TÉCNICO LISBOA

A Comparative Study on the Parallel Efficiency of Low- and High-Order Finite-Volume Schemes

Leonardo da Silva Rosa e Oliveira

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor: Professor Duarte Manuel Salvador Freire Silva de Albuquerque

Examination Committee

Chairperson:	Professor Fernando José Parracho Lau
Supervisor:	Professor Duarte Manuel Salvador Freire Silva de Albuquerque
Member of the Committee:	Professor Mário António Prazeres Lino da Silva

December 2022

Resumo

Embora o uso de esquemas numéricos de segunda ordem permaneça popular devido à sua robustez de convergência e facilidade de implementação, os esquemas de alta ordem surgiram como alternativas promissoras devido à sua precisão numérica e eficiência computacional. No entanto, um argumento recorrente contra a adoção destes esquemas é que, embora sejam mais eficientes que os esquemas de segunda ordem quando executados em série, têm um desempenho fraco em cálculo paralelo.

Este trabalho compara o desempenho em paralelo de esquemas *face least-squares* (FLS) de segunda, quarta, sexta e oitava ordem, que foram previamente desenvolvidos para malhas bidimensionais e não-estruturadas, usando o método de volumes finitos.

O algoritmo FLS foi estendido para três dimensões, e um código correspondente com capacidades de computação paralela foi desenvolvido. O desempenho paralelo dos esquemas de segunda a oitava ordem foi estudado para um problema de convecção-difusão tridimensional usando malhas cartesianas regulares, no supercomputador Oblivion. Os resultados mostraram uma boa escalabilidade em todos os esquemas, sendo que o esquema de oitava ordem apresentou um desempenho equivalente ao de segunda ordem. A eficiência de memória e o tempo de execução também foram analisados em cálculo paralelo, com os esquemas de alta ordem apresentando uma vantagem clara.

Adicionalmente, métodos alternativos referentes ao polinómio de regressão usado pelos esquemas FLS foram testados para problemas bidimensionais, resultando numa maior precisão numérica. Em particular, o uso do polinómio de regressão estendido nas direções tangentes à face mostrou-se muito promissor e, logo, recomenda-se a sua adoção como uma norma em esquemas FLS.

Palavras-chave: Esquemas de alta ordem, Escalabilidade paralela, Computação de alto desempenho, Método de volume finito, Equação de convecção-difusão

Abstract

Although the use of second-order numerical schemes remains widespread due to their convergence robustness and implementation ease, high-order schemes have emerged as promising alternatives due to their increased numerical accuracy and computational efficiency. However, one of the recurring arguments against the adoption of high-order schemes is that, even though they are more efficient than second-order schemes when executed serially, they perform poorly in parallel computing.

The present work compared the parallel performance of second-, fourth-, sixth-, and eighth-order face least-squares (FLS) schemes, which had been previously developed for two-dimensional, unstructured grids under the finite volume method framework.

The FLS algorithm was extended to three dimensions, and a corresponding code with parallel computing capabilities was developed. The parallel performance of the second- through eighth-order schemes was studied for a three-dimensional convection-diffusion problem using regular Cartesian grids, at the Oblivion supercomputer. The obtained results showed good scalability across all schemes, with the eighth-order scheme performing equivalently to the second-order one. Their memory and runtime efficiency were also analyzed for parallel execution, with the high-order schemes presenting an overwhelming advantage.

Additionally, alternative methods regarding the regression polynomial for the FLS schemes were tested for two-dimensional problems, yielding a higher numerical accuracy. In particular, the use of extended regression polynomial showed great promise and should be considered as a standard in FLS schemes.

Keywords: High-order schemes, Parallel scalability, High-performance computing, Finite volume method, Convection-diffusion equation

Acknowledgements

The current work was part of the project “High-Order Immersed Boundary Method for Moving Body Problems (HIBforMBP)”, funded by the Foundation for Science and Technology (FCT) in Portugal with the project reference PTDC/EME-EME/32315/2017. Furthermore, the work was also supported by the Advanced Computing EuroCC Program at the Instituto Superior Técnico (IST), funded by the European High Performance Computing Joint Undertaking (EuroHPC JU).

Most simulations presented in this work were performed at the Oblivion supercomputer managed by the HPC Center of the University of Évora and acquired under the “Enabling Green E-science for the SKA Research Infrastructure (ENGAGE SKA)”, reference POCI-01-0145-FEDER-022217, funded by COMPETE 2020 and FCT. The simulations were funded by the FCT program “Concurso de Projetos de Computação Avançada - 2ª Edição” with the project reference CPCA/A0/426012/2021.

Contents

List of Tables

List of Figures

Acronyms

API	application programming interface
ASM	additive Schwartz method
BiCGSTAB	biconjugate gradient stabilized method
CD2	second-order central differences
CENO	central ENO
CFD	computational fluid dynamics
CG	conjugate gradient
CPU	central processing unit
CS	control surface
CV	control volume
DG	discontinuous Galerkin
DNS	direct numerical simulation
DOF	degree of freedom
ENO	essentially non-oscillatory
FDM	finite difference method
FEM	finite element method
FFT	fast Fourier transform
flop/s	floating-point operations per second
FLS	face least-squares
FLS2	second-order FLS
FLS4	fourth-order FLS
FLS6	sixth-order FLS
FLS8	eighth-order FLS
FVM	finite volume method
GAMG	geometric algebraic multigrid
GPGPU	general-purpose graphics processing unit
GPU	graphics processing unit

HIBforMBP High-Order Immersed Boundary Method for Moving Body Problems

HPC high-performance computing

IEEE Institute for Electrical and Electronics Engineers

KSP Krylov subspace method

LAPACK Linear Algebra Package

LASEF Laboratório de Simulação em Energia e Fluidos

LES large-eddy simulation

MACC Minho Advanced Computing Center

MLS moving least-squares

MPI Message Passing Interface

NASA National Aeronautics and Space Administration

NS Navier-Stokes

OpenMP Open Multi-Processing

PC preconditioner

PDE partial differential equation

PETSc Portable, Extensible Toolkit for Scientific Computation

RANS Reynolds-averaged Navier-Stokes

RNCA Rede Nacional de Computação Avançada

SRT solver runtime

TENO targeted ENO

TVD total variation diminishing

WENO weighted ENO

WLS weighted least-squares

Nomenclature

Subscripts

C	Centroid
f	Face f
t	Triangle t

Greek Symbols

Γ	Diffusive coefficient
η^{st}	Strong scaling efficiency
η^{wk}	Weak scaling efficiency
φ	Transportable variable
ψ	Speedup
$\overline{\psi}$	Algorithmic speedup

Mathematical Operators

$\max$	Maximum
$\min$	Minimum
∇	Gradient
$\nabla\cdot$	Divergence
$\lfloor\cdot\rceil$	Rounding to nearest integer

Roman Symbols

$\mathcal{F}$	Set of faces
$\mathcal{G}$	Set of cubature points
$\mathcal{G}_{T_2}$	Set of cubature points over a triangle
Q	Source term
$\mathcal{T}$	Set of triangles
$\vec{v}$	Velocity vector
w_G	Cubature weight
$\vec{x}$	Position vector

Chapter 1

Introduction

The myriad happenings that occur in the everyday world around us are in essence just a collection of physical phenomena. These are in turn described by their own governing set of mathematical relations. Since these relations often describe the rate in which quantities of interest change in relation to reference quantities, they take the form of differential equations. If such an equation depends on more than one reference quantity or variable, it is classified as a partial differential equation (PDE). PDEs lie at the base of many scientific fields such as finance, chemical reactions, neuroscience, optics, quantum mechanics, thermodynamics, and – of more interest to the current work – fluid mechanics [1].

The development of scientific fields such as fluid mechanics has historically followed a two-pronged approach based on theoretical derivation and experimental observation [2]. The latter should involve the use of an experimental model replicating the desired phenomena in full-scale. While obtaining actual measurements is crucial to study and understand a phenomenon, all measurements inherently suffer from measurement errors due to equipment. Furthermore, it is often difficult or even impossible to build a full-scale model, leading to the use smaller-scale setups from where the final conclusion have to be extrapolated, leading to the yet other error sources. As such, experimental investigations have often been hindered by insufficient accuracy, expensive or even nonexistent equipment, and difficulty in reproducing results.

On the other hand, theoretical prediction has its own set of advantages and limitations, originating from the consideration of a mathematical model instead of a physical one [2]. This set of methods is inherently low-cost and accessible, essentially requiring only paper, pencil, and patience. Alas, the complex nature of many governing systems of PDEs present an often insurmountable challenge in obtaining closed analytical solutions, usually being possible only after several simplifying assumptions or when considering very simple configurations. Indeed, proving either existence or nonexistence of a solution to the Navier-Stokes (NS) equations is one of the seven Clay Millennium Prize Problems [3], each worth US$ 1 million. As such, even though theoretical derivation holds a large importance to the formal development and understanding of a field, it is often of limited practical interest to complex engineering problems.

The aforementioned experimental-theoretical duality formed the sole basis for scientific contributions in fluid dynamics up until the past century. More specifically, the field of experimental fluid dynamics, for example, had its foundations laid in the seventeenth century, while its companion field of theoretical fluid dynamics was developed throughout the eighteenth and nineteenth centuries [4]. The development of computers in the twentieth century, however, birthed a novel “third approach” – computational fluid dynamics (CFD).

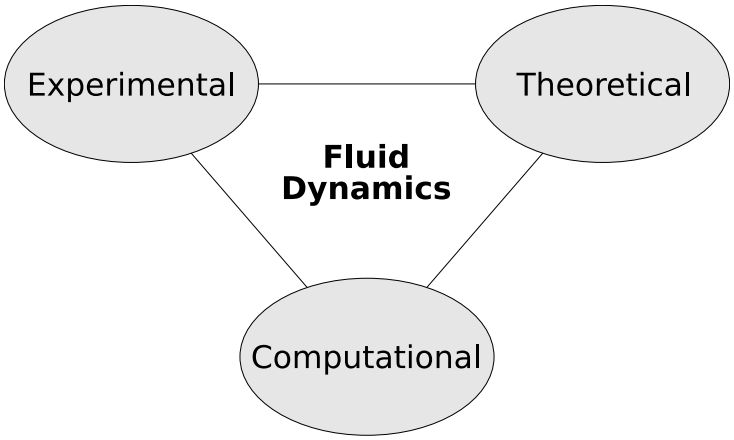


Figure 1.1 The current three-pronged approach to studying fluid dynamics. Adapted from Anderson [4].

Hirsch [5] defines CFD as "the set of methodologies that enable the computer to provide us with a numerical simulation of fluid flows." By using these methods, one can approximately solve the PDEs that govern fluid dynamics (NS, Euler, etc.), thus obtaining a set of data for the desired flow problem. In this sense, CFD is closely related to the experimental approach and may even be regarded as a "numerical experiment" [4], but based on a mathematical instead of a physical model. As such, it combines characteristics of both classical approaches and stands as an equally important aspect to the development of fluid dynamics. Figure 1.1 serves to illustrate the current three-pronged approach to the study of fluid dynamics and how each aspect complements the others.

The field of CFD underwent rapid and drastic developments in the latter half of the twentieth century [4]. This evolution has been fueled, in part, by the development of high-speed digital computers, specially its performance-to-cost ratio [6]. In the 1950s, computers operated at a few hundred floating-point operations per second (flop/s), and there was the field of CFD was only just appearing in academia. The 1970s witnessed the appearance of commercial CFD codes and the start of adherence by industry pioneers; however, the technology still limited applications mainly to two-dimensional flows.

By the turn of the century, the scenario was much different, with major industry powerhouses – such as Boeing [7] and Airbus [8] – using CFD as an indispensable tool. Johnson et al. [7] comment that the number of CFD simulations run at Boeing's commercial airplane division increased from 100-200 in 1973 to more than 20 000 in 2002, noting that these latter run cases involved much more complex physics and geometries. Currently, CFD is used by many industrial sectors besides aeronautics, including automotive [9, 10, 11], marine [12], energy [13, 14, 15, 16, 17], chemical [18, 19], food [20, 21], biomedical [22, 23], among others. According to the market research group IMARC [24], the global CFD market was valued at US$ 2.11 billion in the past year 2021 and is expected to reach US$ 3.43 billion by the year 2027.

1.1 Motivation

As previously noted, the development of CFD accompanied and was even aided by the growth in computing power over the twentieth century. However, the slowing down of Moore's Law – which correlates with computing power in a single processor – lead to widespread development of multi-core processors. Nowadays, all commercially available processors (except microcontrollers) are multi-core, even those included in mobile devices. Inevitably, the high-performance computing (HPC) community also adopted this paradigm, causing a shift from classical vector supercomputers to massively parallel

clusters – i.e. multiple computing units joined together by a high-speed interconnections. Ferziger et al. [6] points out that this increased the economic scalability of supercomputers by significantly reducing their associated costs, since they could now be built even using off-the-shelf components.

Inevitably, there has been a corresponding need for methods and algorithms to take advantage of these available resources. In a 2014 report [25], the National Aeronautics and Space Administration (NASA) presented its vision for state-of-the-art CFD in 2030, along with recommended research strategies for achieving it. The "effective utilization of HPC" was among the five key underlying areas identified in the report, with "robust CFD code scalability" being listed as a major technology gap. The authors pointed out that, at the time, most codes ran efficiently on at most ~1000 cores, which represented only ~0.1% capacity for the then largest supercomputers – a stark contrast to the 1980s when commercial codes ran on vector supercomputers at maximum utilization.

Practical engineering CFD calculations in industry have mostly used solvers based on the Reynolds-averaged Navier-Stokes (RANS) equations, which are obtained by averaging the NS equations over time (for steady flows) or over realizations (for unsteady flows). Given the non-linear nature of the original equations, this averaging technique leads to the appearance of new terms, and, therefore, the new equations do not form a closed set. As such, RANS methods require the use of approximations for these terms, in the form of turbulence models. Overall, these methods have proven to be an important tool in the engineering design process, such as in determining flow features and predicting aerodynamic loads. However, the main limitation of RANS is its inability to accurately predict separated flows, since the turbulence models approximate the entire spectrum of turbulence [26]. An alternate method arises by averaging (filtering) the NS equations over volumes of space, which then allows one to model only the small (subfilter) scales of motion while accurately solving the motions of scales larger than the filter – hence its name, large-eddy simulation (LES).

Traditionally, commercial solvers based on the aforementioned equations have used second-order accurate numerical schemes, because of their robustness in obtaining a converged solution and ease of implementation on any grid type. However, these schemes require very fine grid resolutions for accurate results, which may lead to prohibitively large computational problems in terms of memory requirement. A promising alternative arises in high-order schemes (usually defined as third-order and higher), since these can achieve the same error tolerance by using a coarser grid. This benefit has been shown for several grid types, including structured [27], curvilinear or deformed [28], and unstructured [29]. Additionally, Ekaterinaris [26] shows, through a simple rationale, that the use of high-order methods is also advantageous in terms of solver runtime, barring a drastic increase of the operation count for the linear solver. As such, this makes high-order schemes notably suitable for LES simulations, particularly of complex flows.

The scope of this Thesis is to compare the parallel performance of high- and low-order schemes, from a family of finite volume method (FVM) schemes called face least-squares (FLS). Although the aforementioned works point to this advantage in serial computations, the increased number of halo cells for the high-order methods drastically increases the matrix bandwidth, thus making its superiority in parallel computations a nontrivial conclusion. The present Thesis expands on the previous works of Vasconcelos [30, 31], Diogo [32], and Costa [33, 34], who developed high-order, convection-diffusion schemes (up to eighth-order) for transient problems on unstructured grids.

1.2 Background

Built upon the foundations of the classical finite difference method (FDM) to solve differential equations, the FVM was first developed in the 1960s. While the former is based on nodal relations for differential equations, the latter uses the integral form of conservation laws to provide a balance of forces acting on a discretized control volume [35]. The FVM assumed a prominent role in heat transfer and fluid flow problems through major contributions by the Imperial College research group led by DB Spalding [36, 37]. High-order FVM methods have as one of its precursors the 1990 work by Barth and Frederickson [38], with quadratic reconstructions (third-order) on unstructured grids for the Euler equations. Since then, efforts led to the implementation of fourth-order schemes by Ollivier-Gooch et al. [39, 40, 41] and sixth-order schemes by Clain et al. [42, 43, 44]. For an extensive background on high-order FVM schemes, readers may refer to the comprehensive review articles on structured grids by Ekaterinaris [26] and on unstructured grids by Wang [45].

One of the general criticisms towards high-order FVM schemes is that they lack robustness for flow regions of high gradient or discontinuities, being susceptible to spurious oscillations and thus possibly failing to converge [26]. The traditional way to deal with this is by using flux limiters such as total variation diminishing (TVD) schemes; however, Ferziger et al. [6] point out that this reduces the order of accuracy in the problematic region, where they must become first-order to maintain monotonicity. There have been several methods developed to deal with these oscillations while preserving high-order accuracy, with a major family of these being the essentially non-oscillatory (ENO) schemes [46] and its variants – weighted ENO (WENO) [47, 48], central ENO (CENO) [49], and targeted ENO (TENO) [50]. Cueto-Felgueroso et al. [51, 52] provided an alternative by implementing the moving least-squares (MLS) technique to FVM schemes. This technique provides "shape functions" to the grid by means of least-squares reconstructions, which uses a weight function related to the distance between the reference face and the cell sample. This technique differs from the one used in FLS by considering the mean values in cells, instead of the point-wise values of the centroids.

Nogueira et al. [53] compared two high-order methods – a MLS-based FVM scheme and a finite element method (FEM) scheme based on discontinuous Galerkin (DG) – for both inviscid and viscous flows. The study showed that, even though the accuracy of both methods are equivalent, the FVM is far less expensive computationally-wise, since it requires considerably less degrees of freedom (DOFs). Wang [45] explains that, while the number of DOFs of a FVM scheme depends only on the number of grid points, that of a FEM scheme also depends on the scheme's order of accuracy. This happens because each element in a FEM scheme has multiple DOFs (proportional to the scheme's order), while each cell in a FVM scheme has only a single DOF.

Parallel computing may be divided into two main approaches based on memory architecture: shared-memory systems, such as commercial multiprocessors; and distributed-memory systems, such as commodity clusters. The *de facto* programming standards for these two paradigms are, respectively, Open Multi-Processing (OpenMP) [54] and Message Passing Interface (MPI) [55]. In the HPC field, parallel computing using distributed-memory systems (i.e. distributed computing) is the preferred parallelization strategy due to its inherent scalability, since the number of computing elements incorporated into such systems is limited not by hardware, but only by cost and power [56]. Furthermore, it also allows for larger problems (e.g. whose memory requirements exceed that provided by a single machine) to be solved, since the problem and its associated memory cost is split between different machines. Shared-memory parallelism is sometimes used to provide finer-grain parallelization, especially in low-level software packages such as Linear Algebra Package (LAPACK) [57].

One of the widely used programming libraries for the parallel numerical solutions of PDEs is the Portable, Extensible Toolkit for Scientific Computation (PETSc) [58, 59]. This toolkit, which uses the MPI standard, provides a wide array of data structures and routines for linear algebra, linear solvers (Krylov subspace methods), data and grid management, among others. The scaling of the preconditioned Krylov methods provided in PETSc has been studied by Fettig et al. [60] and Sood et al. [61]. Additionally, Silva [62] also used PETSc to study the scalability of data balancing algorithms, focusing on machine load balancing.

Several authors have performed studies of parallel computation performance for CFD-inspired problems. Younas [63] compares the scalability of several PETSc solvers for both symmetric and non-symmetric Poisson problems. In turn, Eller et al. [64] compare the parallel performance of several PETSc preconditioners, for the case of FEM watershed simulations. Although the use of OpenMP is not directly supported by PETSc, Lange et al. [65, 66] and Weiland et al. [67] have performed studies using matrices derived CFD applications showing that this approach of mixed-mode parallelism may be advantageous for very large instances. There has also been scaling studies for CFD applications without using PETSc, for FVM [68, 69, 70], FEM [71, 72, 73, 74], and FDM [75, 76] schemes. Additionally, when studying WENO schemes, Tsoutsanis et al. [77] noted that the higher-order methods tended to have better scalability than their lower-order counterparts, attributing this difference to the former's larger ratio of computation time to communication time.

The publicly available results of different parallel numerical methods are quite scattered throughout the literature, which hampers comparisons between them. To aid this process, the Author gives a brief summary of the surveyed results in Table 1.1. These are classified according the parallel application programming interface (API) adopted, besides the type and order of studied scheme. The listed results of strong-scaling and, when available, weak-scaling efficiencies correspond to those for the highest number of used cores for which a sustained efficiency was achieved. Note that some studies used nonstandard baselines for measuring scaling efficiency – i.e. efficiency was measured against a reference runtime which was not that of a single-core execution (see Section 2.6 for a formal definition of strong- and weak-scaling efficiencies). Note that the Author's goal with this list is not to determine which methods are "better", as that would be naïve and reductive, given the different natures of both the surveyed methods and the corresponding problems. Instead, it is to provide examples of the level of parallel performance that the scientific community considers acceptable, at the least.

Table 1.1 Survey of available literature results on the parallel performance of different numerical schemes.

Reference	Parallel API	Scheme Type	Order	Scaling Efficiency	
				Strong	Weak
Costa et al. [78]	OpenMP	FVM polynomial reconstruction	2 4 6	56%, 16 cores 69%, 16 cores 75%, 16 cores	n/a n/a n/a
Junqueira-Junior et al. [76]	MPI	FDM, LES	2	68%, 3072 cores	76%, 524 cores
Tsoutsanis, Drikakis [79]	MPI	FVM WENO	3 5	67%, 6144 cores 87%, 6144 cores	n/a n/a
Antoniadis et al. [80]	MPI	FVM CWENO	4	89%, 131072 cores	n/a
Pei et al. [81]	MPI	FEM DG+WENO	3	91%, 100 cores	n/a
Guermond et al. [82]	MPI	FEM Continuous	2	63%, 98304 cores[a]	89%, 98304 cores[b]
Kronbichler et al. [83]	MPI	FEM DG		80%, 3200 cores[c]	86%, 512 cores

Nonstandard baselines: [a]1536 cores, [b]192 cores, [c]128 cores.

1.3 Present Contributions

This section aims to list the contributions made by the present Author to the high-order schemes and codes developed previously by the HIBforMBP group [30, 32, 33].[1] The contributions may be divided into three categories.

First, the Author significantly improved the existing Matlab code for the two-dimensional, unstructured, convection-diffusion solver. Using mostly the vectorization techniques available from the Matlab syntax, the overhauled code was able to obtain a ~50% reduction in runtime when compared to the original code's. The improved code will serve as a new base solver for further studies in the HIBforMBP group.

Second, the Author developed a parallel solver in C using PETSc, for the two-dimensional convection-diffusion equation with structured grids. This was mostly a re-implementation of the aforementioned Matlab code, with a few notable differences. The largest one is the underlying parallel, linear solver using the PETSc framework, which may even be customized by the user at runtime. Additionally, there is also the choice of using double (64-bit) or quadruple (128-bit) precision floating-point formats. Furthermore, since this implementation represented an intermediate step in the overall process – between the original Matlab solver and the final parallel, three-dimensional solver – this solver only considers structured grids. As such, the previously used, general schemes for unstructured grids were specifically simplified for the case of regular, Cartesian grids, allowing for significant time-savings when assembling the coefficient matrix.

Lastly, the Author developed, also in C using PETSc, a parallel solver for the three-dimensional convection-diffusion equation with structured grids. In terms of numerical methods, the existing Cartesian-specific schemes were extended to three dimensions, and a new, improved set of cubature rules was analyzed and chosen. In terms of computational implementation, this solver is mostly an evolution of the parallel, two-dimensional solver, with the notable addition of a runtime option for defining the user's preferred domain decomposition.

1.4 Thesis Outline

The present work contains four chapters. A brief summary of the remaining three chapters will now be given.

Chapter 2 presents the numerical scheme that will be studied and the metrics used to analyze its results. The scheme is derived from the three-dimensional convection-diffusion equation using the finite volume method. This is followed by some details regarding the scheme's implementation, in both two- and three-dimensions. Next, the metrics used to measure the results are explained, including their underlying concepts. Lastly, there is a brief discussion on the approach used for domain decomposition.

