrature points, weights and basis.

- *Interpreter*: it contains classes designed to parse, interpret and evaluate equations in string format.

- *LinearAlgebra*: it contains the interface for BLAS, Lapack and Arpack and it generalises the matrix-vector operations.

- *Memory*: classes to control memory usage and allocation/deallocation of memory.

- *Polylib*: it contains a class used to define Jacobi polynomials.

- *TimeItegration*: it encapsulates the concept of time-integration.

A.2 *StdRegions* Sub-library

The *StdRegions* sub-library bundles all classes that mimic a spectral/*hp* element expansion on a standard region

$$u(\boldsymbol{\xi}) = \sum_{n \in \mathcal{N}} \phi_n(\boldsymbol{\xi}) \hat{u}_n, \tag{A.2}$$

where $\boldsymbol{\xi}$ are the coordinates on the standard reference system. The data required to define an expansion on a standard region are:

- the coefficient vector $\hat{\boldsymbol{u}}$,

- the polynomial expansion $\phi_n(\boldsymbol{\xi})$ defined by the discrete basis matrix $\boldsymbol{B}$,

- the vector $\boldsymbol{u}$ which represents the values of the expansion at the quadrature points $\boldsymbol{\xi}_i$.

All standard shapes can be abstracted in a similar way. Therefore, it is possible to define these data structures in an abstract base class (i.e. the class *StdExpansion*). The methods which are identical among all shapes are implemented in this class. On the other

hand, the methods which are different from shape to shape are defined here and adapted later on using polymorphism.

The specialisation of the methods (integration, differentiation, etc.) goes down the inheritance tree, from methods which are common to all standard expansions (implemented in *StdExpansion*), methods which depend on the dimensionality of the expansion (implemented in *StdExpansion1D*, *StdExpansion2D* and *StdExpansion3D*) and methods which are shape-specific (implemented in *StdSegExp*, *StdQuadExp*, etc.).

A.3 *SpatialDomains* Sub-library

The *SpatialDomains* sub-library encapsulates the concept of mapping. Given a standard element defined in *StdRegions*, we need a series of operations to map operations defined over $\boldsymbol{\xi}$) to the real geometry (over $\boldsymbol{x}$), $i.e.$ we need a tool to move from the standard system to our real coordinate system (the one used in *LocalRegions*).

The most important class is the *Geometry*. These classes are the representation of an element in physical space and they are equipped with the following data structures:

- an object of *StdExpansion* class, and
- a data structure that contains the metric terms (Jacobian, derivative metrics) of the transformations.

A.4 *LocalRegions* Sub-library

The *LocalRegions* library is designed to encompass all classes that encapsulate the elemental spectral/*hp* expansions in physical space. A local expansion essentially is a standard expansion that has a additional coordinate transformation that maps the standard element to the local element. The classes in the *LocalRegions* sub-library are derived from the corresponding *StdExpansion* classes but they are supplied with an additional data member representing the geometry of the local element (coming from *SpatialDomain*). Depending on the shape-specific class in the *LocalRegions* library, this additional data member is an object of the corresponding class in the *Geometry* class structure.

A.5 *MultiRegions* Sub-library

In the *MultiRegions* sub-library we encapsulate the assembly concept. This library is meant to collect a series of subdomains, objects of *LocalRegions*, over which a spectral/*hp* expansion is defined, and then to assemble all the local contributions into the global discretisation defining at the same time the type of connectivity between elements.

The base class is *ExpList*. Following all the classes are the abstraction of a multi-elemental spectral/*hp* element expansion. The tree start from *ExpList* and then it specialises through dimensionality first (*ExpList1D*, *ExpList2D*, etc.) and connectivity afterward (*DisContField1D*, *ContField1D*, *DisContField2D*,etc.). Objects of these last classes should be used when solving partial differential equations using a discontinuous or continuous Galerkin approach. These classes enforce a coupling between elements and specify boundary conditions. Disregarding the connectivity between elements, we can define a global expansion as:

$$u^{\delta}(\boldsymbol{x}) = \sum_{e=1}^{N_{\rm el}} \sum_{n=0}^{\mathcal{N}^e-1} \hat{u}_n^e \phi_n^e(\boldsymbol{x}) \tag{A.3}$$

where

- $N_{\rm el}$ is the number of elements,
- $\mathcal{N}^e$ is the number of local expansion modes within the element e,
- $\phi_n^e(\boldsymbol{x})$ is the n^{th} local expansion mode within the element e,
- $\hat{u}_n^e$ is the n^{th} local expansion coefficient within the element e.