Chapter 3 presents the performed tests and obtained results. First, the scheme's implementation are verified numerically, showing their effective convergence orders. Next, two alternative methods for calculating the regression polynomial are presented and shown to produce more accurate results. Then, the best preconditioned iterative solver is chosen by comparing the solver runtimes of a small-scale, two-dimensional study. Finally, the results from the scalability and parallel efficiency study of the three-dimensional code is presented.

[1] See Appendix A for more information on past and new developed codes.

Chapter 4 presents the main conclusions of the present work, focusing on the scheme's parallel computation metrics. The chapter ends with a section on suggestions for future work, including the possible use of fast Fourier transform (FFT) to accelerate the solver and the next steps for scheme's implementation for the NS equations.

Chapter 2

Methodology

This chapter introduces the background needed for the subsequent presentation and analysis of results. First, the numerical method of face least-squares (FLS) is derived in Section 2.1 from the convection-diffusion equation, with special attention to possible simplifications due to the use of regular Cartesian grids. Then, Sections 2.2, 2.3 and 2.4 present the finer details of the FLS schemes and their implementation, namely the stencil generation algorithm, treatment of boundary cells, and choice of cubature rules. Section 2.6 introduces the measurements and metrics that will be used to analyze the results in the next chapter. Lastly, Section 2.5 explains the parallelization approach used for domain decomposition.

2.1 The Numerical Method – Face Least-Squares Schemes

The convection-diffusion equation (2.1) is a general conservation equation present in fields such as fluid flow and heat transfer [2]. In its general, steady-state format, the equation consists of three terms. The convective term represents the forced transport of quantity φ through a velocity field. The diffusive term quantifies the transport of quantity φ due to differences in its concentration across the domain. The convective and diffusive terms are each characterized by their own coefficient, respectively $\vec{v}$ and Γ. Lastly, the source term Q accounts for any local addition or removal of φ.

$$\underbrace{\nabla \cdot (\vec{v}\varphi)}_{\text{convection}} - \underbrace{\nabla \cdot (\Gamma \nabla \varphi)}_{\text{diffusion}} = \underbrace{Q}_{\text{source}} \tag{2.1}$$

2.1.1 Derivation of the Semi-Discretized Equations

Obtaining the numerical solution of an equation means having values of φ for a finite set of points belonging to the domain of interest. As such, this implies having to transform the original equation expressed in a continuous domain to an approximate equation expressed in a discrete domain. This process, known as discretization, will thus replace the original differential equations with a system of algebraic equations. The discretization used in this work will follow the finite volume method (FVM), which is of special interest to fluid mechanics since its results will exhibit integral conservation of the quantity φ [2, 84]. Following the procedure described by Moukalled et al. [84], the first step in this

method is to integrate (2.1) over a control volume (CV), as shown in (2.2).

$$\int_{CV} \Big(\nabla \cdot (\vec{v}\varphi) - \nabla \cdot (\Gamma \nabla \varphi) \Big) \, dV = \int_{CV} Q \, dV \, . \tag{2.2}$$

By applying the Divergence (Green-Gauss) Theorem on the previous equation, one transforms the volume integrals of the convective and diffusive terms into surface integrals over the surface of the CV – i.e. the control surface (CS) – resulting in

$$\int_{CS} \Big(\vec{v}\varphi - \Gamma \nabla \varphi \Big) \cdot d\vec{S} = \int_{CV} Q \, dV \, , \tag{2.3}$$

where the term $d\vec{S}$ represents the area normal of the surface, defined as positive outwards. Furthermore, defining $\mathcal{F}(I)$ as the set of all faces of cell I, then one obtains

$$\sum_{f \in \mathcal{F}(I)} \int \Big(\vec{v}\varphi - \Gamma \nabla \varphi \Big) \cdot d\vec{S}_f = \int_{CV} Q \, dV \, . \tag{2.4}$$

The source term integration – right-hand side of (2.4) – may be calculated using cubature[1] rules for numerical integration (see Section 2.4), resulting in

$$\int_{CV} Q \, dV \simeq V \sum_{g \in \mathcal{G}(I)} w_{G_g} Q(\vec{x}_g) \, , \tag{2.5}$$

where $\mathcal{G}(I)$ is the set of cubature points associated with cell I, $\vec{x}_g$ is a vector representing the spatial position of cubature point g, and w_{G_g} is the weight of corresponding to the cubature point g according to the chosen cubature rule. Similarly, the surface integrals of the convective and diffusive terms in (2.4) can also be calculated using cubature rules,

$$\sum_{f \in \mathcal{F}(I)} \int \Big(\vec{v}\varphi - \Gamma \nabla \varphi \Big) \cdot d\vec{S}_f \simeq \sum_{f \in \mathcal{F}(I)} \sum_{g \in \mathcal{G}(f)} w_{G_g} \Big(\vec{v}\varphi(\vec{x}_g) - \Gamma \nabla \varphi(\vec{x}_g) \Big) \cdot \vec{S}_f \, , \tag{2.6}$$

where $\mathcal{G}(f)$ is the set of cubature points associated with face f. As such, combining (2.5) and (2.6) into (2.4) yields the semi-discretized convection-diffusion equation

$$\sum_{f \in \mathcal{F}(I)} \sum_{g \in \mathcal{G}(f)} w_{G_g} \Big(\vec{v}\varphi(\vec{x}_g) - \Gamma \nabla \varphi(\vec{x}_g) \Big) \cdot \vec{S}_f = V \sum_{g \in \mathcal{G}(I)} w_{G_g} Q(\vec{x}_g) \, . \tag{2.7}$$

[1] For the present work, the Author defines the terms “quadrature” and “cubature” as the numerical computational of, respectively, a univariate and a multivariate integral, following the convention of Krommer and Ueberhuber [85].

2.1.2 Polynomial Approximations of φ and $\nabla\varphi$

The terms $\varphi(\vec{x}_g)$ and $\nabla\varphi(\vec{x}_g)$ – evaluated at the cubature point g – are approximated using a polynomial centered on face f. Let this regression polynomial be represented in the following form:

$$\begin{aligned}\varphi(\vec{x}_g) = \kappa_0 &+ \kappa_x(x_g - x_f) + \kappa_y(y_g - y_f) + \kappa_z(z_g - z_f) \\ &+ \kappa_{xx}(x_g - x_f)^2 + \kappa_{xy}(x_g - x_f)(y_g - y_f) + \kappa_{xz}(x_g - x_f)(z_g - z_f) \\ &+ \kappa_{yy}(y_g - y_f)^2 + \kappa_{yz}(y_g - y_f)(z_g - z_f) + \kappa_{zz}(z_g - z_f)^2 + \dots \end{aligned} \tag{2.8}$$

In vector form, one may represent this approximation by the product between a coefficient vector $\boldsymbol{\kappa}_f$ and a vector $\mathbf{d}_f(\vec{x}_g)$ of the terms representing the distance between $\vec{x}_g$ and $\vec{x}_f$,

$$\varphi(\vec{x}_g) = \mathbf{d}_f(\vec{x}_g)\,\boldsymbol{\kappa}_f\,. \tag{2.9}$$

However, this still leaves the coefficient vector $\boldsymbol{\kappa}_f$ as unknown. Now, consider a neighboring cell near the same face f. Using the previous expression, the value of φ at this cell's centroid $\vec{x}_{\mathrm{nb}}$ may be written as

$$\varphi(\vec{x}_{\mathrm{nb}}) = \mathbf{d}_f(\vec{x}_{\mathrm{nb}})\,\boldsymbol{\kappa}_f \tag{2.10}$$

Therefore, grouping several these neighboring cells into a stencil $s(f)$ and combining their individual expressions for φ yields the following matrix-vector equation

$$\boldsymbol{\varphi}_{s(f)} = \mathbf{D}_{s(f)}\,\boldsymbol{\kappa}_f \tag{2.11}$$

where $\boldsymbol{\varphi}_{s(f)}$ contains φ_c, $c \in s(f)$ and $\mathbf{D}_{s(f)}$ is a matrix created by concatenating vectors $\mathbf{d}_f(\vec{x}_c)$, $c \in s(f)$. In an expanded matrix form, the previous equation is given by

$$\begin{bmatrix}\varphi_{c_1}\\ \varphi_{c_2}\\ \vdots\\ \varphi_{c_N}\end{bmatrix} = \begin{bmatrix}1 & x_{c_1}-x_f & y_{c_1}-y_f & z_{c_1}-z_f & (x_{c_1}-x_f)^2 & (x_{c_1}-x_f)(y_{c_1}-y_f) & \cdots\\ 1 & x_{c_2}-x_f & y_{c_2}-y_f & z_{c_2}-z_f & (x_{c_2}-x_f)^2 & (x_{c_2}-x_f)(y_{c_2}-y_f) & \cdots\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots\\ 1 & x_{c_N}-x_f & y_{c_N}-y_f & z_{c_N}-z_f & (x_{c_N}-x_f)^2 & (x_{c_N}-x_f)(y_{c_N}-y_f) & \cdots\end{bmatrix}\begin{bmatrix}\kappa_0\\ \kappa_x\\ \kappa_y\\ \vdots\end{bmatrix} \tag{2.12}$$

This allows one to pose the issue of determining $\mathbf{c}_f$ as a weighted least-squares problem expressed as the minimization of a residual function, as given by the following

$$\text{minimize:}\quad r\left(\boldsymbol{\varphi}'_{s(f)}\right) = \left(\boldsymbol{\varphi}_{s(f)} - \boldsymbol{\varphi}'_{s(f)}\right)^{\mathrm{T}} \mathbf{W}_{s(f)} \left(\boldsymbol{\varphi}_{s(f)} - \boldsymbol{\varphi}'_{s(f)}\right)\,. \tag{2.13}$$

In this expression, $\boldsymbol{\varphi}$ and $\boldsymbol{\varphi}'$ are, respectively, the original data set of the generated stencil $s(f)$ and the corresponding obtained solution. Furthermore, $\mathbf{W}_{s(f)}$ is a diagonal matrix containing the weights assigned to each cell in the stencil. The inclusion of weights gives a greater accuracy to the method by allowing cells closer to the target face f to be given a greater influence on the value of $\boldsymbol{\kappa}_f$. The choice of weight functions has been previously studied by Vasconcelos [30] and Diogo [32], each for two different types of stencil algorithms. For the stencil type considered in the current work (see Section 2.2), Diogo recommends $w_c = |\vec{x}_c - \vec{x}_f|^{-6}$, $c \in s(f)$.

Multiplying both sides of (2.11) by $\mathbf{D}_f^T \mathbf{W}_{s(f)}$ and then performing some algebraic manipulation, one obtains an equation for the coefficient vector $\boldsymbol{\kappa}_f$ given by (2.14), where $\mathbf{P}_{s(f)}$ is the pseudo-inverse

matrix (or local matrix).

$$\boldsymbol{\kappa}_f = \mathbf{P}_{s(f)}\boldsymbol{\varphi}_{s(f)}\,, \quad \mathbf{P}_{s(f)} \equiv \left(\mathbf{D}_{s(f)}^T\mathbf{W}_{s(f)}\mathbf{D}_{s(f)}\right)^{-1}\mathbf{D}_{s(f)}^T\mathbf{W}_{s(f)} \tag{2.14}$$

Then, substituting the obtained expression for $\boldsymbol{\kappa}_f$ into (2.9) allows for $\varphi(\vec{x}_g)$ to be calculated from the corresponding stencil values, as shown in (2.15). Similarly, $\nabla\varphi(\vec{x}_g)$ can be calculated from (2.16). In the particular case of regular Cartesian grids, the values in coefficient vector $\mathbf{d}_f(\vec{x}_g)$ depend only on cubature point g, since the distribution of cubature points will be the same for any face; as such, the subscript f has been omitted. Additionally, many of these terms in $\mathbf{d}_g$ will be null; therefore, only a small portion of the matrix multiplication $\mathbf{d}_g\mathbf{P}_f$ needs to be computed. Unfortunately, the computation of $\mathbf{P}_{s(f)}$ itself is expensive, due to the need to compute an inverse matrix.

$$\varphi(\vec{x}_g) = \quad \mathbf{d}_f(\vec{x}_g)\mathbf{P}_{s(f)}\boldsymbol{\varphi}_{s(f)} \tag{2.15}$$

$$\nabla\varphi(\vec{x}_g) = \nabla\mathbf{d}_f(\vec{x}_g)\mathbf{P}_{s(f)}\boldsymbol{\varphi}_{s(f)} \tag{2.16}$$

Substituting these two expressions into (2.7) leads to the final equation used in the method, which will be used to create the linear system $\mathbf{A}\boldsymbol{\varphi}_{s(f)} = \mathbf{b}$,

$$\sum_{f\in\mathcal{F}(I)}\sum_{g\in\mathcal{G}(f)} w_{G_g}\left[\left(\vec{S}_f\cdot\vec{v}\right)\mathbf{d}_f(\vec{x}_g)\mathbf{P}_{s(f)} - \Gamma\left(\vec{S}_f\cdot\nabla\right)\mathbf{d}_f(\vec{x}_g)\mathbf{P}_{s(f)}\right]\boldsymbol{\varphi}_{s(f)} = V\sum_{g\in\mathcal{G}(I)} w_{G_g}Q(\vec{x}_g) \tag{2.17}$$

2.1.3 Simplifications for Cartesian Grids

For the current case of a strictly regular Cartesian grid, the regression polynomial given in (2.8) may be extensively simplified. Consider a normal-tangential coordinate system η-τ_a-τ_b centered on the face centroid, with the normal unit vector $\hat{\eta}$ defined by the face normal, i.e. $\hat{\eta} \equiv \hat{n}_f$. Given the properties of the grid, $\hat{\eta}$ will coincide with one of the original Cartesian coordinate system vectors, and then $\hat{\tau}_a, \hat{\tau}_b$ may be defined to coincide with the remaining vectors according to the right-hand rule. For example, in the case of a northern face, the corresponding tangential coordinate system would be defined as $\hat{\eta} = \hat{y}$, $\hat{\tau}_a = \hat{z}$, and $\hat{\tau}_b = \hat{x}$, as seen Figure 2.1.

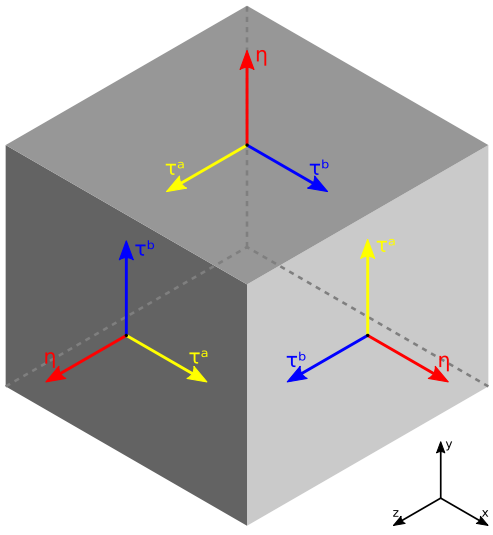


Figure 2.1 Face-centered, normal-tangential coordinate system η-τ_a-τ_b illustrated on selected cell faces in a three-dimensional, Cartesian grid.

As such, the regression polynomial – using a Cartesian coordinate system in (2.8) – is written in the aforementioned normal-tangential coordinate system as

$$\begin{aligned}\varphi(\vec{x}_g) = c_0 &+ c_\eta(\eta_g - \eta_f) + c_{\tau_a}(\tau_{a,g} - \tau_{a,f}) + c_z(\tau_{b,g} - \tau_{b,f}) \\ &+ c_\eta(\eta_g - \eta_f)^2 + c_{\eta\tau_a}(\eta_g - \eta_f)(\tau_{a,g} - \tau_{a,f}) + c_{\eta\tau_b}(\eta_g - \eta_f)(\tau_{b,g} - \tau_{b,f}) \\ &+ c_{\tau_a\tau_a}(y_g - y_f)^2 + c_{\tau_a\tau_b}(\tau_{a,g} - \tau_{a,f})(\tau_{b,g} - \tau_{b,f}) + c_{\tau_b\tau_b}(\tau_{b,g} - \tau_{b,f})^2 + \dots \end{aligned} \tag{2.18}$$

Thus, given that all cubature points for the face flux terms lie on the face itself, it becomes clear that all terms containing $(\eta_g - \eta_f)$ will be null.[2] This drastically reduces the number of required terms in the polynomial. As such, the simplified polynomial is given by

$$\begin{aligned}\varphi(\vec{x}_g) = c_0 &+ c_{\tau_a}(\tau_{a,g} - \tau_{a,f}) + c_z(\tau_{b,g} - \tau_{b,f}) \\ &+ c_{\tau_a\tau_a}(y_g - y_f)^2 + c_{\tau_a\tau_b}(\tau_{a,g} - \tau_{a,f})(\tau_{b,g} - \tau_{b,f}) + c_{\tau_b\tau_b}(\tau_{b,g} - \tau_{b,f})^2 + \dots \end{aligned} \tag{2.19}$$

Note that, even though the resulting polynomial does not include many of the original terms, the regression fit is done considering all terms in $\mathbf{D}_{s(f)}$, since these correspond to the distance between the target face and the stencil points – instead of that between the target face and its corresponding cubature points.

The term $\nabla\varphi(\vec{x}_g)$ can be obtained by calculating the gradient of (2.18). However, upon inspection of (2.6), one notices that the full required term is actually $\nabla\varphi(\vec{x}_g) \cdot \vec{S}_f$. Therefore, for the case of Cartesian grids, this simplifies to $\frac{\partial}{\partial\eta}\varphi(\vec{x}_g)S_f$. Therefore, it suffices to take the derivative of (2.18) with respect to η only, instead of calculating the full gradient. Furthermore, since any resulting terms containing $(\eta_g - \eta_f)$ will be null, one needs to consider only the terms whose original form is $(\eta_g - \eta_f)(\tau_{a,g} - \tau_{a,g})^i(\tau_{b,g} - \tau_{b,f})^j$ where $i, j \geq 0$. The result is then given by

$$\nabla\varphi(\vec{x}_g) \cdot \vec{S}_f = S_f c_\eta + S_f c_{\eta\tau_a}(\tau_{a,g} - \tau_{a,f}) + S_f c_{\eta\tau_b}(\tau_{b,g} - \tau_{b,f}) + \dots \tag{2.20}$$

These simplifications decrease the necessary size of vectors $\mathbf{d}_f$ and $\boldsymbol{\kappa}_f$. In turn, from equation (2.14), this consequently decreases the size of the pseudoinverse matrix $\mathbf{P}_f$. The computation of this matrix is the most time-consuming step in creating the coefficient matrix $\mathbf{A}$ of an FLS scheme, due to the need to calculate the inverse of a square matrix with size corresponding to $\mathbf{d}_f$. Thus, these simplifications yield significant savings in computation time for the considered case of Cartesian grids.

2.2 Stencil Generation

As part of the FLS scheme, a stencil composed of neighboring cells and boundary faces needs to be generated for every face. There were two possible choices for the stencil generation method: using vertex-neighboring cells [30] or using face-neighboring cells [32]. These methods are based on the concept of generating a stencil by repeatedly adding neighboring cells from a previous set. It starts at the target face and then moves outwards. The core difference between them lies in the criteria used to determine the cell neighbors.

Both methods have the same starting steps, which consist of adding to the stencil any cell that

[2] Assuming that an appropriate cubature rule is chosen. However, this is true for the most commonly used cubature rules, including those given in Section 2.4.

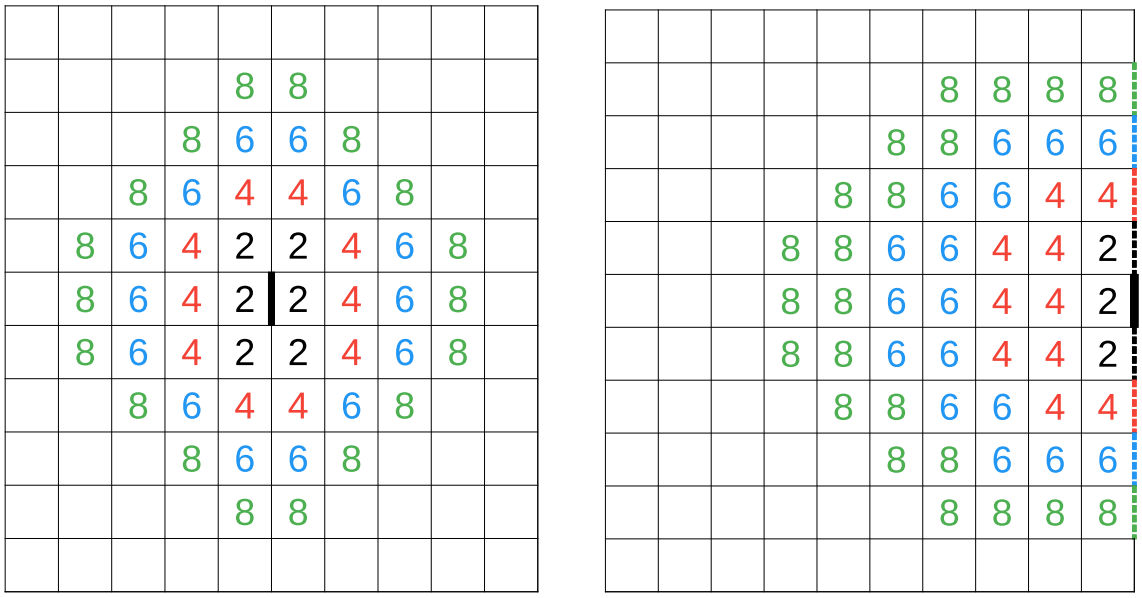
immediately borders the target face. These cells may be categorized as first-level cells and are indicated by a black number "2" in Figure 2.2. In the next set of steps, cells neighboring the recently-added cells are in turn inserted into the stencil. Each method determines these second-level cells differently. The original method developed by Vasconcelos [30] chooses cells that share a vertex with any first-degree neighbor cell, while the method developed by Diogo [32] chooses cells that share a face. Then, this second set of steps is repeated until the stencil dimensions are large enough in the directions of the polynomial, satisfying both

$$n_{s,\eta} \equiv \left\lfloor \frac{\max \eta_s - \min \eta_s}{L_{\text{ref}}} + 1.1 \right\rceil \geq p \,, \tag{2.21}$$

$$n_{s,\tau} \equiv \left\lfloor \frac{\max \tau_s - \min \tau_s}{L_{\text{ref}}} + 1.1 \right\rceil \geq p + 1 \,. \tag{2.22}$$

Here, the notation $\lfloor a \rceil$ represents the rounding of a to the nearest integer, and the $\max, \min$ operations yield the maximum or minimum centroid coordinate (η or τ) of all stencil points.

This stopping criterion is determined by the order p of the chosen FLS method and differs for the normal (η) and tangential (τ) directions. For example, the FLS4 method requires a stencil with at least four cells in the direction normal to the target face and at least five cells in the corresponding tangent direction – which in Figure 2.2a would include all cells marked with either a black "2" or a red "4". Table 2.1 lists the stencil size for an inner face in two- and three-dimensional problems, and the corresponding total stencil size for an inner cell – i.e. the combined size of all face stencils corresponding to an inner cell, without repetitions.



(a) Inner-face stencil **(b)** Boundary-face stencil

Figure 2.2 Stencil generation method using face-neighboring cells, illustrated for the two-dimensional cases of an inner face (a) and a boundary face (b). The target face is shown in thick solid black, while any other boundary faces added to the stencil are shown in dashed.

Furthermore, each time that a cell is added to stencil, the algorithm checks if that cell contains a boundary face. If it does, then that face is also added to the stencil. Taking the previous example as a base, this means that for boundary stencil (i.e. those containing boundary faces) the necessary number of cells is marginally smaller, since the included boundary faces also have an associated φ value and are thus included in the computation of the $\max, \min$ terms in (2.21) and (2.22).

Table 2.1 Individual (face) and total (cell) stencil sizes for two- and three-dimensional Cartesian grids using the face-neighbor algorithm, for non-boundary stencils. The former corresponds to the size of a single, inner-face stencil, while the latter represents the union of all face stencils of an inner-cell.

	2D				3D			
	FLS2	FLS4	FLS6	FLS8	FLS2	FLS4	FLS6	FLS8
Individual (face)	6	16	30	48	18	60	134	248
Total (cell)	9	21	37	57	27	81	171	305

As previously shown by Diogo [32], the stencil generation method using the face-neighbors criteria significantly outperforms its vertex-neighbors counterpart both numerically (as measured by the accuracy of the solution obtained from the resulting numerical method) and computationally (as measured by the program's memory usage and runtime). As such, the current work uses solely the face-neighboring stencil generation method.

2.3 Boundary Treatment

This section describes the treatment given to stencils that contain a boundary face. Since the main focus of this work is to evaluate the parallel performance of the FLS scheme, different boundary treatments will not be deeply studied, given that boundary stencils amount to only $\mathcal{O}\left(N_C^{-1/N_D}\right)$ of total cells, where N_C, N_D are the number of cells and number of dimensions, respectively. Therefore, this work only considers Dirichlet and periodic boundary conditions. Note that Diogo [32] verified that the FLS method also works correctly for Neumann- and Robin-type boundary conditions.

The treatment for Dirichlet-type boundaries requires reorganizing the matrix $\mathbf{D}_f$ given in (2.12). For cells near a boundary, the generated stencil may also include some boundary faces. Organizing the matrix $\mathbf{D}_f$ and vector $\mathbf{\Phi}_{s(f)}$ in a way such that all boundary values are at the lower rows, the boundary version of (2.12) is then given in expanded matrix format by

$$\begin{bmatrix} \varphi_{c_1} \\ \vdots \\ \varphi_{c_N} \\ \hline \varphi_{bf_1} \\ \vdots \\ \varphi_{bf_M} \end{bmatrix} = \left[\begin{array}{ccccccc} 1 & x_{c_1}-x_f & y_{c_1}-y_f & z_{c_1}-z_f & (x_{c_1}-x_f)^2 & (x_{c_1}-x_f)(y_{c_1}-y_f) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & x_{c_N}-x_f & y_{c_N}-y_f & z_{c_N}-z_f & (x_{c_N}-x_f)^2 & (x_{c_N}-x_f)(y_{c_N}-y_f) & \cdots \\ \hline 1 & x_{bf_1}-x_f & y_{bf_1}-y_f & z_{bf_1}-z_f & (x_{bf_1}-x_f)^2 & (x_{bf_1}-x_f)(y_{bf_1}-y_f) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & x_{bf_M}-x_f & y_{bf_M}-y_f & z_{bf_M}-z_f & (x_{bf_M}-x_f)^2 & (x_{bf_M}-x_f)(y_{bf_M}-y_f) & \cdots \end{array}\right] \begin{bmatrix} \kappa_0 \\ \kappa_x \\ \kappa_y \\ \vdots \end{bmatrix} \tag{2.23}$$

where N and M are, respectively, the number of cells and of boundary faces in the stencil. The horizontal lines in (2.23) are merely a visual aid to help show the division of the stencil into cells and boundary faces. As before, the matrix $\mathbf{D}_f$ is then used to calculate the pseudo-inverse matrix $\mathbf{P}_f$.

2.4 Cubature Rules Used

This section discusses the cubature rules used to numerically integrate the source term and cell fluxes, as discussed in Section 2.1.1. More specifically, these rules provide the coordinates $\vec{x}_g$ and weights w_{G_g} for each integration point g used in (2.17). For ease of reference, the cubature rules used

for each of the two- and three-dimensional cases are listed in different subsections.

The aforementioned cubature rules were purposely given in expression form, so that any future modification or testing may be done with any desired precision. With this, the author attempts to avoid the common problem of having insufficiently precise values, which arises when listing only the numerical values for a cubature rule since most of these are taken from older references from when extended precision (e.g. quadruple precision) such as was not such an important aspect. For the current work, the author used the Wolfram|Alpha online engine [86] to evaluate the analytical expressions to double- and quadruple-precision as required.

2.4.1 Two-Dimensional Cases

Face Fluxes

The integration for cell fluxes given by (2.6) is performed over each cell face. In the two-dimensional case, these faces are simply line segments. As such, it is straightforward to calculate the integral using quadrature rules, described by the coordinates and weights shown in Table 2.2.

Table 2.2 Quadrature rules used for the p-degree integration using N_G points over the interval $[-1; 1]$, with the respective coordinate ζ_g and weight w_{G_g} for each integration point g [87].

p	N_G	ζ_g	w_{G_g}
1	1	0	1
3	2	$\pm\sqrt{\frac{1}{3}}$	$\frac{1}{2}$
5	3	0	$\frac{4}{9}$
		$\pm\sqrt{\frac{3}{5}}$	$\frac{5}{18}$
7	4	$\pm\frac{1}{7}\sqrt{21+14\sqrt{1.2}}$	$\frac{1}{72}(18-\sqrt{30})$
		$\pm\frac{1}{7}\sqrt{21-14\sqrt{1.2}}$	$\frac{1}{72}(18+\sqrt{30})$

Since these integration point coordinates are given for the normalized interval $[-1, 1]$, these need to be transformed into the appropriate dimensions. For the current case of Cartesian regular grids, this is done simply by $\vec{x}_g = \vec{x}_C + \frac{1}{2}\zeta_g L_{\text{ref}}\hat{\tau}$, where $\vec{x}_C$ is the position of the cell centroid, $\hat{\tau}$ is a positive unit vector such that $\hat{\tau} \cdot \vec{S}_f = 0$, and L_{ref} is the cell's reference length (i.e. the length of one side).

Source Term

For the calculation of the source term in the two-dimensional case – following the previous works of Vasconcelos [30] and Diogo [32] – each computational cell I is divided into a set of triangles $\mathcal{T}(I)$, as shown in Figure 2.3. Thus, the numerical integration of the source term is done for each triangle $t \in \mathcal{T}(I)$, and then the contributions are summed to obtain the total source term, as shown in (2.24), where $\mathcal{G}_{T_2}$ represents the set of cubature points over a triangle. Note that, for formality, V_t is kept as a volume term given its units, even though it actually simplifies to a surface term times unity, $V_t = zS_t = S_t$.

$$\int_I Q_\varphi \, dV_I = \sum_{t\in\mathcal{T}} \left(\int_t Q_\varphi \, dV_t \right) = \sum_{t\in\mathcal{T}} \left(V_t \sum_{g\in\mathcal{G}_{T_2}} w_{G_g} Q_\varphi(\vec{x}_{g,t}) \right) \tag{2.24}$$

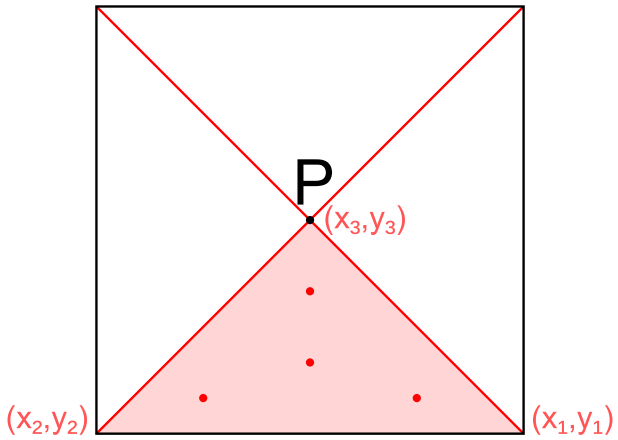


Figure 2.3 Cell division into triangles for source term calculation, with 4th-order cubature points illustrated.

The cubature points for a triangle may be grouped into three types, defined by the repetition of the simplex coordinates, or coefficients, α, β, γ, which affects their multiplicity M – i.e. the number of points associated to each type (see, for example, [88]). Let the three point types be named as $\mathrm{I}, \mathrm{II}, \mathrm{III}$; then,

$$\text{Type I} \quad : \quad \alpha_g = \beta_g = \gamma_g \quad \Rightarrow \quad M_A = \binom{2}{2} = 1 \tag{2.25}$$

$$\text{Type II} \quad : \quad \alpha_g \neq \beta_g = \gamma_g \quad \Rightarrow \quad M_B = \binom{3}{2} = 3 \tag{2.26}$$

$$\text{Type III} \quad : \quad \alpha_g \neq \beta_g \neq \gamma_g \quad \Rightarrow \quad M_C = \binom{4}{2} = 6 \tag{2.27}$$

This influence of repeated coefficients on the point multiplicity is clearer when considering the point coordinate $\vec{x}_g$ in its alternative form $x_g = \alpha_g x_l + \beta_g x_m + \gamma_g x_n$, $l \neq m \neq n$. For example, since all coefficients for Type I all repeated, it is only possible to form one cubature point; conversely, since all coefficients for Type III are unique, it is possible to construct six cubature points. The values of the coefficients α, β, γ used for the two-dimensional source term integration are compiled in Table 2.3 for the relevant orders. Expressions for $p = 1, 3, 5$ were taken from Stroud [89], and values for $p = 7$ from Dunavant [88]. An asterisk besides the N_G column entry denotes that the rule uses the optimal number of points, according to Cools [90].

Table 2.3 Cubature rules used for the p-degree integration using N_G points of a triangle region defined by vertices $(0,0)$-$(1,0)$-$(0,1)$, with the corresponding simplex coordinates α_g-β_g-γ_g, weight w_{G_g}, and multiplicity M for each integration point g [88, 89, 90].

p	N_G	M	α_g	β_g	γ_g	w_{G_g}
1	1*	1	$1/3$	$1/3$	$1/3$	1
3	4*	1	$1/3$	$1/3$	$1/3$	$-9/16$
		3	$3/5$	$1/5$	$1/5$	$-25/48$
5	7*	1	$1/3$	$1/3$	$1/3$	$9/40$
		3	$(9+2\sqrt{15})/21$	$(6-\sqrt{15})/21$	$(6-\sqrt{15})/21$	$(155-\sqrt{15})/1200$
		3	$(9-2\sqrt{15})/21$	$(6+\sqrt{15})/21$	$(6+\sqrt{15})/21$	$(155+\sqrt{15})/1200$
7	13	1	0.333333333333333	0.333333333333333	0.333333333333333	−0.149570044467682
		3	0.479308067841920	0.260345966079040	0.260345966079040	0.175615257433208
		3	0.869739794195568	0.065130102902216	0.065130102902216	0.053347235608838
		6	0.048690315425316	0.312865496004874	0.638444188569810	0.077113760890257

The final coordinates $\vec{x}_{g,t}$ of the cubature point g of triangle t may then be written as the transforma-

tion of the simplex coordinates $\boldsymbol{\kappa}_g$ given by (2.28), where $x_{\nu,t}, y_{\nu,t}$ represent the coordinates of vertex ν of triangle t, as shown by Diogo [32].

$$\vec{x}_{g,t} = \mathbf{X}_t \boldsymbol{\kappa}_g = \begin{bmatrix} x_{1,t} & x_{2,t} & x_{3,t} \\ y_{1,t} & y_{2,t} & y_{3,t} \end{bmatrix} \begin{bmatrix} \alpha_g \\ \beta_g \\ \gamma_g \end{bmatrix} \tag{2.28}$$

Since the present work only considers regular Cartesian grids – instead of the more general unstructured grids used by Diogo [32] – some simplifications can be made, allowing for a larger portion of the calculations to be hard-coded into the algorithm. First, since all cells are squares, all divisions will be isosceles triangles of base L_{ref} and sides $\sqrt{2}L_{\text{ref}}$.

Fixing one vertex of the triangle to always be the cell centroid (say, vertex 3, as in Figure 2.3), then the other two vertices may be defined to be consecutive cell vertices in the clockwise direction. As such, defining the east-facing division to be triangle 1, the coordinate $\vec{x}_{\nu,t}$ of vertex ν for any triangle t may be written according to (2.29) and (2.30), where $\xi_5 \equiv \xi_1$, $\mu_5 \equiv \mu_1$.

$$x_{1,t} = x_{3,t} + \xi_t \,, \qquad x_{2,t} = x_{3,t} + \xi_{t+1} \,, \qquad \text{where} \quad \boldsymbol{\xi} = \tfrac{1}{2} L_{\text{ref}} \begin{bmatrix} -1 & -1 & 1 & 1 \end{bmatrix} \tag{2.29}$$

$$y_{1,t} = y_{3,t} + \mu_t \,, \qquad y_{2,t} = y_{3,t} + \mu_{t+1} \,, \qquad \text{where} \quad \boldsymbol{\mu} = \tfrac{1}{2} L_{\text{ref}} \begin{bmatrix} -1 & 1 & 1 & -1 \end{bmatrix} \tag{2.30}$$

Using these expressions, (2.28) may be rewritten as

$$\vec{x}_{g,t} = \begin{bmatrix} x_{3,t} + \xi_t & x_{3,t} + \xi_{t+1} & x_{3,t} \\ y_{3,t} + \mu_t & y_{3,t} + \mu_{t+1} & y_{3,t} \end{bmatrix} \begin{bmatrix} \alpha_g \\ \beta_g \\ \gamma_g \end{bmatrix} = \begin{bmatrix} \alpha_g \xi_t + \beta_g \xi_{t+1} + (\alpha_g + \beta_g + \gamma_g) x_{3,t} \\ \alpha_g \mu_t + \beta_g \mu_{t+1} + (\alpha_g + \beta_g + \gamma_g) y_{3,t} \end{bmatrix} \tag{2.31}$$

Therefore, remembering that $\alpha + \beta + \gamma \equiv 1$ and $\vec{x}_{3,t} \equiv \vec{x}_C$, the coordinates for each integration point type may then be written purely in terms of the cell centroid, L_{ref}, and the cubature coefficients α-β-γ according to (2.32)-(2.41), where $\vec{\lambda}_t = \xi_t \hat{\imath} + \mu_t \hat{\jmath}$ and $\vec{x}_C$ is the position of the cell centroid.

$$\text{Type I}: \quad \vec{x}_{g,t} = \vec{x}_C + \alpha_g \vec{\lambda}_t + \alpha_g \vec{\lambda}_{t+1} \tag{2.32}$$

$$\text{Type II}: \begin{cases} \vec{x}_{g,t} = \vec{x}_C + \alpha_g \vec{\lambda}_t + \beta_g \vec{\lambda}_{t+1} & (2.33) \\ \vec{x}_{g,t} = \vec{x}_C + \beta_g \vec{\lambda}_t + \alpha_g \vec{\lambda}_{t+1} & (2.34) \\ \vec{x}_{g,t} = \vec{x}_C + \beta_g \vec{\lambda}_t + \beta_g \vec{\lambda}_{t+1} & (2.35) \end{cases}$$

$$\text{Type III}: \begin{cases} \vec{x}_{g,t} = \vec{x}_C + \alpha_g \vec{\lambda}_t + \beta_g \vec{\lambda}_{t+1} & (2.36) \\ \vec{x}_{g,t} = \vec{x}_C + \alpha_g \vec{\lambda}_t + \gamma_g \vec{\lambda}_{t+1} & (2.37) \\ \vec{x}_{g,t} = \vec{x}_C + \beta_g \vec{\lambda}_t + \alpha_g \vec{\lambda}_{t+1} & (2.38) \\ \vec{x}_{g,t} = \vec{x}_C + \beta_g \vec{\lambda}_t + \gamma_g \vec{\lambda}_{t+1} & (2.39) \\ \vec{x}_{g,t} = \vec{x}_C + \gamma_g \vec{\lambda}_t + \alpha_g \vec{\lambda}_{t+1} & (2.40) \\ \vec{x}_{g,t} = \vec{x}_C + \gamma_g \vec{\lambda}_t + \beta_g \vec{\lambda}_{t+1} & (2.41) \end{cases}$$

2.4.2 Three-Dimensional Cases

Since the code for the three-dimensional case was specifically developed for regular Cartesian grids instead of unstructured grids from the previous 2D works, it uses a simpler set of cubature rules – especially since the volume integral is calculated directly instead of considering cell subdivisions. The criteria for choosing the given rules was that their weights are positive, that they use the least possible number of points, and that their points lie preferably inside the region (or on the boundary, otherwise). In the following tables listing cubature rules, an asterisk besides the N_G column entry denotes that the rule uses the optimal (i.e. minimum possible) number of points, according to Cools [90].

Furthermore, to present the following tables in a more compact format, these use the notation defined by Stroud [89] of a symmetric $(\cdots)_S$ and fully-symmetric $(\cdots)_{FS}$ set of points:

$$(\chi_1, \ldots, \chi_m)_S \ni (\chi_{p_1}, \ldots, \chi_{p_m}), \qquad \forall p_1, \ldots, p_m \in \{1, \ldots, m\} : p_1 \neq \ldots \neq p_m \tag{2.42}$$

$$(\chi_1, \ldots, \chi_m)_{FS} \ni (\pm\chi_{p_1}, \ldots, \pm\chi_{p_m}), \qquad \forall p_1, \ldots, p_m \in \{1, \ldots, m\} : p_1 \neq \ldots \neq p_m \tag{2.43}$$

where all $\pm$ are independent. Therefore, a symmetric set of points can be understood as the set of all permutations of the coordinates of a reference point. The fully-symmetric set has a similar construction, but also includes sign variations for all coordinates. As such, a symmetric set as described by (2.42) has a total of $m!$ points, and a fully-symmetric set as given by (2.43) has $2^m(m!)$ points. Naturally, these sets will be smaller if there is a repeated value in either set or a zero in the fully-symmetric set.

Face Fluxes

For the face flux terms, the integration given by (2.6) is performed over a square region, using the cubature rules described in Table 2.4. The expressions were taken from Stroud [89], with a correction for $p = 3$ from Cools [90]. Once again, the integration points must be transformed in the appropriate dimensions since they are presented for a square with vertices $(1, 1)_{FS}$. This is achieved by $\vec{x}_g = \vec{x}_C + \frac{1}{2}L_{\text{ref}}(\xi_g \hat{\tau}_a + \mu_g \hat{\tau}_b)$, where $\hat{\tau}_a, \hat{\tau}_b$ is mapped to $\hat{\imath}, \hat{\jmath}, \hat{k}$ as described in Section 2.1.1.

Table 2.4 Cubature rules used for the p-degree integration using N_G points over the square region defined by vertices $(1, 1)_{FS}$, with the respective coordinates (ξ_g, μ_g) and weights w_{G_g} for each integration point g [89, 90]. The entries a, b are auxiliary values.

p	N_G	(ξ_g, μ_g)	w_{G_g}	a	b
1	1*	$(0, 0)$	1	–	–
3	4*	$(a, a)_{FS}$	$\frac{1}{4}$	$\sqrt{\frac{1}{3}}$	–
5	7*	$(0, 0)$	$\frac{2}{7}$	–	–
		$(0, \pm b)$	$\frac{5}{63}$	–	$\sqrt{\frac{14}{15}}$
		$(\pm a, \pm b)$	$\frac{5}{36}$	$\sqrt{\frac{3}{5}}$	$\sqrt{\frac{1}{3}}$
7	12*	$(a, 0)_{FS}$	$\frac{49}{810}$	$\sqrt{\frac{6}{7}}$	–
		$(a, a)_{FS}$	$\frac{178981+2769\sqrt{583}}{1888920}$	$\sqrt{\frac{114-3\sqrt{583}}{287}}$	–
		$(a, a)_{FS}$	$\frac{178981-2769\sqrt{583}}{1888920}$	$\sqrt{\frac{114+3\sqrt{583}}{287}}$	–

Source Term

The source term integration is done directly over the cube region defined by the cell itself. For that, the code uses the cubature rules given in Table 2.5 for a cubic region with vertices $(1,1,1)_{FS}$. The expressions for $p = 1, 3, 7$ were adapted from Stroud [89], and for $p = 5$ from Peterson [91]. The coordinate transformation from the normalized dimensions (ξ_g, μ_g, ζ_g) to grid-appropriate dimensions is given by $\vec{x}_g = \vec{x}_C + \frac{1}{2} L_{\text{ref}} (\xi_g \hat{\imath} + \mu_g \hat{\jmath} + \zeta_g \hat{k})$.

Table 2.5 Cubature rules used for the p-degree integration using N_G points over the cubic region defined by vertices $(1,1,1)_{FS}$, with the respective coordinates (ξ_g, μ_g, ζ_g) and weights w_{G_g} for each integration point g [89, 90, 91]. The entries a, b are auxiliary values, as is $c \equiv 71440 + 6802\sqrt{19}$.

p	N_G	(ξ_g, μ_g, ζ_g)	w_{G_g}	a	b
1	1*	$(0,0,0)$	1	–	–
3	6*	$(1,0,0)_{FS}$	$\frac{1}{6}$	–	–
5	13*	$(0,0,0)$	$\frac{4}{19}$	–	–
		$\pm(a,b,b)_S$	$\frac{5}{912}\left(12+\sqrt{\frac{229-50\sqrt{19}}{61}}\right)$	$\sqrt{\frac{1919-148\sqrt{19}+4\sqrt{c}}{3285}}$	$-\sqrt{\frac{1121+74\sqrt{19}-2\sqrt{c}}{3285}}$
		$\pm(a,a,b)_S$	$\frac{5}{912}\left(12-\sqrt{\frac{229-50\sqrt{19}}{61}}\right)$	$\sqrt{\frac{1919-148\sqrt{19}-4\sqrt{c}}{3285}}$	$\sqrt{\frac{1121+74\sqrt{19}+2\sqrt{c}}{3285}}$
7	34	$(a,0,0)_{FS}$	$\frac{1078}{29160}$	$\sqrt{\frac{6}{7}}$	–
		$(a,a,0)_{FS}$	$\frac{343}{29160}$	$\sqrt{\frac{6}{7}}$	–
		$(a,a,a)_{FS}$	$\frac{217107+4145\sqrt{238}}{5452920}$	$\sqrt{\frac{960-3\sqrt{28798}}{2726}}$	–
		$(a,a,a)_{FS}$	$\frac{217107-4145\sqrt{238}}{5452920}$	$\sqrt{\frac{960+3\sqrt{28798}}{2726}}$	–

2.5 Domain Decomposition

As part of their previous works, Vasconcelos [30] and then Diogo [32] developed Matlab codes in which they implemented their contributions to the FLS scheme. It is noteworthy to mention that the Matlab language was chosen for its comparatively simple syntax and straightforward implementation of mathematical operations, especially those regarding matrices and linear algebra. However, Matlab – being an interpreted language – is significantly slower than compiled languages, such as C and Fortran.[3] The Author determined that a Matlab-implemented, three-dimensional FLS scheme would be prohibitively expensive in terms of runtime, besides having limited capability for parallel testing. Therefore, the author chose to implement the three-dimensional schemes using the C language. Furthermore, the author also decided to develop a two-dimensional code in C, as an intermediate step from the original two-dimensional code in Matlab. As such, the author used the previously developed Matlab code as a basis for the novel C code. See Appendix A for more information on the past Matlab and the new C codes.

To cover for the lack of native linear algebra routines in the C language, the author used the Portable, Extensible Toolkit for Scientific Computation (PETSc) library, which provides high-level routines and data structures for matrix manipulation, linear solvers, data and grid management, among others [58, 59]. PETSc uses a distributed-memory approach for parallel programming, with the Message Passing

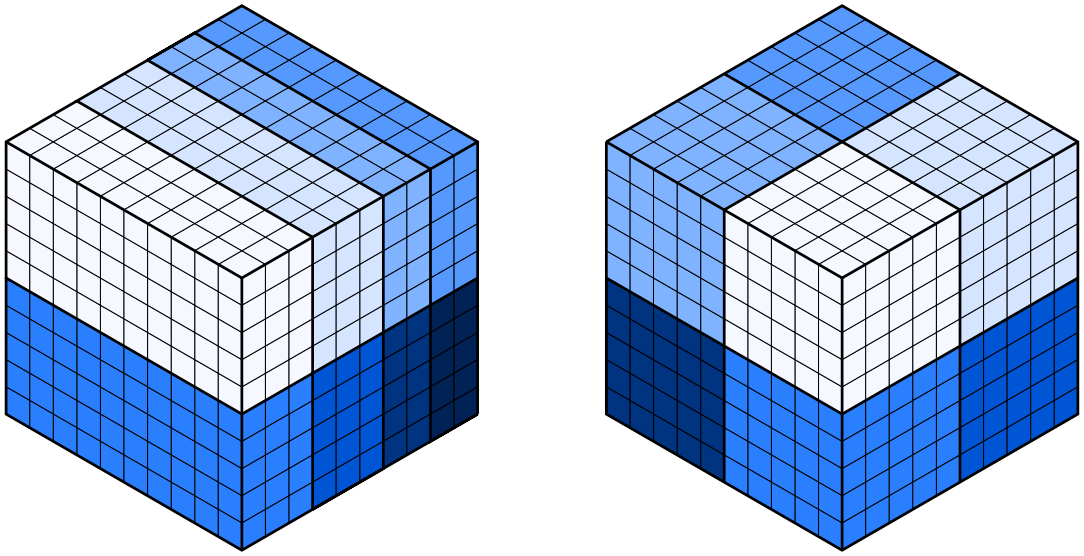
[3] See Ning [92] for an informal discussion on some different scientific programming languages.