The Fouirer spectral/*hp* element approach is an extension of the 1D and the 2D spectral/*hp* element method. This technique allows to study 3D problems combining the spectral/*hp* element method with a spectral method. In the case of one homogenous direction, the third dimension (z-axis) is expanded with an harmonic expansion (a Fourier series). In each quadrature point of the Fourier discretisation we can find a 2D plane discretised with a 2D spectral/*hp* element expansion. In the case of two homogeneous directions a plane is discretised with a 2D Fourier expansion (y-z plane). In each one of the quadrature point of this 2D harmonic expansion there is a 1D spectral/*hp* element discretisation. The homogenous classes derive directly form *ExpList*, and they are *ExpListHomogeneous1D* and

ExpListHomogeneous2D. This classes are used to represent the collections of 2D (or 1D) spectral/*hp* element discretisations which are located in the Fourier expansion quadrature points to create a 3D problem.

Appendix B

Time-Stepping Schemes Tableau

In this appendix we report some further time-integration schemes which have been implemented in *Nektar++* framework to provide a full reference of the implementation and an extension to what reported in Chapters 2, 3 and in (Vos et al. 2011). This can also been seen as a guideline on how insert new time-integration methods for future *Nektar++* users.

B.1 Multi-Step Methods

In contrast to multi-stage methods, multi-step methods have a single stage, but the solution at the new time-level is computed as a linear combination of information at the r previous time-levels. Linear multi-step methods can be formulated to satisfy the relation

$$\boldsymbol{y}_n = \sum_{i=1}^{r} \alpha_i \boldsymbol{y}_{n-i} + \Delta t \sum_{i=0}^{r} \beta_i \boldsymbol{F}_{n-i}. \tag{B.1}$$

B.1.1 Forward Euler

The Forward Euler method, which corresponds to the first order Adams-Bashforth scheme, is an explicit one-step method of the form

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\boldsymbol{f}(\boldsymbol{y}_{n-1}) \right), \tag{B.2}$$

which has the following GLM representation

$$\left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] = \left[\begin{array}{c|c} 0 & 1 \\ \hline 1 & 1 \end{array}\right]. \tag{B.3}$$

B.1.2 Backward Euler

The Backward Euler method is a one-step implicit method, identical to the first order Adams-Moulton scheme and the first order implicit BDF method, and can be written as

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\boldsymbol{f}(\boldsymbol{y}_n)\right), \tag{B.4}$$

which has the following GLM representation

$$\left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] = \left[\begin{array}{c|c} 1 & 1 \\ \hline 1 & 1 \end{array}\right]. \tag{B.5}$$

B.1.3 Adams-Bashforth Order 2

This explicit two-step method can be written as

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\frac{3}{2}\boldsymbol{f}(\boldsymbol{y}_{n-1}) - \frac{1}{2}\boldsymbol{f}(\boldsymbol{y}_{n-2})\right), \tag{B.6}$$

which has the following GLM representation

$$\left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] = \left[\begin{array}{c|cc} 0 & 1 & 0 \\ \hline \frac{3}{2} & 1 & \frac{-1}{2} \\ 1 & 0 & 0 \end{array}\right]. \tag{B.7}$$

B.1.4 Adams-Bashforth Order 3

The third order Adams-Bashforth schems is an explicit three-step method of the form

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\frac{23}{12}\boldsymbol{f}(\boldsymbol{y}_{n-1}) - \frac{4}{3}\boldsymbol{f}(\boldsymbol{y}_{n-2}) + \frac{5}{12}\boldsymbol{f}(\boldsymbol{y}_{n-3})\right), \tag{B.8}$$

which has the following GLM representation

$$\left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] = \left[\begin{array}{c|cccc} 0 & 1 & \frac{23}{12} & -\frac{4}{3} & \frac{5}{12} \\ \hline 0 & 1 & \frac{23}{12} & -\frac{4}{3} & \frac{5}{12} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{array}\right]. \tag{B.9}$$

B.1.5 Adams-Moulton Order 2

The Adams-Moulton method, is a family of multi-step implict schemes for solving ODEs. While the first order scheme corresponds to the Backward Euler approach mentioned before, the second order ones has the following form

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\frac{1}{2}\boldsymbol{f}(\boldsymbol{y}_n) + \frac{1}{2}\boldsymbol{f}(\boldsymbol{y}_{n-1}) \right), \tag{B.10}$$

which has the following GLM representation

$$\left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] = \left[\begin{array}{c|cc} \frac{1}{2} & 1 & \frac{1}{2} \\ \hline \frac{1}{2} & 1 & \frac{1}{2} \\ 1 & 0 & 0 \end{array}\right]. \tag{B.11}$$

B.2 Multi-Stage Methods

Multi-stage methods consist of a single step and many stages. They can be represented as a general linear method with $r = 1$. It is sufficient to write $U = [\,1\ \ 1\ \cdots\ 1\,]^\top$, $V = [1]$ and to set the coefficient matrices A and B to the matrix A and the single row $b^\top$ of the corresponding Butcher tableau.