Interface (MPI) model employed for communication between processors. Unless otherwise explicitly stated, all PETSc objects are inherently parallel and are distributed among processors based on a specified MPI communicator.

PETSc provides various classes for grid and data management, for various mesh types. Given the focus of the current work, the Author chose to use the `DMDA` class, which provides abstractions for topology and geometry of Cartesian meshes and manages the required communication. The creation of such a class for three-dimensional grids is handled by the routine `DMDACreate3d()`. This routine allows the programmer to specify the number of processors (`m`, `n`, `p`) along the dimensions of the array domain such that $N_p = \mathtt{m} \times \mathtt{n} \times \mathtt{p}$, as shown in Figure 2.4. However, any or all of these may instead be calculated by PETSc if the programmer uses the macro `PETSC_DECIDE`. This allows for a fixed type of domain decomposition regardless of the total number of processors, without the need for hard coding the distribution. For example, to ensure a pencil- or slab-type decomposition where one dimension is not decomposed, the programmer can specify the number of processors in just one direction and pass `PETSC_DECIDE` for the other two.

In the current work, the Author used the PETSc default for domain decomposition, by passing `PETSC_DECIDE` for all directions. Since the number of cells in each direction is equal for regular Cartesian grids, PETSc attempts to divide the total number of processors as evenly as possible among all three directions. This results in a checkerboard-like decomposition. If a distribution is not possible given the provided combination of number of cells and number of processors (e.g. $N_C = 10^3$, $N_p = 11$), an error is called indicating that the partition in a direction is too thin.

Overall, the routine `DMDACreate3d()` allows for considerable flexibility in the creation and parallel management of a Cartesian grid. Since the global dimensions in each direction are required parameters, the code is already prepared to process non-uniform Cartesian grids. Besides the aforementioned parameters, there are a few more whose influence could be the focus of future studies. Namely, there is the option of specifying the number of cells for each processor along any or all directions. This allows for an unequal distribution if desired – e.g. having groups with only inner cells be larger than those containing boundary cells. As such, further study regarding the influence of domain decomposition on



(a) `m = 4, n = 1, p = 2` **(b)** `m = n = p = PETSC_DECIDE`

Figure 2.4 Illustration of two examples of domain decomposition for a mesh of $N_C = 10^3$ among $N_p = 8$ processors. For the current work, only the default decomposition with `PETSC_DECIDE` was used.

the parallel performance of the FLS schemes is left for future work.

2.6 Numerical and Computational Metrics

This section describes the measurement methods used to quantify the numerical and computational performance of the developed code.

2.6.1 Numerical Accuracy

Numerical Error Norms

A numerical method's accuracy can be measured by the numerical error vector $\boldsymbol{\varepsilon}$ defined in (2.44), where $\boldsymbol{\varphi}, \boldsymbol{\varphi}'$ are the analytical and numerical solution vectors, respectively.

$$\boldsymbol{\varepsilon} = \boldsymbol{\varphi} - \boldsymbol{\varphi}' \tag{2.44}$$

However, rather than using a vector, it is often convenient and sufficient to condense this measurement into a single real value using norms [93]. This allows for a global overview of the numerical error that is of a more practical use to the objective of the current work. In general, the weighted p-norm $\|\boldsymbol{a}\|_{p,w}$ of an n-length vector $\boldsymbol{a}$ is given by

$$\|\boldsymbol{a}\|_{p,w} = \left(\sum_{i=1}^{n} |a_i|^p w_i \right)^{1/p} . \tag{2.45}$$

The present work uses the weighted 1-norm (2.46) and the infinity-norm (2.47). For the considered cases, the latter is weighted by the inverse of the domain size N_C (i.e. number of cells). These error norms can also be interpreted as the maximum and mean error, respectively. The final formulas are given by

$$\|\boldsymbol{\varepsilon}\|_1 = \frac{1}{N_C} \sum_{i=1}^{N_C} |\varepsilon_i| \tag{2.46}$$

$$\|\boldsymbol{\varepsilon}\|_\infty = \max_i |\varepsilon_i| , \tag{2.47}$$

Convergence Order

The convergence order is defined as the rate in which the numerical error decreases in relation to the grid refinement, measured by reference length L_{ref}. Its measurement helps to determine the correctness of the computational implementation of a numerical scheme (i.e. code verification). The convergence order of each numerical scheme was measured using both the infinity-norm and the weighted 1-norm, resulting in the two convergence order measurements $\mathcal{O}_\infty$ and $\mathcal{O}_1$, respectively. These are given by the

following expressions,

$$\mathcal{O}_1 = \frac{\log ||\boldsymbol{\varepsilon}||_{1,a} - \log ||\boldsymbol{\varepsilon}||_{1,b}}{\log L_{\mathrm{ref},a} - \log L_{\mathrm{ref},b}} , \tag{2.48}$$

$$\mathcal{O}_\infty = \frac{\log ||\boldsymbol{\varepsilon}||_{\infty,a} - \log ||\boldsymbol{\varepsilon}||_{\infty,b}}{\log L_{\mathrm{ref},a} - \log L_{\mathrm{ref},b}} \tag{2.49}$$

where the subscripts a, b represent values corresponding to meshes with different levels of refinement.

2.6.2 Parallel Computational Performance

Speedup

The main reason of using a parallel code is to solve a problem faster than when compared to its serial counterpart [94]. As such, one of the main metrics used to evaluate the performance of a parallel code is its speedup ψ, which represents the ratio between serial execution time and parallel execution time [95]. Denoting $T(N_C, N_p)$ as the runtime of an N_C-sized problem executed across N_p processors, the speedup may be defined as

$$\psi(N_C, N_p) \equiv \frac{T_s(N_C)}{T(N_C, N_p)} , \tag{2.50}$$

where $T_s(N_C)$ is the runtime of the best *purely serial* code. Although this is the most correct way to measure speedup [6, 95], the measurement of T_s would entail the development of yet another complete code, since the backbone of the developed code – PETSc – is itself an intrinsically parallel framework, meaning that it is not optimized for a purely serial execution. As such, the current work uses an alternative measurement – the algorithmic speedup $\bar{\psi}$ – as given by

$$\bar{\psi}(N_C, N_p) \equiv \frac{T(N_C, 1)}{T(N_C, N_p)} . \tag{2.51}$$

Strong-Scaling Efficiency

Consider a fixed problem size of N_C, and denote $f_s = f_s(N_C)$ as the fraction of operations that cannot be parallelized and must thus be executed sequentially. The speedup obtained by using N_p processors to solve the problem must then satisfy

$$\bar{\psi}(N_C, N_p) \leq \frac{1}{f_s + (1 - f_s)/N_p} . \tag{2.52}$$

This relation, called Amdahl's Law, provides an upper bound on speedup for each problem case. As such, by taking the limit as N_p approaches infinity, one finds that the speedup of a fixed-size problem is ultimately limited by the fraction of serial operations in the code, a behavior first noticed by Amdahl [96]. Thus, one may define a metric called the strong-scaling efficiency η^{st} and given by

$$\eta^{st}(N_C, N_p) \equiv \frac{\bar{\psi}(N_C, N_p)}{N_p} = \frac{T(N_C, 1)}{N_p T(N_C, N_p)} , \tag{2.53}$$

which indicates, for a fixed problem size, how well the computation uses the available parallel resources. For example, for an ideal parallel computation (i.e. $\eta^{st} = 1$) on four processors, the corresponding

speedup should be $\psi = 4$. However, as shown in Amdahl's Law, the speedup is limited by the fraction of serial operations in the code, and thus the strong-scaling efficiency will obey the relation

$$\eta^{st}(N_C, N_p) \leq \frac{1}{1 + f_s(N_p - 1)} \,. \tag{2.54}$$

Weak-Scaling Efficiency

As previously explained, Amdahl's Law provides an upper limit to a computation's strong-scaling efficiency, which considers scaling for a fixed problem size. However, as noted by Gustafson [97], this implicitly assumes that the problem size and the number of processors used are independent. In practice, this is almost never happens, but instead the problem size scales with the number of processors. For example, when there are more available processors, one usually solves a larger-sized problem to use the facilities at hand; conversely, if the problem size is relatively small, it would be considered wasteful to use many processors.

Following this observation, Gustafson-Barsis's Law states that upper limit of the achievable speedup is given by the relation

$$\bar{\psi}(N_C, N_p) \leq N_p + (1 - N_p)s \,, \tag{2.55}$$

where $s = s(N_C, N_p)$ is the fraction of total runtime that is spent in the serial code. Note that, since s is based on total runtime, it is a function of both N_C and N_p – unlike f_s, which only depends on N_C.

As explained by Quinn [94], the underlying difference between Amdahl's and Gustafson-Barsis's Laws is that they approach the idea of speedup in opposite manners. While Amdahl's Law takes a serial computation and predicts the corresponding runtime if that computation were to be run in parallel on multiple processors, Gustafson-Barsis's Law considers the reverse – given a parallel computation, it estimates the runtime if it were to be run serially on a single processor. Thus, it is useful to define a new efficiency metric – the weak-scaling efficiency η^{wk} given by

$$\eta^{wk}(wl, N_p) = \frac{T(wl, 1)}{T(wl, N_p)} \,. \tag{2.56}$$

The key difference between strong- and weak-scaling efficiency is that, while the former considers scaling with a fixed problem size, the latter considers scaling with a fixed problem size per processor – i.e. a fixed workload, defined by $wl = N_C/N_p$.

2.6.3 On Floating-Point Arithmetic

Floating-Point Representation

In computers, numbers and other kinds of information are stored as a sequence of binary digits (bits), each of which representing a single logical state. Since the 1960s, the preferred manner of storing non-integer numbers has been the floating-point representation, which is done considering its binary representation in scientific notation [98]. The corresponding standard for binary floating-point arithmetic was developed by the Institute for Electrical and Electronics Engineers (IEEE) in 1985, with its current version having been published in 2019 [99].

The floating-point representation using base β stores a number given by $a = (-1)^s c\beta^e$ as three distinct components: the sign s, the significand c, and the exponent e. The IEEE standard uses a normalized representation, meaning that for any nonzero number the leading digit of the significand is always one and is not explicitly stored. The current version provides three basic formats for binary floating-point representation: single-precision, double-precision, and quadruple-precision. These consist of, respectively, 32-bit, 64-bit, and 128-bit words. Table 2.6 summarizes the characteristics of these three basic formats, including the allocation of bits to the different components of the representation.

Table 2.6 IEEE basic representation formats for binary floating-point arithmetic and their bit allocation [99].

Representation name		Bit allocation		
Common	IEEE	Sign	Exponent	Significand
Single precision	binary32	1	8	23
Double precision	binary64	1	11	52
Quadruple precision	binary128	1	15	112

Round-off Errors

The choice of floating-point format ultimately comes down to a balance between the desired accuracy of results and an acceptable computing cost. Using more bits for a representation comes at the cost of increased computation time and required memory, but allows for a greater precision in arithmetic operations stemming from the increased number of significant digits. In the context of scientific computing, this is important to delay the accumulation of round-off errors.

Round-off errors are a type of computing error defined as the difference between a true value and the corresponding value stored in a computer [100]. This type of errors may arise in two different manners, of which the most straightforward is representation error. This error appears when trying to store a number that cannot be exactly represented by the chosen floating-point format [101]. As such, the representation has to be approximated into the finite precision offered by the format, therefore incurring a rounding error. For example, the decimal number 0.1 has an infinite repeating representation in binary and cannot be perfectly represented. Using single precision, this representation is given by

$$0.1_{10} \xrightarrow{\text{binary32}} 1.10011001100110011001101_2 \times 2^{-4} = 0.100000001490116119384765625_{10} \tag{2.57}$$

where the notation 0.1_{10} indicates that 0.1 is represented using base 10. The single-precision estimate can only be accurate up to $2^{-23} \approx 10^{-7}$, since it uses 23 bits for the significand. The other, less common case of representation error is when the desired number is out of the allowable range of the given format, which is related to the maximum and minimum possible exponent values.

The other origin for round-off errors is in floating-point arithmetic [101]. According to the IEEE standard, the results from elementary arithmetic operations must be exactly rounded. This means that the operation itself must be exact, but then the result has to be rounded to fit in the representation format. Continuing from the previous example, the operation $0.1 + 2$ in single precision yields

$$0.1_{10} \oplus 2_{10} \xrightarrow{\text{binary32}} 1.00001100110011001100110_2 \times 2^1 = 2.099999904632568359375_{10} \tag{2.58}$$

where $\oplus$ is the binary addition operator. Even though 2 is an exactly representable number (i.e. machine number), the operation induced an additional rounding error besides the representation error from 0.1.

To make matters worse, subtracting 2 from the result does not yield the original approximation for 0.1, but instead retains the accumulated error,

$$(0.1_{10} \oplus 2_{10}) \ominus 2_{10} \xrightarrow{\text{binary32}} 1.10011001100110011000000_2 \times 2^{-4} = 0.099999904632568359375_{10} \quad (2.59)$$

where $\ominus$ is the binary subtraction operator. This means that the rounding error from floating-point arithmetic accumulates with operations, up to a point where all digits in the result are meaningless. Note that a similar behavior occurs for multiplication and division operations.

Consider an expansion of the previous example, where first the addition operation is done n times, then the subtraction operation is also done n times. Table 2.7 shows how the results evolves with increasing values of n. It is clear that, as the number of operations increases, the result becomes progressively more contamined by round-off errors.

Table 2.7 Result of addition/subtraction example performed using single precision for different values of n. The example may be given by the pseudocode: `f = 0.1; do {f += 2} n times; do {f -= 2} n times.`

n	Binary significand	Decimal result	Relative error
1	1.10011001100110011000000	0.099999904632568359375	9.54 E-7
2	1.10011001100110100000000	0.1000003814697265625	3.81 E-6
4	1.10011001100110000000000	0.09999847412109375	1.53 E-5
6	1.10011001101000000000000	0.100006103515625	6.10 E-5
8	1.10011001100000000000000	0.0999755859375	2.44 E-4
10	1.10011010000000000000000	0.10009765625	9.77 E-4
12	1.10011000000000000000000	0.099609375	3.91 E-3
14	1.10100000000000000000000	0.1015625	1.56 E-2
16	1.10000000000000000000000	0.09375	6.25 E-2
18	1.00000000000000000000000	0.125	2.50 E-1
20	zero	0.0	1.00 E-0

Choosing a Precision Format: double *vs.* quadruple

As illustrated in the previous example, the numerical result obtained from a sequence of interdependent calculations may be contaminated by round-off errors. There are several strategies to mitigate this effect, including solutions implemented on hardware and others introduced by the compiler [101]. Additionally, a user or programmer of a scientific computation code should strive to avoid unstable algorithms and ill-conditioned problems, since these exacerbate the accumulation of round-off errors [101, 102].

Another solution strategy for this issue is to use a floating-point format with higher precision, i.e. more digits in the significand. While the rate at which the digits in the significand are contaminated remains the same, having more digits overall means that, for the same amount of operations, a relatively smaller portion of the significand contains round-off errors. Additionally, since these contaminated digits are further right in the mantissa, they also represent a smaller absolute value. Repeating the previous example with double- and quadruple-precision formats illustrates this benefit, as shown in Table 2.8, which also lists the increased cost in computation time.

Quadruple precision offers a much more accurate result and is, therefore, an interesting option when determining the numerical accuracy of the FLS schemes. It can also be used in one-off tests to verify if a result obtained using double precision has been contaminated by round-off errors. On the other hand, double precision presents a good cost-benefit when accuracy is not the sole objective and is thus the better alternative when analyzing computational performance. As a rule of thumb for the remainder of

Table 2.8 Relative errors and elapsed time of addition/subtraction example performed using single, double, and quadruple precision for different values of n.

n	Relative Error			Elapsed Time [s]		
	binary32	binary64	binary128	binary32	binary64	binary128
1	9.54 E-7	8.88 E-16	7.70 E-34	-	-	-
10	9.77 E-4	9.09 E-13	7.89 E-31	7.0 E-6	6.0 E-6	2.6 E-5
20	1.00 E-0	9.31 E-10	8.08 E-28	5.4 E-3	5.4 E-3	2.6 E-2

this work, one should assume that any result shown has been obtained with double precision, unless otherwise stated. As a final note, the reason the times for single and double precision are equivalent is due to hardware and compiler instructions, and thus out of scope for this discussion.

Chapter 3

Results and Discussion

This chapter presents the performance analysis of the face least-squares (FLS) schemes. First, their computational implementations in both two- and three-dimensional cases are verified in Section 3.1 by studying each method's numerical accuracy. Section 3.2 consists of studies relating to alternative implementations of the regression polynomial, expanding on previous studies done by Diogo [32]. Then, Section 3.3 shows the selection procedure for the preconditioned Krylov method used in the parallel efficiency studies. Finally, Sections 3.4 and 3.5 show the parallel performance and computational cost of the 3D implementations of the FLS schemes.

3.1 Numerical Errors and Convergence Orders

To verify the implementation of the FLS methods, the author performed a series of tests to measure each method's accuracy and convergence order. The manufactured solutions used are given by

$$\varphi_{\mathrm{2D}}(\vec{x}/3\pi) = \sin x \sin y \,, \qquad \varphi_{\mathrm{3D}}(\vec{x}/3\pi) = \sin x \sin y \sin z \,. \tag{3.1}$$

Figure 3.1 shows the surface plot for the solution used in the 2D test cases.These functions were chosen because they allow for the simultaneous verification of several terms, although another C^∞ function (e.g. an exponential) could have fit the same purpose. However, Diogo [32] noted when testing the FLS schemes that contamination by round-off errors appear earlier on these sinusoidal functions than on an exponential. As such, the solutions (3.1) provide an interesting opportunity to study the influence of floating-point precision on the resulting numerical accuracy.

Since the purpose of these tests was to verify the novel implementation in C using the Portable, Extensible Toolkit for Scientific Computation (PETSc) – and not the FLS method itself – the only boundary conditions considered were the Dirichlet type, as previously noted in Section 2.3. The error norms and convergence orders were calculated for all FLS schemes in several grid sizes and also for a second-order central differences (CD2) as a classical reference. For the remainder of this section, the notation used to identify an n-order FLS method will be FLSn.

The error norms $\|\boldsymbol{\varepsilon}\|_1$ and $\|\boldsymbol{\varepsilon}\|_\infty$ for each case are plotted as functions of the cell's reference length $L_{\mathrm{ref}} \equiv (N_C)^{1/N_D}$, where N_C is the number of cells in the mesh and N_D is the number of spatial dimensions. Furthermore, these are listed in tabular format, along with the respective convergence orders $\mathcal{O}_1$ and $\mathcal{O}_\infty$, in Appendix B.

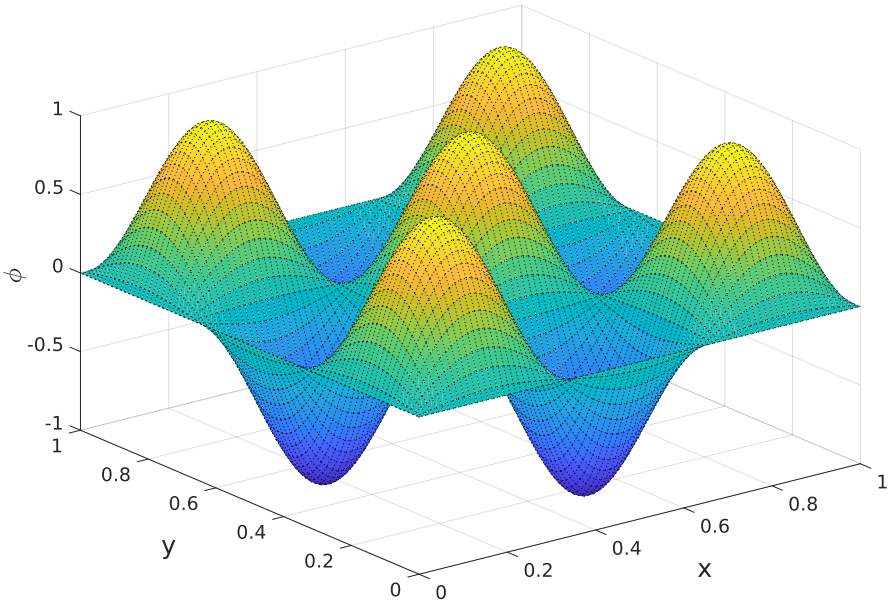


Figure 3.1 Manufactured solution (3.1) used for the 2D verification studies.

3.1.1 Two-Dimensional Case

To obtain more relevant value for the finer grid sizes, the test was run in 128-bit quadruple precision. This allowed for a delayed appearance of round-off errors, albeit with a much higher computational cost; such decision presented an acceptable trade-off since the test's objective was to confirm numerical accuracy and not computational performance. The resulting system of equations was solved with the biconjugate gradient stabilized method (BiCGSTAB) preconditioned using block-Jacobi (see Section 3.3 for more on solvers and preconditioners). The evolutions of the error norms with respect to the reference length L_{ref} for the two-dimensional case are plotted in Figure 3.2.

In terms of convergence order, the results clearly show that all FLS schemes exhibit their expected asymptotic behavior. The only small exception to this is FLS8 for error norms of $\lesssim$1E-15, when the round-off error starts to dominate the global numerical error and the scheme then loses its theoretical

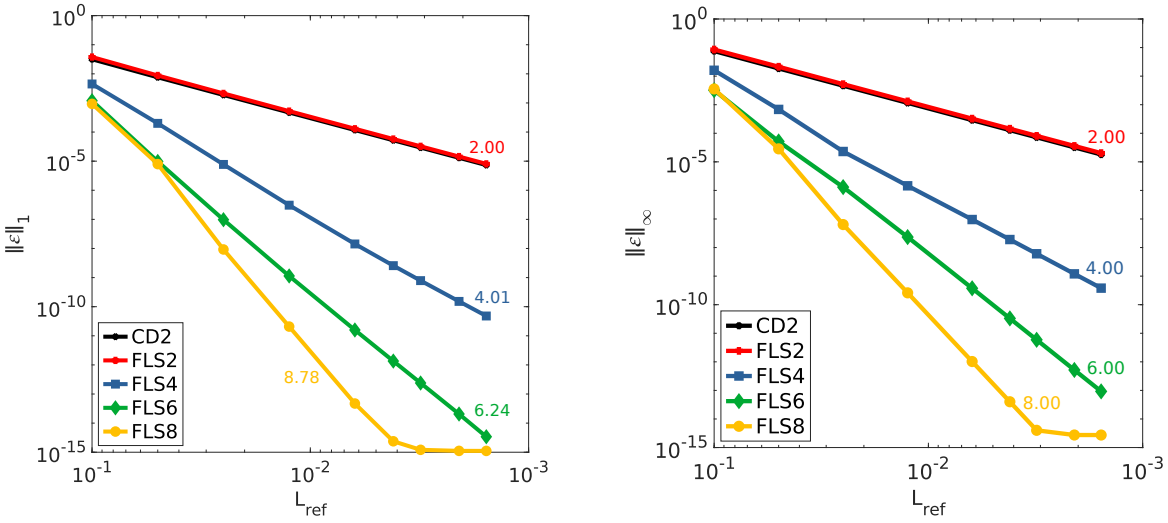


Figure 3.2 Numerical error norms $\|\varepsilon\|_1$ (left) and $\|\varepsilon\|_\infty$ (right) for two-dimensional FLS schemes, using the sinusoidal analytical solution given by (3.1). Values for CD2 are also shown for comparison.

convergence order. In terms of numerical accuracy, the obtained results agree with those obtained by Diogo [32], save for fluctuations due to a different choice of solver. Note also that the FLS2 scheme presents a slightly larger error than its CD2 counterpart, since the latter's numerical error contains additional contributions from the weighted least-squares regression error. Overall, these results demonstrate the correct implementation of the two-dimensional FLS schemes.

Comparison between Quadruple- and Double-Precision Results

The results shown above were obtained using quadruple precision, which helps delay the contamination from round-off errors as stated in Section 2.6.3. Nonetheless, in Figure 3.2 this limit is clearly identifiable by the FLS8 plateau around $\|\varepsilon\| \sim 10^{-15}$. If one were to use double precision instead, this value would be even higher. This is shown in Figure 3.3, which plots the results obtained using quadruple and double precision for chosen FLS schemes alongside each other. The use of quadruple precision leads to a significant improvement in the numerical accuracy for finer meshes with $N_C \gtrsim 100^2$, which quickly become dominated by round-off errors when using double precision. Notice that the saturation values when using double precision are consistent with the results obtained by Diogo [32] for both the FLS6 and FLS8 schemes.