B.2.1 Explicit Runge-Kutta 2

The explicit second order Runge-Kutta scheme has the following Butcher table

$$\begin{array}{c|c} c & A \\ \hline & b^\top \end{array} = \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \tag{B.12}$$

which has the following GLM representation

$$\left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] = \left[\begin{array}{cc|c} 0 & 0 & 1 \\ 1 & 0 & 1 \\ \hline \frac{1}{2} & \frac{1}{2} & 1 \end{array}\right]. \tag{B.13}$$

B.2.2 Explicit Runge-Kutta 4

The explicit fourth-order Runge-Kutta scheme is one of the most known and used scheme for the solution of ODEs and its Butcher tableau is

$$\begin{array}{c|c} c & A \\ \hline & b^\top \end{array} = \begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}, \tag{B.14}$$

which has the following GLM representation

$$\left[\begin{array}{c|c} A & U \\ \hline B & V \end{array}\right] = \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 0 & 1 \\ 0 & \frac{1}{2} & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ \hline \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} & 1 \end{array}\right]. \tag{B.15}$$

B.3 Implicit-Explicit Methods

In case we want to treat some of the numerical operators explicitly and some implicitly we can take advantages of the IMEX schemes family, which encompasses a series of multi-stage and multi-step schemes.

B.3.1 Backward-Forward Euler

The IMEX first order multi-step scheme can be written as

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\boldsymbol{g}(\boldsymbol{y}_n) + \boldsymbol{f}(\boldsymbol{y}_{n-1}) \right), \tag{B.16}$$

and it yields the following GLM format

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|cc} 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{array}\right]. \tag{B.17}$$

B.3.2 CN-AB

This is a second order Crank-Nicholson/Adams-Bashforth linear multi-step scheme of the form

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\frac{1}{2}\boldsymbol{g}(\boldsymbol{y}_n) + \frac{1}{2}\boldsymbol{g}(\boldsymbol{y}_{n-1}) + \frac{3}{2}\boldsymbol{f}(\boldsymbol{y}_{n-1}) - \frac{1}{2}\boldsymbol{f}(\boldsymbol{y}_{n-2}) \right), \tag{B.18}$$

and it can be represented using the following GLM matrix

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|cccc} \frac{1}{2} & 0 & 1 & \frac{1}{2} & \frac{3}{2} & -\frac{1}{2} \\ \hline \frac{1}{2} & 0 & 1 & \frac{1}{2} & \frac{3}{2} & -\frac{1}{2} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right]. \tag{B.19}$$

B.3.3 Stiffly-Stable IMEX-3

The third order stiffly-stable IMEX scheme defined for the velocity correction scheme in Chapter 2 is

$$\boldsymbol{y}_n = \boldsymbol{y}_{n-1} + \Delta t \left(\frac{1}{2}\boldsymbol{g}(\boldsymbol{y}_n) + \frac{1}{2}\boldsymbol{g}(\boldsymbol{y}_{n-1}) + \frac{3}{2}\boldsymbol{f}(\boldsymbol{y}_{n-1}) - \frac{1}{2}\boldsymbol{f}(\boldsymbol{y}_{n-2}) \right), \tag{B.20}$$

and it can be represented as

$$\left[\begin{array}{c:c|c} A^{\mathrm{IM}} & A^{\mathrm{EX}} & U \\ \hline B^{\mathrm{IM}} & B^{\mathrm{EX}} & V \end{array}\right] = \left[\begin{array}{c:c|cccccc} \frac{6}{11} & 0 & \frac{18}{11} & -\frac{9}{11} & \frac{2}{11} & \frac{18}{11} & -\frac{18}{11} & \frac{6}{11} \\ \hline \frac{6}{11} & 0 & \frac{18}{11} & -\frac{9}{11} & \frac{2}{11} & \frac{18}{11} & -\frac{18}{11} & \frac{6}{11} \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right]. \tag{B.21}$$