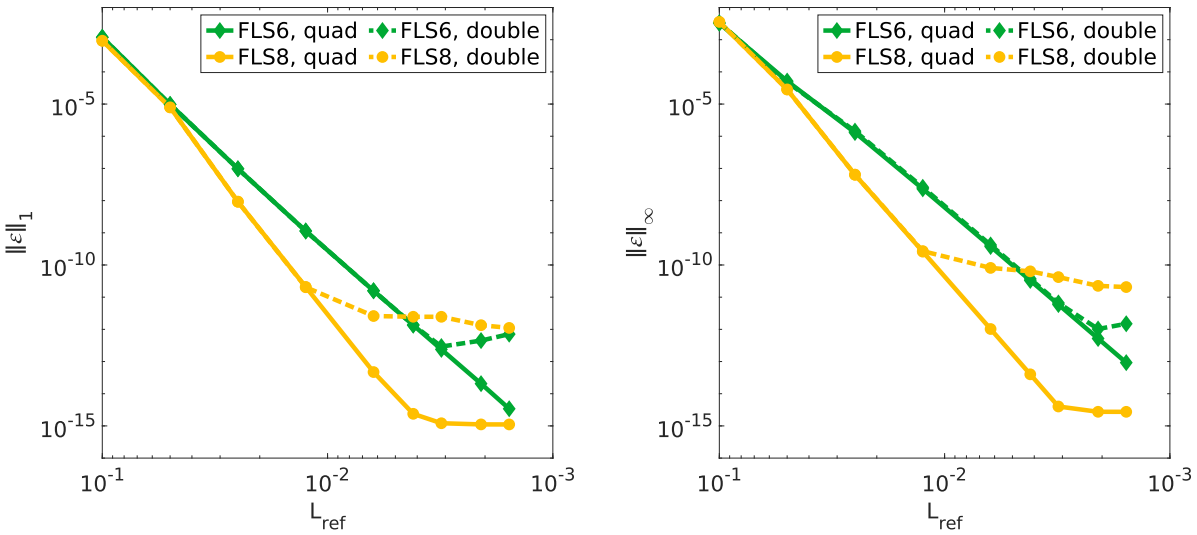


Figure 3.3 Influence of floating-point format on the resulting error norms $\|\varepsilon\|_1$ (left) and $\|\varepsilon\|_\infty$ (right) for FLS6 and FLS8 schemes in 2D, using the analytical solution given by (3.1). Solid curves represent results obtained with quadruple precision, while dashed curves represent those obtained with double precision.

3.1.2 Three-Dimensional Case

As with the previous case, the test was run in 128-bit quadruple precision using BiCGSTAB preconditioned with block-Jacobi. The error norms $\|\varepsilon\|_1$ and $\|\varepsilon\|_\infty$ for the three-dimensional case are plotted in Figure 3.4, with the corresponding convergence orders annotated for each method.

Similarly to the two-dimensional case, the three-dimensional implementation of all FLS schemes exhibit the expected asymptotic behavior. Unlike the previous case, the results do not enter a region where round-off error dominates. This happens because the error is not small enough and the 3D problem is memory-limited – i.e. using a finer grid would exceed the system's available memory capacity.

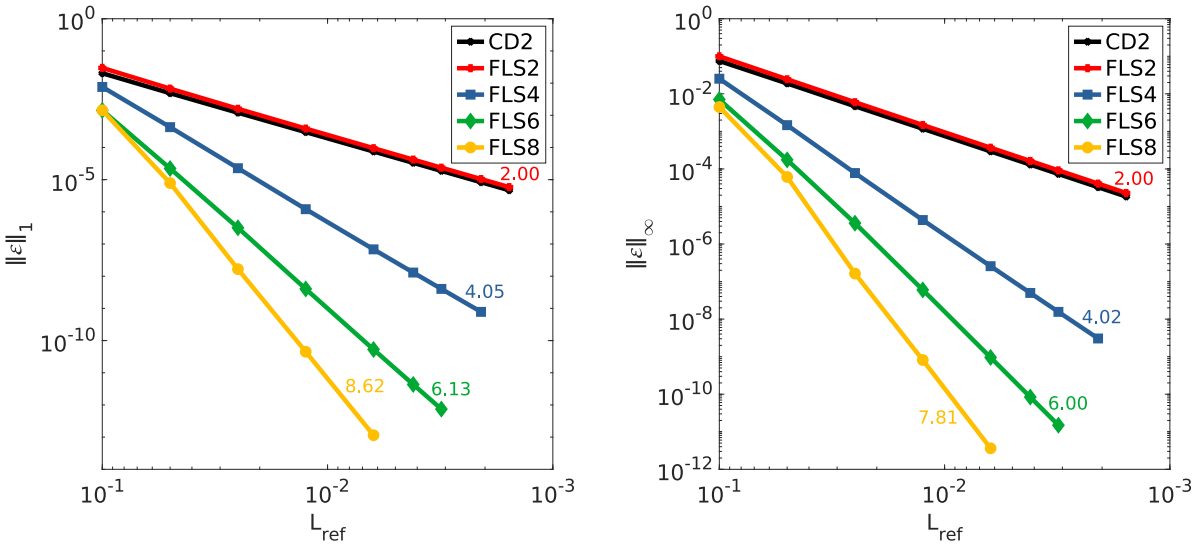


Figure 3.4 Numerical error norms $\|\varepsilon\|_1$ (left) and $\|\varepsilon\|_\infty$ (right) for three-dimensional FLS schemes, using the sinusoidal analytical solution given by (3.1). Values for CD2 are also shown for comparison.

3.2 Alternative Methods for the Regression Polynomial

This section introduces two alternative methods regarding the regression polynomial and the calculation of the coefficient vector $\boldsymbol{\kappa}_f$. To compare these approaches to the original one, tests in two dimensions were performed using the following analytical solution, with periodic boundary conditions:

$$\varphi(\vec{x}/2\pi) = \sin x \sin y + \sin x \cos y + \cos x \sin y + \cos x \cos y \,. \tag{3.2}$$

This analytical solution was chosen over the previous (3.1) because it has a greater variety of terms in its Taylor series expansion, thus leading to a better comparison of the regression polynomial approaches. Furthermore, the use of periodic boundary conditions leads to a faster and simpler comparative analysis of the asymptotic regime.

3.2.1 Extended Regression Polynomial

In the original approach of the FLS method, the regression polynomial is symmetric regarding its tangent and normal terms. However, the stencil algorithm introduced in Section 2.2 generates a stencil that has an extra layer cells in the tangential direction relative to the target face (on which the regression is centered) than in the normal one. As such, it is reasonable to extend the regression polynomial in the tangential direction by introducing additional layer of high-order terms. This is illustrated in Figure 3.5, which shows the terms included in the regression polynomial for each scheme – with the colors red, blue, green, and yellow corresponding to second-, fourth-, sixth-, and eight-order schemes, respectively. The base terms required for the original weighted least-squares (WLS) approach are shown upon a solid background, while the additional terms for this extended approach are marked by stripes. Note that the polynomial for any high-order scheme also includes the terms corresponding to lower-order schemes.

The numerical errors corresponding to each approach is plotted in Figure 3.6; the plot shows the 1-norm errors $\|\varepsilon\|_1$, while the auxiliary table lists relative error reductions obtained from the extended approach. As shown, the extended approach provides consistent error reduction, increasing with the

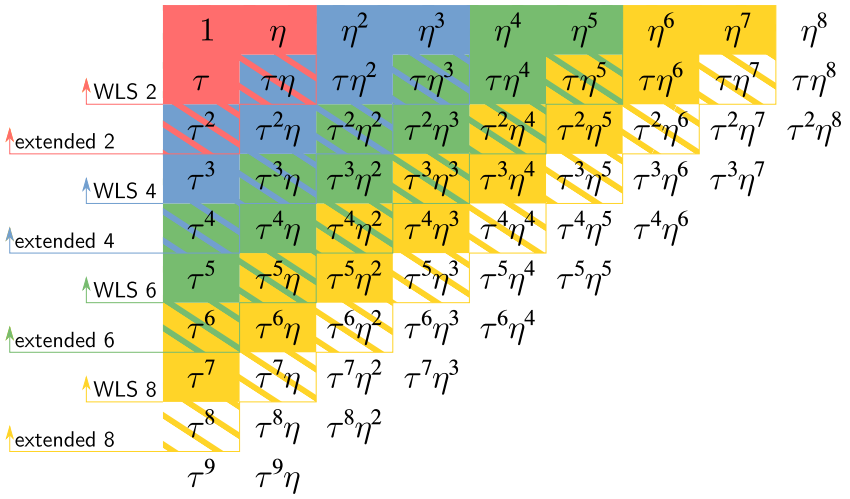
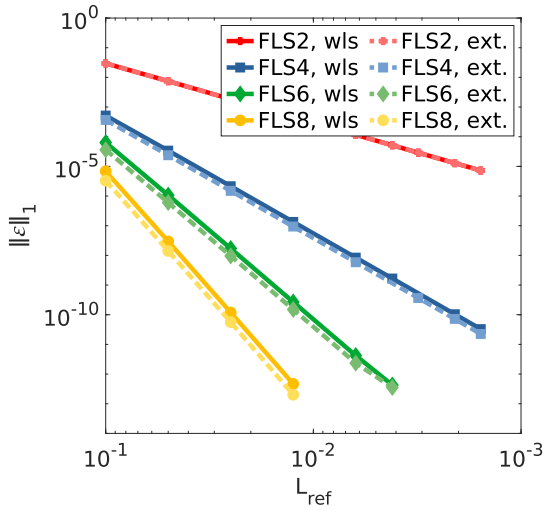


Figure 3.5 Polynomial terms included in the regression polynomial for each scheme in the original and extended approaches. The colors red, blue, green, and yellow correspond to second-, fourth-, sixth-, and eight-order schemes, respectively. The base terms required for the original approach are shown upon a solid background, while the additional terms for the extended approach are marked by stripes.

scheme order since more terms are included. The obtained reductions are between 30% and 50% for the high-order schemes, being 53% for FLS8. However, this improvement comes at the cost of a slight increase in computation time when solving the local least-square matrices.

Diogo [32] previously studied this approach with the vertex-neighbors stencil algorithm. Although the analytical solution used for that study was slightly different – corresponding to (3.1) – the error reductions obtained were more significant than those obtained here. However, the face-neighbors stencil already produces a more accurate result than the vertex-neighbors, so a less drastic improvement for the extended approach with the former was expected. The norm-1 error obtained by Diogo for FLS2 was reduced by 17%, the one for FLS4 by 59%, the one for FLS6 by 71%, and the one for FLS8 by 72%.



	Error reduction $1 - \|\varepsilon\|_1^{\text{ext}} / \|\varepsilon\|_1^{\text{wls}}$
FLS2	0.2 %
FLS4	30 %
FLS6	44 %
FLS8	53 %

Figure 3.6 Comparison of the numerical error norm $\|\varepsilon\|_1$ obtained for the original (solid line) and extended polynomial (dashed line) approaches, using the analytical solution given by (3.2). The auxiliary table lists the corresponding error reduction obtained with the extended approach for each scheme.

3.2.2 Direct Computation of Polynomial Coefficients

In the general case of an unstructured grid, the layout of cells in the grid is case-specific. Therefore, even with a well-defined algorithm, the stencil dimensions are unknown *a priori*. This is the main reason behind the use of a WLS approach when building these numerical schemes. By having a stencil of size larger than the required number of polynomial coefficients, this approach allows for a stable and robust scheme even for the most irregular of grids. However, this comes at cost of regression errors, which are an additional error source induced from the least-squares regression procedure.

However, the layout of cells in a structured grid follows a regular and known geometrical pattern. Thus, for a given stencil algorithm, the resulting stencil size is known *a priori*. As such, it is possible to use an alternative approach which does not induce regression error. By having the same number of stencil points as the required polynomial coefficients, one may directly define the pseudo-inverse matrix as the inverse of the distance matrix $\mathbf{D}_f$, without the need for a statistical regression,

$$\boldsymbol{\varphi}_{s(f)} = \mathbf{D}_{s(f)}\boldsymbol{\kappa}_f \qquad \Rightarrow \qquad \boldsymbol{\kappa}_f = \mathbf{P}_{s(f)}\boldsymbol{\varphi}_{s(f)}\,, \quad \mathbf{P}_{s(f)} \equiv \mathbf{D}_{s(f)}^{-1} \tag{3.3}$$

Diogo [32] studied how this direct approach compared to the WLS approach, using the vertex-neighbor stencil algorithm, for the case of convection-diffusion in two dimensions. In two dimensions, this stencil algorithm produces, for a p-order scheme, a rectangular stencil with $p+1$ points in the tangential direction from the target face and p points in the normal direction, for a total of p^2+p stencil points. On the other hand, the face-neighbor stencil algorithm used in the present work and described in Section 2.2 produces, in two dimensions, a hexagonal stencil with $\frac{1}{2}p^2 + 2p$ points. For schemes of fourth-order and higher, this hexagonal stencil can be obtained from the rectangular stencil of the former algorithm by trimming its corners; for the second-order case, both algorithms produce the same stencil.

For a p-order scheme, the number of required terms in the polynomial approximation is $\frac{1}{2}p^2 + \frac{1}{2}p$, which is less than the stencil points produced by the aforementioned algorithms. As such, to proceed with the direct computation of the polynomial coefficients, either the stencil algorithms must be modified or more terms must be included in polynomial. Following the study by Diogo [32], this work will consider solely the latter approach. Figure 3.7 illustrates the polynomial terms for each scheme – with the colors red, blue, green, and yellow corresponding to second-, fourth-, sixth-, and eight-order schemes, respectively. The base terms required for the WLS approach are shown upon a solid background, while the additional terms for the direct approach are marked by stripes. Note that the polynomial for any high-order scheme also includes the terms corresponding to lower-order schemes.

The numerical errors corresponding to each approach are compared in Figure 3.8; the plot shows the 1-norm errors $\|\varepsilon\|_1$, while the auxiliary table lists relative error reductions obtained from the direct approach. As shown, the direct approach provides a significant error reduction, of up to 56% in FLS8. However, it is important to note that this approach is, in general case of unstructured grids, not feasible due to the unknown geometric distribution of the local stencils. Lastly, the Author notes that the outlier behavior from FLS4 – which presents a higher error using the direct approach – remains unexplained, and its investigation is therefore left as future work.

In the study by Diogo [32] – using as analytical solution (3.1) – the norm-1 error for FLS6 was reduced by 74% and for FLS8 by 79%. As was the case with the extended approach, the error reductions obtained in the present study are less pronounced than those obtained by Diogo. However, as was the case with the extended approach, this was expected since the face-neighbor stencil used here already produces a more accurate result than the vertex-neighbor stencil used by Diogo.

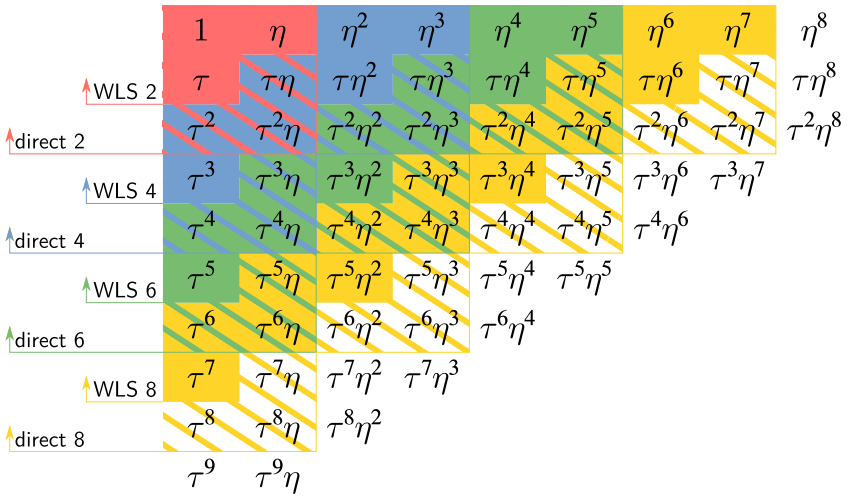
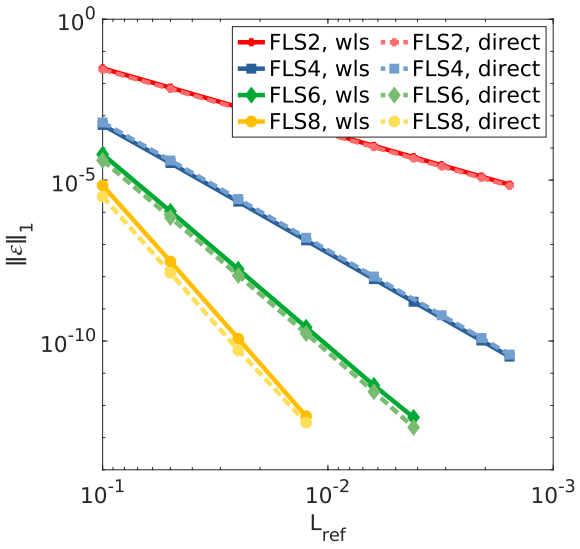


Figure 3.7 Polynomial terms included in the regression polynomial for each scheme in the original WLS and direct approaches. The colors red, blue, green, and yellow correspond to second-, fourth-, sixth-, and eight-order schemes, respectively. The base terms required for the original approach are shown upon a solid background, while the additional terms for direct approach are marked by stripes.



	Error reduction $1 - \|\varepsilon\|_1^{\mathrm{dir}} / \|\varepsilon\|_1^{\mathrm{wls}}$
FLS2	8 %
FLS4	-20 %
FLS6	36 %
FLS8	56 %

Figure 3.8 Comparison of numerical error norm $\|\varepsilon\|_1$ obtained for the original WLS (solid line) and direct (dashed line) approaches, using the analytical solution given by (3.2). The auxiliary table lists the corresponding error reduction obtained with the direct approach for each scheme.

3.3 Selection of Preconditioner and Solver

This section describes the preliminary studies done with the two-dimensional code to help choose a preconditioner-solver combination for the full-fledged studies on parallel performance (see Section 3.4). Both versions of the code use the same approach to solve the linear system resulting from the discretization process. Using the PETSc framework, the system is solved with an iterative linear solver, through the combination of a preconditioner (PC) and a Krylov subspace method (KSP) [58]. Table 3.1 lists the full names and PETSc acronyms of the preconditioners and solvers considered in this work.

The main goal of this preliminary study was to reduce the myriad choices of preconditioner-solver combinations available in the PETSc library into a handful of promising candidates, selected mostly based on solver runtime. Given the preliminary nature of this study, it was performed in an in-house

Table 3.1 List of full names and PETSc acronyms for preconditioners (PCs) and solvers considered.

	PETSc Acronym	Name
PCs	BJACOBI	Block-Jacobi
	ASM	Additive Schwarz Method
	GAMG	Geometric Algebraic Multigrid
Solvers	CG	Conjugate Gradient
	BICG	BiConjugate Gradient
	BCGS	BiConjugate Gradient Stabilized
	IBCGS	Improved BiConjugate Gradient Stabilized
	GMRES	Generalized Minimal Residual
	FGMRES	Flexible Generalized Minimal Residual
	TFQMR	Transpose-Free Quasi-Minimal Residual

Linux server at Laboratório de Simulação em Energia e Fluidos (LASEF) instead of at a proper high-performance computing (HPC) system. As such, the resulting runtimes values should not to be taken as absolute, but instead used as an order-of-magnitude estimates – which is sufficient for the desired comparison. The server – codenamed Hopper – consists of two 8-core Intel Xeon E5-2650 processors running at 3 GHz.

The results of solver runtime (including preconditioner setup time) for the FLS2 scheme are shown in Table 3.2 (serial) and Table 3.3 (parallel, $N_P = 8$), where a dash indicates that either the combination is invalid or the system did not converge. The entries for the additive Schwartz method (ASM) and geometric algebraic multigrid (GAMG) preconditioners have divisions that represent the additional parameters of overlap (ASM-only) and threshold percentage (GAMG-only). As shown, the choice of GAMG preconditioner considerably reduces the runtime by orders-of-magnitude in relation to Block Jacobi and ASM. However, this preconditioner resulted in a stable method only for lower threshold values (≤ 0.02). In relation to solvers, those which formed a valid combination with the GAMG preconditioner exhibit equivalent

Table 3.2 Serial runtime in seconds for preconditioner-solver combinations, with FLS2 scheme.

KSP \ PC	BJACOBI	ASM				GAMG			
		0	1	2	3	0	2	4	6
CG	4412	2625	2530	3279	2600	11 095	133	82	93
BICG	716	834	832	832	831	-	-	-	-
BCGS	548	1063	737	733	739	71	66	73	97
IBCGS	451	563	561	555	567	-	-	-	-
GMRES	2778	4521	2861	3048	2984	61	58	66	85
FGMRES	6076	4990	3700	3639	3037	62	73	82	95
TFQMR	5263	5571	4565	4852	4384	58	54	63	77

Table 3.3 Parallel runtime in seconds, using 8 cores, for preconditioner-solver combinations, with FLS2 scheme.

KSP \ PC	BJACOBI	ASM				GAMG			
		0	1	2	3	0	2	4	6
CG	277	341	-	-	-	1700	13	-	-
BICG	138	191	21	14	7	-	-	-	-
BCGS	100	125	129	113	100	12	11	-	-
IBCGS	110	133	89	110	121	-	-	-	-
GMRES	-	-	414	414	419	11	10	-	-
FGMRES	486	683	719	625	594	13	14	-	-
TFQMR	107	140	-	-	-	12	11	-	-

runtimes, except for an outlier in the conjugate gradient (CG) solver. It is also worth noting the excellent values yielded by the combination of ASM-BiCG.

The previous results with FLS2 showed the promising combinations to be ASM with BiCG and GAMG with any valid solver except CG. As such, to determine if any of these good results were specific to the FLS2 case (e.g. due to the thin matrix bandwidth), additional tests using the FLS8 scheme were performed. The results are shown in Table 3.4, where a dash indicates that the solution did not converge. The ASM-BiCG combination performs poorly in comparison to the GAMG combinations. On the other hand, the GAMG threshold value of zero is now unstable.

Table 3.4 Runtime in seconds for preconditioner-solver combinations, with FLS8 scheme.

	ASM			
	0	1	2	3
serial, BICG	10629	11867	10657	12233
parallel, BICG	2155	-	-	1816

	serial, GAMG				parallel, GAMG			
	0	2	4	6	0	2	4	6
BCGS	-	820	1337	-	-	154	213	-
GMRES	-	770	1298	-	-	165	210	-
FGMRES	-	961	1579	-	-	180	283	-
TFQMR	-	789	1290	-	-	148	206	-

These results indicate that, for the given problem, the best choice of preconditioner is GAMG with a threshold value of 0.02. Since the solvers analyzed in combination with GAMG yielded equivalent results, the BiCGSTAB solver was chosen due to its long-standing tradition among the computational fluid dynamics (CFD) community.

3.4 Parallel Efficiency

This section describes the results obtained regarding the FLS schemes' parallel performance, which were determined using strong- and weak-scaling tests. As mentioned in Section 2.6, these two methods of measuring scalability differ in relation to the evolution of the problem size as the number of processes increase. In strong scaling, the problem size is kept fixed throughout the test; in weak scaling, it is the workload (i.e. problem size per process) that is kept fixed. While both metrics provide useful information, the weak-scaling tests better simulate engineering situations where more resources are allocated to larger problems.

The main issue encountered in executing these tests was the fact that, for higher-order methods, the memory requirement is quite restrictive. This is not a problem when running the code in parallel, since this memory cost is distributed among several machines. However, when running serially in a single machine, the available memory is sometimes insufficient for the complete problem. This means that a full scaling test (i.e. starting from a serial execution) with high-order methods is only possible for smaller problem sizes. As such, the chosen starting mesh size for all schemes was $N_C^{\text{ref}} = 80^3$, with the mesh size evolution for weak-scaling tests listed in Table 3.5.

These tests were performed at the Oblivion supercomputer, which is the newest center of Rede Nacional de Computação Avançada (RNCA) – the Portuguese HPC network [103]. Built with funds from

Table 3.5 Grid sizes N_C, given a number of processors N_p, for weak-scaling tests with reference $N_C^{\text{ref}} = 80^3$.

Grid Size (weak scaling, $N_C^{\text{ref}} = 80^3$)								
N_p	1	2	4	8	16	32	64	128
N_C	80^3	101^3	127^3	160^3	202^3	254^3	320^3	403^3

the EuroHPC Initiative and managed by HPC Center of the Universidade de Évora, this supercomputer consists of 58 nodes, each with two sockets containing 18-core Intel Xeon Gold 6154 processors running at 3 GHz, connected via Infiniband EDR. This entails a peak theoretical performance of ~240 Tflop/s.

The results of selected strong- and weak-scaling tests using $N_C^{\text{ref}} = 80^3$ are shown in Figure 3.9 as plots of solver runtime (SRT), with the corresponding efficiency listed and the dashed lines showing ideal scaling. As expected, for the same grid size higher-order methods exhibit a higher SRT due to a higher number of nonzeros in the matrix. As also expected, the deviation from ideal scaling increases with the number of processes; however, this seems to stabilize for $N_p \geq 4$.

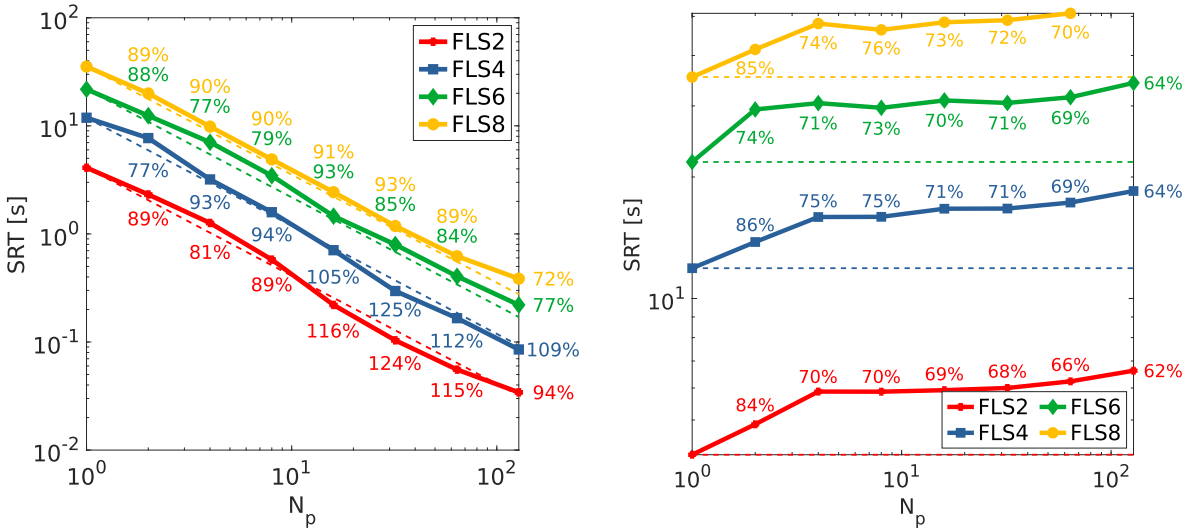


Figure 3.9 Solver runtime SRT for strong scaling (left) and weak scaling (right), using a reference size $N_C = 80^3$. Dashed lines indicate the ideal scaling, while listed values indicate the obtained parallel efficiencies.

For the strong-scaling tests, a significant improvement of efficiency can be observed for 16 and 32 processes, especially for the second- and fourth-order schemes. This is due to an increase in cache performance from the additional cache memory of more systems. This advantage starts to disappear with 64 processes (spread among 32 nodes, or 64 sockets), when the socket bandwidth starts to limit the computation rate. Such decrease in efficiency is more evident with $N_p = 128$, because there is no additional L1 cache relative to $N_p = 64$ since no more sockets are included. This improved cache performance does not happen for higher-order methods (at the considered problem size) because of their higher memory requirement; if a smaller problem size were considered, the same trend would also be observed for the sixth- and eighth-order methods.

As seen by the listed values of weak scaling, the higher-order methods tend to exhibit slightly higher weak-scaling efficiencies of close to 5% higher, which shows a marginal advantage for these methods. Although this is a very promising result, one could argue that the decreased efficiency of the lower-order method stem not from an inherently worse parallel suitability of FLS2, but instead of its lower SRT. This happens because, given a lower runtime, communication and parallel overhead costs have a higher negative effect on the overall parallel performance.

Therefore, another batch of strong- and weak-scaling tests were performed using, instead of a reference size, the same reference SRT for the initial serial execution independently of the numerical scheme. The chosen reference SRT for the strong-scaling tests was $T_s \sim 40\,\text{s}$, while that for weak-scaling tests was $T_s \sim 20\,\text{s}$. The corresponding grid sizes for each test and scheme are listed on Table 3.6.

The results obtained from the aforementioned tests are plotted in Figure 3.10. Using this reference

Table 3.6 Grid sizes N_C, for a given number of processors N_p, in strong- and weak-scaling tests with reference initial solver runtimes of $T_s \sim 40\,\text{s}$ and $T_s \sim 20\,\text{s}$, respectively.

Test Type	N_p	Grid Size N_C			
		FLS2	FLS4	FLS6	FLS8
Strong Scaling	-	160^3	127^3	101^3	80^3
Weak Scaling	1	127^3	101^3	80^3	63^3
	2	160^3	127^3	101^3	80^3
	4	202^3	160^3	127^3	101^3
	8	254^3	202^3	160^3	127^3
	16	320^3	254^3	202^3	160^3
	32	403^3	320^3	254^3	202^3
	64	508^3	403^3	320^3	254^3
	128	640^3	508^3	403^3	320^3

base, the FLS2 scheme exhibits the best parallel performance, with $\eta^{st} \sim 90\%$ and $\eta^{wk} \sim 80\%$. The FLS8 also exhibits an excellent strong-scaling efficiency of $\eta^{st} \sim 90\%$, but has a lower weak-scaling efficiency that is on par with the other methods. The FLS6 exhibited the worse strong-scaling efficiency, with values falling below 70%. Even though, for this comparison, the FLS2 scheme performed best, one could also argue that it was because of the larger problem size used for it, which could result in a better load balancing between processes.

It is important to discuss the observed drop in parallel performance during the weak-scaling tests for values of $N_p > 32$. This happens because the Oblivion supercomputer has 58 nodes, meaning that for $N_p \geq 64$, there will be processes sharing a node and all its related infrastructure: cache, bandwidth, etc. This will inevitably lead to a performance decrease, which only worsens as the number of processes increase. Nonetheless, one may provide an estimate to the SRT values were these later runs executed with a one-to-one mapping of processes to nodes. Let $T(N_C, N_p, N_n)$ be the SRT for an N_C-sized problem executed with N_p processes spread out among N_n nodes. Then, the loss of efficiency in running the problem with a k-fold higher process-per-node concentration may be estimated by the ratio

$$\frac{T(N_C, N_p, kN_n)}{T(N_C, N_p, N_n)}\,. \tag{3.4}$$

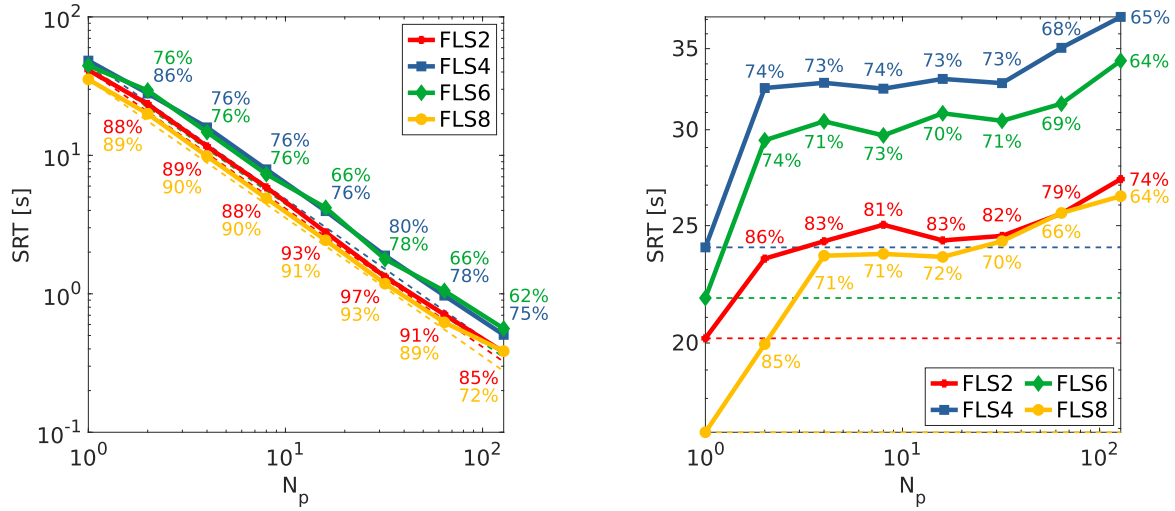


Figure 3.10 Solver runtime SRT for strong scaling (left) and weak scaling (right), using reference SRTs of ~40 s and ~20 s, respectively. Dashed lines indicate ideal scaling; the parallel efficiencies are listed.

Therefore, the SRT of a large problem if it were run with a one-to-one mapping of processes to nodes may be estimated using the aforementioned ratio constructed from SRTs of a smaller problem using less computation resources. For example, for the case of $N_p = 64$, the estimated SRT using this correction may be given by the following expression,

$$T(N_C^{\text{large}}, 64, 64) \;\approx\; T(N_C^{\text{large}}, 64, 32)\, \frac{T(N_C^{\text{small}}, 2, 1)}{T(N_C^{\text{small}}, 2, 2)}\,. \tag{3.5}$$

Using this correction method, the estimated strong and weak parallel efficiencies for 64 and 128 processes are listed in Table 3.7. These estimates better continue the efficiency trend from previous N_p values, without having such an acute drop in efficiency.

Table 3.7 Estimated weak-scaling parallel efficiency η^{wk} for high N_p values, using the correction method described by (3.4),(3.5). Efficiency values are given as percentage points.

Reference base	N_p	η^{wk} [%]			
		FLS2	FLS4	FLS6	FLS8
Size	64	69	74	72	73
	128	67	71	68	-
SRT	64	77	70	68	72
	128	82	72	72	70

3.5 Computational Cost

Lastly, it is worthwhile to compare the computational cost of each FLS method. Figure 3.11 plots the numerical error norm $\|\varepsilon\|_1$ of each method as function of two different metrics for computational cost. On the left, it is plotted against the number of nonzeros N_{nz} of the corresponding coefficient matrix, which is an indirect measurement of the memory required to store the matrix. On the right, it is plotted as function of the mean SRT (measured in CPU time, i.e. $N_p \times T$).

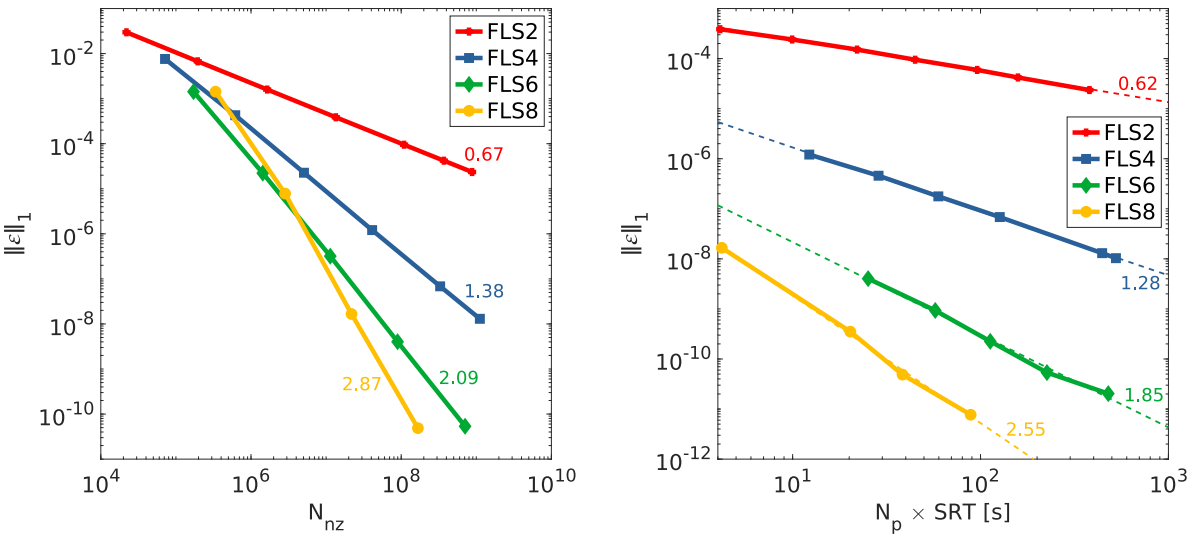


Figure 3.11 Numerical error norm $\|\varepsilon\|_1$ as function of number of nonzeros N_{nz} (left) and solver runtime SRT given in CPU time (right). The listed values represent the decay order of $\|\varepsilon\|_1$.

The figure above clearly shows the computational advantage of higher-order methods. For a given memory or time constraint, higher-order methods yield significantly more accurate results. This may also be seen through another manner: to obtain a given accuracy, a higher-order method requires significantly less memory and time. Both of these are important budget constraints of computational projects, so having methods that are more memory- and time-efficient is a considerable advantage.

Chapter 4

Conclusion

The face least-squares (FLS) schemes previously developed by Vasconcelos [30] and Diogo [32] were implemented in parallel for high-performance computing (HPC) and expanded for three-dimensional problems. The resulting algorithm was verified using the previous Matlab code and also using manufactured analytical solutions based on sinusoidal functions. The influence of the floating-point format used was also analyzed, with the quadruple-precision format providing a significant increase in accuracy for finer meshes in the sixth- and eighth-order schemes. This improvement was achieved by delaying the onset of major contamination by round-off errors in the results. Therefore, an equivalent improvement can also be obtained for second- and fourth-order schemes, given sufficiently fine meshes.

In preparation for the parallel scalability tests, a comparison study of preconditioner-solver combinations was performed at an in-house machine from Laboratório de Simulação em Energia e Fluidos (LASEF). The criteria for choosing the combination to be used in the scalability tests were that of fastest runtime (serial and parallel) and robustness. The choice of geometric algebraic multigrid (GAMG) preconditioner was clear from the obtained results, which showed it vastly outperforming its counterparts in both serial and parallel runtime. The choice of solver was less obvious, with four solvers performing the best with not much difference between them, when paired with the GAMG preconditioner. As such, the Author chose to use the biconjugate gradient stabilized method (BiCGSTAB) solver due to its long-standing tradition, especially within the computational fluid dynamics (CFD) community.

The parallel performance of the developed algorithm was analyzed for three-dimensional problems at the Oblivion supercomputer managed by the HPC Center of Universidade de Évora. To measure scalability, a series of scalability tests were performed, with both mesh size and solver runtime fixed as reference points, thus yielding a total of four performance tests for comparison between each FLS scheme. No one scheme presented a significant advantage over others across all tests, with values for strong- and weak-scaling efficiencies of over, respectively, 85% and 70% being achieved consistently. Overall, the obtained results show that high-order methods, even with their increased number of halo cells for communication, may indeed have a parallel performance that is on par with low-order methods. Thus, this provides evidence to refute a major argument against the use of high-order methods in HPC.

Furthermore, the computational cost of low- and high-order FLS schemes were compared. This cost was measured in terms of both memory and runtime requirements for a given accuracy. The obtained results showed that, for a q-order scheme, the numerical error's 1-norm decreased with $\mathcal{O}\left(N_{nz}^{q/3}\right)$ and $\mathcal{O}\left(T^{q/3}\right)$, where N_{nz} and T are the number of nonzeros and the solver runtime (CPU time). Using either of these metrics, the advantage of high-order methods regarding cost efficiency was clearly demonstrated and, thus, presents a major incentive to their use over low-order methods.

Additionally, two alternative methods regarding the regression polynomial for face-neighboring stencils were developed. Following the work of Diogo [32] with vertex-neighboring stencils, these encompassed extending the polynomial to include higher-order terms in the tangential direction and calculating the coefficient vector directly instead of relying on the weighted least-squares approach. Although these methods do not affect the schemes' order of convergence, they did lead to a reasonable reduction in the numerical error of the high-order schemes, at the cost of a slight increase to computation effort. In particular, the extended approach appears quite promising given its flexibility and ease of implementation. On the other hand, the direct approach remains a niche method for Cartesian grids and also presents implementation challenges for non-periodic boundary conditions.

4.1 Future Work

The methods studied and implemented during the current work have shown to be very promising. As such, there are several research avenues for future work, regarding either the computational and numerical aspects. Furthermore, the current project has recently received additional computational resources, this time in the Bob supercomputer operated by the Minho Advanced Computing Center (MACC) [104]. Performing additional tests on a different cluster aids in verifying overall parallel performance, since the results from a single-machine test may contain hardware bias. These tests are currently ongoing.

Given that a higher matrix bandwidth leads to higher memory and time requirements, it could be worthwhile to study how to decrease these effects. One possibility is using fast Fourier transforms along one or more dimensions, which Costa [105] has shown to yield promising results in parallel computing. The Author has completed some preliminary research into this topic for the course *Project in Aerospace Engineering*, with results presented in Appendix C. Another approach is to use matrix-free methods, which consist of using values that are known at compile time rather than calculating them during runtime. As such, these methods do not require the explicit storage of the matrix. However, such approach is not valid for unstructured grids, whose layouts are unknown at compile time. Nonetheless, matrix-free methods could provide a significant improvement for structured grids, especially given the limiting memory requirements that were observed with high-order schemes. Currently, PETSc also supports this matrix-free approach for iterative solvers.

The parallel implementation introduced in this work was based on a domain-decomposition approach with MPI using PETSc, resulting in a program suited for CPU-only computers. Given the increasing adoption of accelerators such as graphics processing units (GPUs) and general-purpose graphics processing units (GPGPUs) in the HPC field, it would be interesting to adapt the present algorithm for such devices. The PETSc framework already provides an interface for GPU kernel parallelism, including the popular CUDA model. However, data must be copied from the CPU to the GPU. Since this memory copy process is usually slow compared to GPU computing speed, it needs to be keep at a minimum in order to achieve good performance. Thus, practically all computation should be done on the GPU, which entails a major overhaul of the existing code.

Further developments to the numerical aspects of the FLS schemes consist of progressing towards the solution of the Navier-Stokes equations. Previously, Costa [33] developed temporal discretization for the FLS schemes, verifying the resulting algorithm for two-dimensional problems. Currently, there is ongoing research within the HIBforMBP group regarding the implementation of the fractional-step method with FLS schemes. These developments can be integrated into the existing code, for a parallel implicit solver of the Navier-Stokes equations.

Bibliography

[1] Olver PJ. *Introduction to Partial Differential Equations*. Springer, 2014. ISBN: 978-3-319-02099-0. DOI: 10.1007/978-3-319-02099-0.

[2] Patankar SV. *Numerical Heat Transfer and Fluid Flow*. Hemisphere, 1980. ISBN: 0-89116-522-3.

[3] *Millennium Problems*. Clay Mathematics Institute. URL: https://www.claymath.org/millennium-problems (visited on 2022).

[4] Anderson Jr JD. *Computational Fluid Dynamics. The Basics with Applications*. McGraw-Hill, 1995. ISBN: 0-07-001685-2.

[5] Hirsch C. *Numerical Computation of Internal and External Flows*. Vol. 1. 2nd ed. Butterworth-Heinemann, 2007. ISBN: 978-0-7506-6594-0. DOI: 10.1016/B978-0-7506-6594-0.X5037-1.

[6] Ferziger JH, Perić M, Street RL. *Computational Methods for Fluid Dynamics*. 4th ed. Springer, 2020. ISBN: 978-3-319-99693-6. DOI: 10.1007/978-3-319-99693-6.

[7] Johnson FT, Tinoco EN, Yu NJ. "Thirty years of development and application of CFD at Boeing Commercial Airplanes, Seattle". In: *Computers & Fluids* 34.10 (2005), pp. 1115–1151. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2004.06.005.

[8] Abbas-Bayoumi A, Becker K. "An industrial view on numerical simulation for aircraft aerodynamic design". In: *Journal of Mathematics in Industry* 1.10 (2011). ISSN: 2190-5983. DOI: 10.1186/2190-5983-1-10.

[9] Kobayashi T, Tsubokura M. "CFD Application in Automotive Industry". In: *100 Volumes of 'Notes on Numerical Fluid Mechanics': 40 Years of Numerical Fluid Mechanics and Aerodynamics in Retrospect*. Ed. by Hirschel EH, Krause E. Springer, 2009, pp. 285–295. ISBN: 978-3-540-70805-6. DOI: 10.1007/978-3-540-70805-6_22.

[10] Francesconi G, Miretti L, Lorefice L, Pitillo F, Paola N. "Application of Adjoint Methods on Drag Reduction of Current Production Cars". In: *CO2 Reduction for Transportation Systems Conference*. SAE International, 2018. DOI: 10.4271/2018-37-0016.

[11] Ando K, Kuratani N, Fukuda H. "Aerodynamic Performance Evaluation System at the Early Concept Stage of Automotive Styling Development Based on CFD". In: *SAE 2016 World Congress and Exhibition*. SAE International, 2016. DOI: 10.4271/2016-01-1584.

[12] Trivedi C, Cervantes MJ, Gunnar Dahlhaug O. "Numerical Techniques Applied to Hydraulic Turbines: A Perspective Review". In: *Applied Mechanics Reviews* 68.1 (2016). ISSN: 0003-6900. DOI: 10.1115/1.4032681.

[13] Blocken B. "50 years of Computational Wind Engineering: Past, present and future". In: *Journal of Wind Engineering and Industrial Aerodynamics* 129 (2014), pp. 69–102. ISSN: 0167-6105. DOI: 10.1016/j.jweia.2014.03.008.

[14] Mehta D, van Zuijlen A, Koren B, Holierhoek J, Bijl H. “Large Eddy Simulation of wind farm aerodynamics: A review”. In: *Journal of Wind Engineering and Industrial Aerodynamics* 133 (2014), pp. 1–17. ISSN: 0167-6105. DOI: 10.1016/j.jweia.2014.07.002.

[15] Yadav AS, Bhagoria J. “Heat transfer and fluid flow analysis of solar air heater: A review of CFD approach”. In: *Renewable and Sustainable Energy Reviews* 23 (2013), pp. 60–79. ISSN: 1364-0321. DOI: 10.1016/j.rser.2013.02.035.

[16] Tabet F, Gökalp I. “Review on CFD based models for co-firing coal and biomass”. In: *Renewable and Sustainable Energy Reviews* 51 (2015), pp. 1101–1114. ISSN: 1364-0321. DOI: 10.1016/j.rser.2015.07.045.

[17] Dernbecher A, Dieguez-Alonso A, Ortwein A, Tabet F. “Review on modelling approaches based on computational fluid dynamics for biomass combustion systems”. In: *Biomass Conversion and Biorefinery* 9.1 (2019), pp. 129–182. ISSN: 2190-6823. DOI: 10.1007/S13399-019-00370-Z.

[18] Golshan S, Sotudeh-Gharebagh R, Zarghami R, Mostoufi N, Blais B, Kuipers J. “Review and implementation of CFD-DEM applied to chemical process systems”. In: *Chemical Engineering Science* 221 (2020), p. 115646. ISSN: 0009-2509. DOI: 10.1016/j.ces.2020.115646.

[19] Ngo SI, Lim Y-I. “Multiscale Eulerian CFD of Chemical Processes: A Review”. In: *ChemEngineering* 4.2 (2020). ISSN: 2305-7084. DOI: 10.3390/chemengineering4020023.

[20] Xia B, Sun D-W. “Applications of computational fluid dynamics (CFD) in the food industry: a review”. In: *Computers and Electronics in Agriculture* 34.1 (2002), pp. 5–24. ISSN: 0168-1699. DOI: 10.1016/S0168-1699(01)00177-6.

[21] Anandharamakrishnan C. *Computational Fluid Dynamics Applications in Food Processing*. SpringerBriefs in Food, Health, and Nutrition. Springer, 2013. DOI: 10.1007/978-1-4614-7990-1.

[22] Basri EI, Basri AA, Riazuddin VN, Shahwir S, Mohammad Z, Ahmad K. “Computational fluid dynamics study in biomedical applications: a review”. In: *International Journal of Fluids and Heat Transfer* 1.2 (2016), pp. 2–14.

[23] Lee B-K. “Computational Fluid Dynamics in Cardiovascular Disease”. In: *Korean Circ J* 41.8 (2011), pp. 423–430. ISSN: 1738-5520. DOI: 10.4070/kcj.2011.41.8.423.

[24] *Computational Fluid Dynamics Market. Global Industry Trends, Share, Size, Growth, Opportunity and Forecast 2022-2027*. Financial market report. IMARC Group, 2022. URL: https://www.imarcgroup.com/computational-fluid-dynamics-market.

[25] Slotnick JP, Khodadoust A, Alonso J, Darmofal D, Gropp W, Lurie E, Mavriplis DJ. *CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences*. Contractor Report NASA/CR-2014-218178. National Aeronautics and Space Administration, 2014. URL: https://ntrs.nasa.gov/citations/20140003093.

[26] Ekaterinaris JA. “High-order accurate, low numerical diffusion methods for aerodynamics”. In: *Progress in Aerospace Sciences* 41.3 (2005), pp. 192–300. ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2005.03.003.

[27] Zingg D, De Rango S, Nemec M, Pulliam T. “Comparison of Several Spatial Discretizations for the Navier-Stokes Equations”. In: *Journal of Computational Physics* 160.2 (2000), pp. 683–704. ISSN: 0021-9991. DOI: 10.1006/jcph.2000.6482.

[28] Visbal MR, Gaitonde DV. “On the Use of Higher-Order Finite-Difference Schemes on Curvilinear and Deforming Meshes”. In: *Journal of Computational Physics* 181.1 (2002), pp. 155–185. ISSN: 0021-9991. DOI: 10.1006/jcph.2002.7117.

[29] Mosedale A, Drikakis D. “Assessment of Very High Order of Accuracy in Implicit LES models”. In: *Journal of Fluids Engineering* 129.12 (2007), pp. 1497–1503. ISSN: 0098-2202. DOI: 10.1115/1.2801374.

[30] Vasconcelos AGR. “A Very High-Order Finite Volume Method Based on Weighted Least Squares for the Solution of Poisson Equation on Unstructured Grids”. MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2017. FENIX: 1972678479053984 [MEAER].

[31] Vasconcelos AGR, Albuquerque DMS, Pereira JCF. “A very high-order finite volume method based on weighted least squares for elliptic operators on polyhedral unstructured grids”. In: *Computers & Fluids* 181 (2019), pp. 383–402. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2019.02.004.

[32] Diogo FJM. “A Very High-Order Finite Volume Technique for Convection-Diffusion Problems on Unstructured Grids”. MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2019. FENIX: 283828618790445 [MEAER].

[33] Costa PMP. “A Novel Approach for Temporal Discretizations with High-order Finite Volume Schemes”. MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2019. FENIX: 1972678479054891 [MEMEC].

[34] Costa PMP, Albuquerque DMS. “A novel approach for temporal simulations with very high-order finite volume schemes on polyhedral unstructured grids”. In: *Journal of Computational Physics* 453 (2022), p. 110960. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2022.110960.

[35] Cardiff P, Demirdžić I. “Thirty Years of the Finite Volume Method for Solid Mechanics”. In: *Archives of Computational Methods in Engineering* 28 (5 2021), pp. 3721–3780. DOI: 10.1007/s11831-020-09523-0.

[36] Gosman AD, Pun WM, Runchal AK, Spalding DB, Wolfshtein M. *Heat and Mass Transfer in Recirculating Flows*. London: Academic Press, 1969.

[37] Patankar SV, Spalding DB. *Heat and Mass Transfer in Boundary Layers*. London: Morgan-Grampian, 1967.

[38] Barth T, Frederickson P. “Higher order solution of the Euler equations on unstructured grids using quadratic reconstruction”. In: *28th Aerospace Sciences Meeting*. DOI: 10.2514/6.1990-13.

[39] Ollivier-Gooch CF, Van Altena M. “A High-Order-Accurate Unstructured Mesh Finite-Volume Scheme for the Advection-Diffusion Equation”. In: *Journal of Computational Physics* 181.2 (2002), pp. 729–752. ISSN: 0021-9991. DOI: 10.1006/jcph.2002.7159.

[40] Nejat A, Ollivier-Gooch CF. “A high-order accurate unstructured finite volume Newton-Krylov algorithm for inviscid compressible flows”. In: *Journal of Computational Physics* 227.4 (2008), pp. 2582–2609. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2007.11.011.

[41] Michalak K, Ollivier-Gooch CF. “Limiters for Unstructured Higher-Order Accurate Solutions of the Euler Equations”. In: *46th AIAA Aerospace Sciences Meeting and Exhibit*. DOI: 10.2514/6.2008-776.

[42] Clain SL, Machado GJ, Pereira RMS. “A new very high-order finite volume method for the 2D convection-diffusion problem on unstructured meshes”. In: *IV Confererência Nacional em Mecânica dos Fluidos, Termodinâmica e Energia*. 2012. HAL: hal-00675743.

[43] Costa R, Clain S, Machado GJ. “New cell-vertex reconstruction for finite volume scheme: Application to the convection-diffusion-reaction equation”. In: *Computers & Mathematics with Applications* 68.10 (2014), pp. 1229–1249. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2014.08.015.

[44] Costa R, Clain S, Machado GJ. “A sixth-order finite volume scheme for the steady-state incompressible Stokes equations on staggered unstructured meshes”. In: *Journal of Computational Physics* 349 (2017), pp. 501–527. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2017.07.047.

[45] Wang Z. “High-order methods for the Euler and Navier-Stokes equations on unstructured grids”. In: *Progress in Aerospace Sciences* 43.1 (2007), pp. 1–41. ISSN: 0376-0421. DOI: 10.1016/j.paerosci.2007.05.001.

[46] Ollivier-Gooch CF. “High-order ENO schemes for unstructured meshes based on least-squares reconstruction”. In: *35th AIAA Aerospace Sciences Meeting and Exhibit*. 1997. DOI: 10.2514/6.1997-540.

[47] Friedrich O. “Weighted Essentially Non-Oscillatory Schemes for the Interpolation of Mean Values on Unstructured Grids”. In: *Journal of Computational Physics* 144.1 (1998), pp. 194–212. ISSN: 0021-9991. DOI: 10.1006/jcph.1998.5988.

[48] Hu C, Shu C-W. “Weighted Essentially Non-oscillatory Schemes on Triangular Meshes”. In: *Journal of Computational Physics* 150.1 (1999), pp. 97–127. ISSN: 0021-9991. DOI: 10.1006/jcph.1998.6165.

[49] Ivan L, Groth CP. “High-order solution-adaptive central essentially non-oscillatory (CENO) method for viscous flows”. In: *Journal of Computational Physics* 257 (2014), pp. 830–862. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2013.09.045.

[50] Fu L, Hu XY, Adams NA. “A new class of adaptive high-order targeted ENO schemes for hyperbolic conservation laws”. In: *Journal of Computational Physics* 374 (2018), pp. 724–751. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2018.07.043.

[51] Cueto-Felgueroso L, Colominas I, Fe J, Navarrina F, Casteleiro M. “High-order finite volume schemes on unstructured grids using moving least-squares reconstruction. Application to shallow water dynamics”. In: *International journal for numerical methods in engineering* 65.3 (2006), pp. 295–331.

[52] Cueto-Felgueroso L, Colominas I, Nogueira X, Navarrina F, Casteleiro M. “Finite volume solvers and Moving Least-Squares approximations for the compressible Navier-Stokes equations on unstructured grids”. In: *Computer Methods in Applied Mechanics and Engineering* 196.45 (2007), pp. 4712–4736. ISSN: 0045-7825. DOI: 10.1016/j.cma.2007.06.003.

[53] Nogueira X, Cueto-Felgueroso L, Colominas I, Gómez H, Navarrina F, Casteleiro M. “On the accuracy of finite volume and discontinuous Galerkin discretizations for compressible flow on unstructured grids”. In: *International Journal for Numerical Methods in Engineering* 78.13 (2009), pp. 1553–1584. DOI: 10.1002/nme.2538.

[54] *OpenMP Application Programming Interface*. Specification v5.2. OpenMP Architecture Review Board, 2021.

[55] *MPI: A Message-Passing Interface Standard*. Specification version 4.0. Message Passing Interface Forum, 2021.

[56] Sterling T, Anderson M, Brodowicz M. “The Essential MPI”. In: *High Performance Computing: Modern Systems and Practices*. Morgan Kaufmann, 2018. Chap. 8, pp. 249–284. ISBN: 978-0-12-420158-3. DOI: 10.1016/B978-0-12-420158-3.00008-3.

[57] Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D. *LAPACK Users' Guide*. 3rd. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8.

[58] Balay S, Abhyankar S, Adams MF, Benson S, Brown J, Brune P, Buschelman K, Constantinescu E, Dalcin L, Dener A, Eijkhout V, Gropp WD, Hapla V, Isaac T, Jolivet P, Karpeev D, Kaushik D, Knepley MG, Kong F, Kruger S, May DA, McInnes LC, Mills RT, Mitchell L, Munson T, Roman JE, Rupp K, Sanan P, Sarich J, Smith BF, Zampini S, Zhang H, Zhang H, Zhang J. *PETSc/TAO Users Manual*. Tech. rep. ANL-21/39 - Revision 3.17. Argonne National Laboratory, 2022.

[59] Balay S, Gropp WD, McInnes LC, Smith BF. "Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries". In: *Modern Software Tools in Scientific Computing*. Ed. by Arge E, Bruaset AM, Langtangen HP. Birkhäuser Press, 1997, pp. 163–202. DOI: 10.1007/978-1-4612-1986-6_8.

[60] Fettig J, Kwok W-Y, Saied F. "Scaling Behavior of Linear Solvers on Large Linux Clusters". 2002.

[61] Sood K, Norris B, Jessup E. "Comparative Performance Modeling of Parallel Preconditioned Krylov Methods". In: *2017 IEEE 19th International Conference on High Performance Computing and Communications*. 2017, pp. 26–33. DOI: 10.1109/HPCC-SmartCity-DSS.2017.4.

[62] Silva NFL da. "Parallel Implementation of Data Balancing Algorithms". MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2016. FENIX: 283828618789632 [MEIC-A].

[63] Younas M. "Scalable, Parallel Poisson Solvers for CFD Problems". PhD thesis. University of Groningen, 2012. ISBN: 9789036753678.

[64] Eller PR, Cheng J-RC, Nguyen HV, Maier RS. "Improving parallel performance of large-scale watershed simulations". In: *Procedia Computer Science* 1.1 (2010). ICCS 2010, pp. 801–808. ISSN: 1877-0509. DOI: 10.1016/j.procs.2010.04.086.

[65] Lange M, Gorman G, Weiland M, Mitchell L, Southern J. "Achieving Efficient Strong Scaling with PETSc Using Hybrid MPI/OpenMP Optimisation". In: *Supercomputing*. Ed. by Kunkel JM, Ludwig T, Meuer HW. Berlin, Heidelberg: Springer, 2013, pp. 97–108. ISBN: 978-3-642-38750-0.

[66] Lange M, Gorman G, Weiland M, Mitchell L, Guo X, Southern J. "Benchmarking mixed-mode PETSc performance on high-performance architectures". 2013. arXiv: 1307.4567 [cs.DC].

[67] Weiland M, Mitchell L, Gorman G, Kramer S, Parsons M, Southern J. "Mixed-mode implementation of PETSc for scalable linear algebra on multi-core processors". 2012. arXiv: 1205.2005 [cs.DC].

[68] Gorobets A, Soukov S, Bogdanov P. "Multilevel parallelization for simulating compressible turbulent flows on most kinds of hybrid supercomputers". In: *Computers & Fluids* 173 (2018), pp. 171–177. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2018.03.011.

[69] Tsoutsanis P, Antoniadis AF, Jenkins KW. "Improvement of the computational performance of a parallel unstructured WENO finite volume CFD code for Implicit Large Eddy Simulation". In: *Computers & Fluids* 173 (2018), pp. 157–170. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2018.03.012.

[70] Freret L, Groth CP, Nguyen TB, Sterck HD. "High-Order Finite-Volume Scheme with Anisotropic Adaptive Mesh Refinement: Efficient Inexact Newton Method for Steady Three-Dimensional Flows". In: *23rd AIAA Computational Fluid Dynamics Conference*. 2017. DOI: 10.2514/6.2017-3297.

[71] He X, Wang K, Feng Y, Lv L, Liu T. "An implementation of MPI and hybrid OpenMP/MPI parallelization strategies for an implicit 3D DDG solver". In: *Computers & Fluids* 241 (2022), p. 105455. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2022.105455.

[72] Yan Z, Chen R, Cai X-C. "Large eddy simulation of the wind flow in a realistic full-scale urban community with a scalable parallel algorithm". In: *Computer Physics Communications* 270 (2022), p. 108170. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2021.108170.

[73] Dürrwächter J, Meyer F, Kuhn T, Beck A, Munz C-D, Rohde C. “A high-order stochastic Galerkin code for the compressible Euler and Navier-Stokes equations”. In: *Computers & Fluids* 228 (2021), p. 105039. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2021.105039.

[74] Houba T, Dasgupta A, Gopalakrishnan S, Gosse R, Roy S. “Supersonic turbulent flow simulation using a scalable parallel modal discontinuous Galerkin numerical method”. In: *Scientific Reports* 9.14442 (2019). DOI: 10.1038/s41598-019-50546-w.

[75] Ouro P, Lopez-Novoa U, Guest MF. “On the performance of a highly-scalable Computational Fluid Dynamics code on AMD, ARM and Intel processor-based HPC systems”. In: *Computer Physics Communications* 269 (2021), p. 108105. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2021.108105.

[76] Junqueira-Junior C, Azevedo JLF, Panetta J, Wolf WR, Yamouni S. “On the scalability of CFD tool for supersonic jet flow configurations”. In: *Parallel Computing* 93 (2020), p. 102620. ISSN: 0167-8191. DOI: 10.1016/j.parco.2020.102620.

[77] Tsoutsanis P, Antoniadis AF, Drikakis D. “WENO schemes on arbitrary unstructured meshes for laminar, transitional and turbulent flows”. In: *Journal of Computational Physics* 256 (2014), pp. 254–276. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2013.09.002.

[78] Costa R, Nóbrega JM, Clain S, Machado GJ. “Efficient very high-order accurate polyhedral mesh finite volume scheme for 3D conjugate heat transfer problems in curved domains”. In: *Journal of Computational Physics* 445 (2021), p. 110604. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2021.110604.

[79] Tsoutsanis P, Drikakis D. “Addressing the challenges of implementation of high-order finite-volume schemes for atmospheric dynamics on unstructured meshes”. In: *VII European Congress on Computational Methods in Applied Sciences and Engineering - ECCOMAS*. 2016. DOI: 10.7712/100016.1846.8406.

[80] Antoniadis AF, Drikakis D, Farmakis PS, Fu L, Kokkinakis I, Nogueira X, Silva PA, Skote M, Titarev V, Tsoutsanis P. “UCNS3D: An open-source high-order finite-volume unstructured CFD solver”. In: *Computer Physics Communications* 279 (2022), p. 108453. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2022.108453.

[81] Pei W, Jiang Y, Li S. “An Efficient Parallel Implementation of the Runge-Kutta Discontinuous Galerkin Method with Weighted Essentially Non-Oscillatory Limiters on Three-Dimensional Unstructured Meshes”. In: *Applied Sciences* 12.9 (2022). ISSN: 2076-3417. DOI: 10.3390/app12094228.

[82] Guermond J-L, Kronbichler M, Maier M, Popov B, Tomas I. “On the implementation of a robust and efficient finite element-based parallel solver for the compressible Navier-Stokes equations”. In: *Computer Methods in Applied Mechanics and Engineering* 389 (2022), p. 114250. ISSN: 0045-7825. DOI: 10.1016/j.cma.2021.114250.

[83] Kronbichler M, Fehn N, Munch P, Bergbauer M, Wichmann K-R, Geitner C, Allalen M, Schulz M, Wall WA. “A Next-Generation Discontinuous Galerkin Fluid Dynamics Solver with Application to High-Resolution Lung Airflow Simulations”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’21. New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 9781450384421. DOI: 10.1145/3458817.3476171.

[84] Moukalled F, Mangani L, Darwish M. *The Finite Volume Method in Computational Fluid Dynamics. An Advanced Introduction with OpenFOAM® and Matlab®*. Springer, 2016. ISBN: 978-3-319-16874-6. DOI: 10.1007/978-3-319-16874-6.

[85] Krommer AR, Ueberhuber CW. *Fundamentals of Computational Integration*. Society for Industrial and Applied Mathematics, 1998. Chap. 3, p. 49. ISBN: 0-89871-374-9.

[86] *Wolfram|Alpha: Computational Intelligence*. Wolfram. URL: https://www.wolframalpha.com/ (visited on 2021).

[87] Abramowitz M, Stegun IA, eds. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. National Bureau of Standards. ISBN: 0-486-61272-4.

[88] Dunavant DA. "High degree efficient symmetrical Gaussian quadrature rules for the triangle". In: *International Journal for Numerical Methods in Engineering* 21.6 (1985), pp. 1129–1148. DOI: 10.1002/nme.1620210612.

[89] Stroud AH. *Approximate Calculation of Multiple Integrals*. Ed. by Forsythe G. Automatic Computation. Englewood Cliffs, NJ: Prentice-Hall, 1971. ISBN: 13-043893-6.

[90] Cools R. "An encyclopaedia of cubature formulas". In: *Journal of Complexity* 19.3 (2003). Oberwolfach Special Issue, pp. 445–453. ISSN: 0885-064X. DOI: 10.1016/S0885-064X(03)00011-6.

[91] Peterson JW. *Analytical Formulae for Two of A. H. Stroud's Quadrature Rules*. 2009. arXiv: 0909.5106 [math.NA].

[92] Ning A. *Scientific Programming Languages*. Flight, Optimization, and Wind Laboratory. 2015. URL: https://flow.byu.edu/posts/sci-prog-lang (visited on 02/24/2022).

[93] Engeln-Müllges G, Uhlig F. *Numerical Algorithms with C*. Berlin: Springer, 1996. ISBN: 978-3-540-60530-0.

[94] Quinn MJ. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2004. ISBN: 0-07-282256-2.

[95] Freeman TL, Phillips C. *Parallel Numerical Algorithms*. Prentice Hall, 1992. ISBN: 0-13-651597-5.

[96] Amdahl GM. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS '67 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1967, pp. 483–485. ISBN: 9781450378956. DOI: 10.1145/1465482.1465560.

[97] Gustafson JL. "Reevaluating Amdahl's Law". In: *Commun. ACM* 31.5 (1988), pp. 532–533. ISSN: 0001-0782. DOI: 10.1145/42411.42415.

[98] Overton ML. *Numerical computing with IEEE floating point arithmetic*. Society for Industrial and Applied Mathematics, 2001. ISBN: 0-89871-571-7.

[99] *IEEE Standard for Floating-Point Arithmetic*. IEEE Std 754-2019 (Revision of IEEE 754-2008). Institute for Electrical and Electronics Engineers, 2019. DOI: 10.1109/IEEESTD.2019.8766229.

[100] Kong Q, Siauw T, Bayen AM. "Representation of Numbers". In: *Python Programming and Numerical Methods*. Academic Press, 2021. Chap. 9, pp. 145–156. ISBN: 978-0-12-819549-9. DOI: 10.1016/B978-0-12-819549-9.00018-X.

[101] Goldberg D. "What Every Computer Scientist Should Know about Floating-Point Arithmetic". In: *ACM Comput. Surv.* 23.1 (1991), pp. 5–48. ISSN: 0360-0300. DOI: 10.1145/103162.103163.

[102] Chapra SC. "Roundoff and Truncation Errors". In: *Applied Numerical Methods with MATLAB for Engineers and Scientists*. 4th. McGraw-Hill, 2018. Chap. 4. ISBN: 978-0-07-339796-2.

[103] *High Performance Computing da Universidade de Évora*. Rede National de Computação Avançada. URL: https://rnca.fccn.pt/hpc-ue/ (visited on 07/2022).

[104] *Resources*. Minho Advanced Computing Center. URL: https://macc.fccn.pt/resources (visited on 10/2022).

[105] Costa P. "A FFT-accelerated multi-block finite-difference solver for massively parallel simulations of incompressible flows". In: *Computer Physics Communications* 271 (2022), p. 108194. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2021.108194.

Appendix A

Code Documentation

A.1 List of HIBforMBP Git Repositories

As part of the work done in the context of the HIBforMBP Project, the Author sought to organize the codes used and developed by the group for ease of reference and future use. As such, these were uploaded into public Git repositories hosted by GitLab, under the HIBforMBP group umbrella. The web address of all these repositories are of the form `https://gitlab.com/hibformbp/<NAME>`, where `<NAME>` is the name of the repository. This appendix lists these repositories and provides a short summary of the included contents.

`dif-2d-unstruct` Contains the original Matlab code developed by Vasconcelos [1] to solve two-dimensional Poisson problems. The code allows for import and use of an external grid, or the user may choose to generate a Cartesian grid using the built-in functionality.

`conv-dif-2d-unstruct` Contains both the original Matlab code developed by Diogo [2] and a computationally improved version developed by the present Author using Matlab vectorization. This code solves two-dimensional convection-diffusion problems on any grid by importing and using an external grid. Alternatively, the user may choose to generate a Cartesian grid using the built-in functionality.

`conv-dif-2d-cart` Contains the parallel solver for two-dimensional, convection-diffusion problems in Cartesian grids, developed for the current work. The code is written in C, using the PETSc library for the linear-algebra and parallel computing framework. Besides the base FLS schemes, this solver also includes additional methods for the regression polynomial – namely, extended polynomial and direct coefficients.

`conv-dif-3d-cart` Contains the parallel solver for three-dimensional, convection-diffusion problems in Cartesian grids, developed for the current work. Besides the base FLS schemes, this solver also includes some FFT capabilities (see Appendix C), which are not yet completely implemented. As with the 2D solver, the code is written in C, using PETSc for the linear-algebra and parallel framework; furthermore, the FFTW library was used for the FFT operations.

HIBforMBP References

[1] Vasconcelos AGR. “A Very High-Order Finite Volume Method Based on Weighted Least Squares for the Solution of Poisson Equation on Unstructured Grids”. MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2017. FENIX: 1972678479053984 [MEAER].

[2] Diogo FJM. “A Very High-Order Finite Volume Technique for Convection-Diffusion Problems on Unstructured Grids”. MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2019. FENIX: 283828618790445 [MEAER].

[3] Costa PMP. “A Novel Approach for Temporal Discretizations with High-order Finite Volume Schemes”. MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2019. FENIX: 1972678479054891 [MEMEC].

[4] Vasconcelos AGR, Albuquerque DMS, Pereira JCF. “A very high-order finite volume method based on weighted least squares for elliptic operators on polyhedral unstructured grids”. In: *Computers & Fluids* 181 (2019), pp. 383–402. ISSN: 0045-7930. DOI: 10.1016/j.compfluid.2019.02.004.

[5] Costa PMP, Albuquerque DMS. “A novel approach for temporal simulations with very high-order finite volume schemes on polyhedral unstructured grids”. In: *Journal of Computational Physics* 453 (2022), p. 110960. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2022.110960.

Appendix B

Selected Results in Tabular Form

This appendix lists, in tabular form, results that have been presented as plots in Chapter 3. Note that the following tables only list primary-type metrics (e.g. error norms and runtime) instead of any derived-type metrics (e.g. convergence order and parallel efficiency) that may be calculated from the corresponding measurements. The main motivation for this appendix is to facilitate any future reproduction or verification of results obtained in the current work.

B.1 Numerical Error Norms

Tables B.1 and B.2 present the numerical error norms $\|\varepsilon\|_1$ and $\|\varepsilon\|_\infty$ for, respectively, two- and three-dimensional face least-squares (FLS) schemes, using the analytical solution given by (3.1) with Dirichlet boundary conditions. Both of these results were obtained using quadruple precision, to delay the onset of contamination by round-off errors. The corresponding linear systems were solved using the biconjugate gradient stabilized method (BiCGSTAB) solver, with the 2D cases being preconditioned with block-Jacobi and the 3D ones with 0.02-threshold, geometric algebraic multigrid (GAMG).

Table B.3 compares the numerical error norm $\|\varepsilon\|_1$ of 2D FLS schemes using the standard or extended regression polynomial, as explained in Section 3.2.1. On the other hand, Table B.4 compares the numerical error norm of 2D FLS schemes with polynomial coefficients computed directly or using the standard WLS method, as explained in Section 3.2.2. Both results were obtained using BiCGSTAB solver preconditioned with block-Jacobi, for the analytical solution (3.2) with periodic boundary conditions.

Table B.1 Numerical error norm $\|\varepsilon\|_1$ and $\|\varepsilon\|_\infty$ for two-dimensional FLS schemes, using the sinusoidal analytical solution given by (3.1) with Dirichlet boundary conditions. Results were obtained from BiCGSTAB preconditioned with block-Jacobi, using quadruple precision. Corresponds to Figure 3.2.

N_C	FLS2		FLS4		FLS6		FLS8	
	$\|\varepsilon\|_1$	$\|\varepsilon\|_\infty$	$\|\varepsilon\|_1$	$\|\varepsilon\|_\infty$	$\|\varepsilon\|_1$	$\|\varepsilon\|_\infty$	$\|\varepsilon\|_1$	$\|\varepsilon\|_\infty$
10^2	3.777278 E–2	8.532236 E–2	4.487545 E–3	1.604884 E–2	1.189534 E–3	3.224379 E–3	9.298174 E–4	3.559529 E–3
20^2	8.777496 E–3	2.131917 E–2	1.990097 E–4	6.845547 E–4	9.746955 E–6	5.172965 E–5	7.926631 E–6	2.851207 E–5
40^2	2.124171 E–3	5.236612 E–3	7.724761 E–6	2.302125 E–5	9.742007 E–8	1.303967 E–6	9.244221 E–9	6.381465 E–8
80^2	5.233854 E–4	1.304881 E–3	3.073132 E–7	1.449615 E–6	1.136594 E–9	2.325380 E–8	2.075563 E–11	2.599164 E–10
160^2	1.298428 E–4	3.249887 E–4	1.429156 E–8	9.572314 E–8	1.582411 E–11	3.751032 E–10	4.711389 E–14	1.025197 E–12
240^2	5.757970 E–5	1.442565 E–4	2.572985 E–9	1.909526 E–8	1.354368 E–12	3.312153 E–11	2.376432 E–15	4.006741 E–14
320^2	3.235251 E–5	8.111442 E–5	7.864716 E–10	6.062065 E–9	2.386211 E–13	5.906673 E–12	1.210030 E–15	4.011987 E–15
480^2	1.436296 E–5	3.602750 E–5	1.520640 E–10	1.200174 E–9	2.053468 E–14	5.192810 E–13	1.113236 E–15	2.746082 E–15
640^2	8.074869 E–6	2.025961 E–5	4.790813 E–11	3.800225 E–10	3.408324 E–15	9.246497 E–14	1.110463 E–15	2.739940 E–15

Table B.2 Numerical error norm $\|\varepsilon\|_1$ and $\|\varepsilon\|_\infty$ for three-dimensional FLS schemes, using the sinusoidal analytical solution given by (3.1) with Dirichlet boundary conditions. Results obtained from BiCGSTAB preconditioned with 0.02-threshold GAMG, using quadruple precision. Corresponds to Figure 3.4.

N_C	FLS2		FLS4		FLS6		FLS8	
	$\|\varepsilon\|_1$	$\|\varepsilon\|_\infty$	$\|\varepsilon\|_1$	$\|\varepsilon\|_\infty$	$\|\varepsilon\|_1$	$\|\varepsilon\|_\infty$	$\|\varepsilon\|_1$	$\|\varepsilon\|_\infty$
10^3	2.971106 E–2	9.985035 E–2	7.601767 E–3	2.527045 E–2	1.429892 E–3	7.056045 E–3	1.430996 E–3	4.470106 E–3
20^3	6.732156 E–3	2.438771 E–2	4.284892 E–4	1.465967 E–3	2.222095 E–5	1.743542 E–4	7.820208 E–6	6.108891 E–5
40^3	1.593045 E–3	5.976863 E–3	2.270009 E–5	7.764135 E–5	3.195246 E–7	3.590689 E–6	1.660670 E–8	1.637591 E–7
80^3	3.862377 E–4	1.480327 E–3	1.210016 E–6	4.352053 E–6	4.025959 E–9	6.006006 E–8	4.496722 E–11	8.141603 E–10
160^3	9.500592 E–5	3.671997 E–4	6.835786 E–8	2.570170 E–7	5.293032 E–11	9.537420 E–10	1.144942 E–13	3.628813 E–12
240^3	4.198506 E–5	1.630484 E–4	1.299856 E–8	4.980098 E–8	4.333037 E–12	8.395048 E–11	–	–
320^3	2.355153 E–5	9.159198 E–5	4.032487 E–9	1.560531 E–8	7.427116 E–13	1.495764 E–11	–	–
480^3	1.043876 E–5	4.066953 E–5	7.805182 E–10	3.052640 E–9	–	–	–	–
640^3	5.863887 E–6	2.286515 E–5	–	–	–	–	–	–

Table B.3 Comparison between numerical error norm $\|\varepsilon\|_1$ of two-dimensional FLS schemes using the standard or extended regression polynomial, for the analytical solution given by (3.2) with periodic boundary conditions. Results were obtained from BiCGSTAB preconditioned with block-Jacobi, using double precision. Corresponds to Figure 3.6.

N_C	FLS2		FLS4		FLS6		FLS8	
	$\|\varepsilon\|_1^{\text{std}}$	$\|\varepsilon\|_1^{\text{ext}}$	$\|\varepsilon\|_1^{\text{std}}$	$\|\varepsilon\|_1^{\text{ext}}$	$\|\varepsilon\|_1^{\text{std}}$	$\|\varepsilon\|_1^{\text{ext}}$	$\|\varepsilon\|_1^{\text{std}}$	$\|\varepsilon\|_1^{\text{ext}}$
10^2	2.966156 E–2	2.967756 E–2	5.166213 E–4	3.680766 E–4	6.345924 E–5	3.632408 E–5	6.844690 E–6	3.288953 E–6
20^2	7.456088 E–3	7.433270 E–3	3.396957 E–5	2.376566 E–5	1.069726 E–6	6.035123 E–7	2.966036 E–8	1.408858 E–8
40^2	1.853241 E–3	1.850208 E–3	2.144876 E–6	1.498189 E–6	1.702547 E–8	9.578087 E–9	1.189372 E–10	5.626748 E–11
80^2	4.639462 E–4	4.629621 E–4	1.344689 E–7	9.385523 E–8	2.673842 E–10	1.502733 E–10	4.648162 E–13	2.008589 E–13
160^2	1.159361 E–4	1.156968 E–4	8.410869 E–9	5.869564 E–9	4.209227 E–12	2.339797 E–12	6.746729 E–14	1.305138 E–13
240^2	5.152492 E–5	5.142004 E–5	1.661583 E–9	1.159498 E–9	4.248937 E–13	3.531481 E–13	2.779259 E–13	5.995041 E–14
320^2	2.898242 E–5	2.892351 E–5	–	3.669113 E–10	2.177108 E–13	6.106207 E–13	4.558117 E–13	1.849039 E–13
480^2	1.288095 E–5	1.285476 E–5	1.038512 E–10	7.220526 E–11	7.181719 E–13	1.764908 E–12	2.760230 E–13	–
640^2	7.245511 E–6	7.230777 E–6	3.273364 E–11	2.246034 E–11	1.775129 E–12	9.316299 E–10	–	1.881223 E–12

Table B.4 Comparison between numerical error norm $\|\varepsilon\|_1$ of two-dimensional FLS schemes with regression polynomial coefficients calculated using the standard WLS method or directly, for the analytical solution given by (3.2) with periodic boundary conditions. Results were obtained from BiCGSTAB preconditioned with block-Jacobi, using double precision. Corresponds to Figure 3.8.

N_C	FLS2		FLS4		FLS6		FLS8	
	$\|\varepsilon\|_1^{\text{wls}}$	$\|\varepsilon\|_1^{\text{direct}}$	$\|\varepsilon\|_1^{\text{wls}}$	$\|\varepsilon\|_1^{\text{direct}}$	$\|\varepsilon\|_1^{\text{wls}}$	$\|\varepsilon\|_1^{\text{direct}}$	$\|\varepsilon\|_1^{\text{wls}}$	$\|\varepsilon\|_1^{\text{direct}}$
10^2	2.966156 E–2	2.712610 E–2	5.166213 E–4	6.183625 E–4	6.345924 E–5	4.106365 E–5	6.844690 E–6	3.065137 E–6
20^2	7.456088 E–3	6.835835 E–3	3.396957 E–5	4.073754 E–5	1.069726 E–6	6.881694 E–7	2.966036 E–8	1.304510 E–8
40^2	1.853241 E–3	1.696763 E–3	2.144876 E–6	2.577670 E–6	1.702547 E–8	1.093757 E–8	1.189372 E–10	5.213950 E–11
80^2	4.639462 E–4	4.248207 E–4	1.344689 E–7	1.615782 E–7	2.673842 E–10	1.715824 E–10	4.648162 E–13	2.908276 E–13
160^2	1.159361 E–4	1.061556 E–4	8.410869 E–9	1.010601 E–8	4.209227 E–12	2.665925 E–12	6.746729 E–14	5.960253 E–13
240^2	5.152492 E–5	4.717745 E–5	1.661583 E–9	–	4.248937 E–13	2.077695 E–13	2.779259 E–13	5.882923 E–13
320^2	2.898242 E–5	2.653687 E–5	–	6.312916 E–10	2.177108 E–13	4.767544 E–13	4.558117 E–13	4.728013 E–13
480^2	1.288095 E–5	1.179403 E–5	1.038512 E–10	1.248379 E–10	–	9.785511 E–13	2.760230 E–13	1.177561 E–12
640^2	7.245511 E–6	6.634121 E–6	3.273364 E–11	3.795285 E–11	–	9.955417 E–13	–	1.471543 E–12

B.2 Solver Runtimes

Tables B.5 and B.6 present the solver runtime (SRT) from strong- and weak-scaling test, respectively, performed at the Oblivion supercomputer with up to 128 processes. Each table lists results from two battery of tests, which differ in the chosen comparison point – i.e. a reference starting mesh size or a reference starting SRT. The tests used the three-dimensional FLS schemes with the analytical solution given by (3.1), with Dirichlet boundary conditions. Results were obtained using the BiCGSTAB solver preconditioned with 0.02-threshold GAMG, using double precision.

Table B.5 Solver runtime (SRT) from strong-scaling tests performed at the Oblivion supercomputer, using two different comparison points – reference mesh size and reference starting SRT. Corresponds to Figure 3.9 (left) and Figure 3.10 (left).

N_p	Ref. $N_C = 80^3$				Ref. $SRT\|_{N_p=1} \approx 40\,\mathrm{s}$			
	FLS2	FLS4	FLS6	FLS8	FLS2	FLS4	FLS6	FLS8
1	4.1012	11.886	21.786	35.374	41.307	48.326	44.475	35.374
2	2.3166	7.7050	12.414	19.958	23.481	28.012	29.403	19.958
4	1.2634	3.1940	7.0541	9.8499	11.609	15.918	14.724	9.8499
8	0.57855	1.5882	3.4587	4.8949	5.8758	7.9251	7.3124	4.8949
16	0.22028	0.70712	1.4600	2.4359	2.7817	3.9586	4.1820	2.4359
32	0.10362	0.29634	0.79645	1.1835	1.3265	1.8902	1.7789	1.1835
64	0.05555	0.16614	0.40542	0.62198	0.70810	0.97076	1.0504	0.62198
128	0.03420	0.08527	0.22032	0.38539	0.37944	0.50544	0.56084	0.38539

Table B.6 Solver runtime (SRT) from weak-scaling tests performed at the Oblivion supercomputer, using two different comparison points – reference starting mesh size and reference starting SRT. Corresponds to Figure 3.9 (right) and Figure 3.10 (right).

N_p	Ref. $N_C = 80^3$				Ref. $SRT\|_{N_p=1} \approx 20\,\mathrm{s}$			
	FLS2	FLS4	FLS6	FLS8	FLS2	FLS4	FLS6	FLS8
1	4.1012	11.886	21.786	35.374	20.179	23.988	21.786	16.879
2	4.8783	13.802	29.403	41.400	23.481	32.475	29.403	19.958
4	5.8798	15.918	30.483	47.998	24.268	32.799	30.483	23.609
8	5.8758	15.934	29.677	46.295	25.040	32.435	29.677	23.692
16	5.9271	16.689	30.952	48.350	24.308	33.042	30.952	23.556
32	6.0016	16.704	30.512	48.915	24.516	32.778	30.512	24.273
64	6.2324	17.286	31.521	50.876	25.577	35.053	31.521	25.609
128	6.6232	18.460	34.203	–	27.311	37.176	34.203	26.439

Appendix C

FFT-Based Acceleration of Numerical Schemes

This appendix, which consists of previous work done by the Author for the university course *Project in Aerospace Engineering*, presents acceleration approaches – based on fast Fourier transforms (FFTs) – to the numerical solution of partial differential equations (PDEs). The numerical methods presented will be applied to the diffusion equation (C.1) and to the convection-diffusion equation (C.2). These two are given here in their steady form in terms of conservation quantity φ, with Q being a source term and Γ, $\vec{v}$ the diffusion and convection coefficients, respectively.

$$\nabla \cdot (\Gamma \nabla \varphi) = Q \tag{C.1}$$

$$\nabla \cdot (\vec{v}\varphi) - \nabla \cdot (\Gamma \nabla \varphi) = Q \tag{C.2}$$

This previous work presented an FFT-based acceleration for the numerical solutions of equations (C.1) and (C.2). This method, presented in Section C.1.1, was based on the recent work by Costa [1, 2], who has applied it only for Poisson's equation under the finite difference method (FDM) framework. The section also presents two approaches for adapting the method to convection-diffusion equations, by treating convection terms implicitly or explicitly.

Furthermore, the Author studied different approaches of combining this method to the face least-squares (FLS) schemes in Section C.2. An implementation of FFT acceleration into the second-order FLS (FLS2) was derived in detail, but it remains unclear if this same derivation may be adapted to higher-order methods. Additionally, a brief discussion regarding boundary conditions and related issues was also included.

Lastly, the deferred-correction approach was briefly introduced in Section C.3. The approach was successfully implemented for the second-order central differences (CD2) scheme and FLS schemes of all orders, significantly improving upon past literature results. However, the approach was unsuccessful when using the FFT-accelerated CD2.

It is important to note that, except for Section C.1.1, this work presented novel investigation results, which are not available in literature. All of developed code is available online under the "FFT" branch of Git repository hosted at `gitlab.com/hibformbp/conv-dif-3d-cart`.

C.1 FFT Acceleration in an FDM Framework

C.1.1 Poisson Problems

Consider Poisson's equation, given by (C.3) – which is identical to (C.1) except that Γ, being constant throughout space, has been incorporated into the source term $f \equiv Q/\Gamma$.

$$\nabla^2 \varphi = f \tag{C.3}$$

The method for using a FFT to accelerate the solution of Poisson problems explained here follows the work of Costa [2] in the FDM framework. The usual CD2 expression for the second derivative of a function ϕ is given by

$$\frac{d^2}{dx^2}\phi(x) = \frac{\phi(x-\Delta x) - 2\phi(x) + \phi(x+\Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2) \tag{C.4}$$

where $\mathcal{O}(\Delta x^2)$ represent the higher-order terms. Since the expression will be used numerically, it is useful to consider a discrete representation $\phi_i \equiv \phi(x)$, $\phi_{i\pm1} \equiv \phi(x \pm \Delta x)$, where Δx now represents the grid spacing along x. Thus, performing the discretization of (C.3) using CD2 in all three dimensions yields the following equation, which represents a linear system described by a matrix with seven non-zero diagonals:

$$\frac{\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j,k}}{\Delta x^2} + \frac{\varphi_{i,j-1,k} - 2\varphi_{i,j,k} + \varphi_{i,j+1,k}}{\Delta y^2} + \frac{\varphi_{i,j,k-1} - 2\varphi_{i,j,k} + \varphi_{i,j,k+1}}{\Delta z^2} = f_{i,j,k} \tag{C.5}$$

Using the method of eigenfunctions expansions, it is possible to reduce the complexity of the above system by decoupling along an arbitrary dimension. This "Fourier synthesis" described by Schumann and Sweet [3] is achieved by using a Fourier-based discrete expansion operator $\mathcal{F}_\zeta$ along the chosen direction ζ. Let the chosen direction be x. Schumann and Sweet [3] give the following result,

$$\mathcal{F}_x\left(\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j,k}\right) = \Lambda_i \widehat{\varphi}_{i,j,k}\,, \quad \Lambda_i \equiv -4\sin^2\frac{\kappa_i}{2n_x} \tag{C.6}$$

where $\widehat{\square} = \mathcal{F}_x(\square)$, Λ_i is the eigenvalue corresponding to grid point i along the expansion direction x, κ_i is the associated wavenumber, and n_x is the number of grid points along x. Then, applying the operator $\mathcal{F}_x$ to (C.5) results in the Helmholtz equation

$$\frac{\Lambda_i \widehat{\varphi}_{i,j,k}}{\Delta x^2} + \frac{\widehat{\varphi}_{i,j-1,k} - 2\widehat{\varphi}_{i,j,k} + \widehat{\varphi}_{i,j+1,k}}{\Delta y^2} + \frac{\widehat{\varphi}_{i,j,k-1} - 2\widehat{\varphi}_{i,j,k} + \widehat{\varphi}_{i,j,k+1}}{\Delta z^2} = \widehat{f}_{i,j,k} \tag{C.7}$$

Although Schumann and Sweet [3] give these eigenvalues in the form shown in (C.6), these may be also written in the form $2\cos(\kappa_i/n_x)$-2 by applying trigonometric identities. By using the linearity propriety of operator $\mathcal{F}$, one can rewrite the eigenfunction expansion along x and separate $\varphi_{i,j,k}$ from its neighbors. Combining these two expression, one may define a simplified eigenvalue λ_i, which represents the transformation of only the neighboring points according to (C.8).

$$\begin{aligned}
&\mathcal{F}\left(\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j,k}\right) = \Lambda_i \widehat{\varphi}_{i,j,k} = \left(2\cos\frac{\kappa_i}{n_x} - 2\right)\widehat{\varphi}_{i,j,k} \\
&\mathcal{F}\left(\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j,k}\right) = \mathcal{F}\left(\varphi_{i-1,j,k} + \varphi_{i+1,j,k}\right) - 2\widehat{\varphi}_{i,j,k} \\
\therefore\quad &\mathcal{F}\left(\varphi_{i-1,j,k} + \varphi_{i+1,j,k}\right) = \lambda_i \widehat{\varphi}_{i,j,k}, \quad \lambda_i \equiv 2\cos\frac{\kappa_i}{n_x}
\end{aligned} \tag{C.8}$$

As such, the Helmoltz equation (C.7) may be given in following alternate form

$$\frac{\lambda_i \widehat{\varphi}_{i,j,k} - 2\widehat{\varphi}_{i,j,k}}{\Delta x^2} + \frac{\widehat{\varphi}_{i,j-1,k} - 2\widehat{\varphi}_{i,j,k} + \widehat{\varphi}_{i,j+1,k}}{\Delta y^2} + \frac{\widehat{\varphi}_{i,j,k-1} - 2\widehat{\varphi}_{i,j,k} + \widehat{\varphi}_{i,j,k+1}}{\Delta z^2} = \widehat{f}_{i,j,k} \tag{C.9}$$

It is important to note that the eigenfunction expansion operator $\mathcal{F}$ and the wavenumbers κ_i both depend on the boundary conditions at the end of the chosen decoupling direction. Table C.1 lists these for different combinations of boundary conditions. Furthermore, the grid spacing in the decoupling direction must be constant.

Table C.1 Wavenumbers κ_q, forward transforms $\mathcal{F}$, and backward transforms $\mathcal{F}^{-1}$ (with normalization factor θ) used in (C.7), for different combinations of boundary conditions. (I)DFT stands for the (inverse) discrete Fourier transform, and DCT/DST for the standard types of discrete cosine/sine transforms. Note that some authors incorporate the normalization factor into the inverse transform itself; the values shown here are in accordance with the FFTW library [4].

Boundary Conditions	$\kappa_q/(2\pi)$	$\mathcal{F}$	$\theta\mathcal{F}^{-1}$	θ
Periodic	q	DFT	IDFT	n
Neumann	$q/2$	DCT-II	DCT-III	$2n$
Dirichlet	$(q+1)/2$	DST-II	DST-III	$2n$
Neumann-Dirichlet	$(2q+1)/4$	DCT-IV	DCT-IV	$2n$

Results

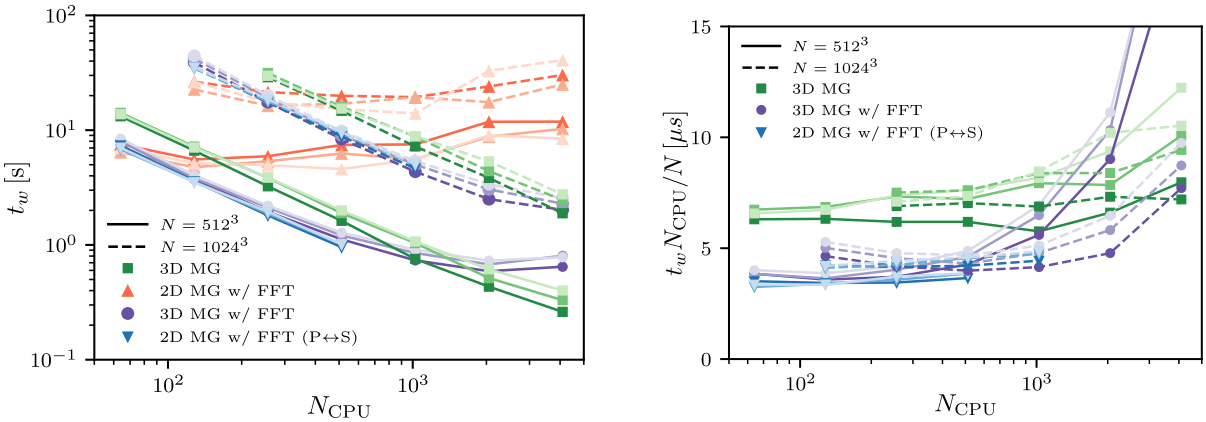
The method described above was implemented with the CD2 scheme for Poisson problems. The results are summarized in Table C.2, which show the error norm $\|\varepsilon\|_1$ and runtime for different grid sizes using both the standard and FFT-accelerated approaches. The error values serve as a verification means, showing that the FFT approach was correctly implemented. As seen from the runtime values, the FFT approach is significantly faster than the standard approach. In fact, this advantage increases with grid size, as shown by the increasing value of runtime ratio.

Table C.2 Comparison of error norm and runtime values for the standard and the FFT-accelerated CD2 schemes, using several grid sizes.

N	Error Norm $\|\varepsilon\|_1$		Runtime [s]		
	Standard	FFT	Standard	FFT	Ratio
10^3	2.022588E-02	2.022588E-02	8.48E-03	2.99E-02	0.28
20^3	4.843051E-03	4.843051E-03	6.85E-02	5.95E-02	1.15
40^3	1.197906E-03	1.197906E-03	9.48E-01	4.85E-01	1.95
80^3	2.986803E-04	2.986803E-04	1.10E+01	3.69E+00	2.98
160^3	7.462042E-05	7.462042E-05	1.52E+02	2.84E+01	5.34

Results from Literature

Costa [2] compared the computational performance of the FDM-based direct numerical simulation (DNS) solvers with and without FFT acceleration, with different decomposition approaches and decoupling directions. The results are shown in Figure C.1 for the case of lid-driven cavity flow, with increasing number of allocated central processing units (CPUs); Table C.3 gives the corresponding speedups of using an FFT-based method relative to traditional FDM. As shown in Figure C.1a and Table C.3, the methods with FFT decomposition (except for $\mathrm{2D\ MG\ w/\ FFT}$) exhibit a significant advantage over



(a) Strong scaling, given in wall-clock time

(b) CPU time per grid size

Figure C.1 Comparison of computational performance obtained by Costa [2] using different methods for a lid-driven cavity flow, with 512^3 and 1024^3 grid cells. The notation $3\mathrm{D\ MG}$ represents the traditional FDM (using multigrid) without FFT decomposition, while the other notations are methods with FFT decomposition but using different decomposition approaches. The choice of FFT-based synthesis direction is represented by increasing color lightness, in order of x, y, z respectively.

the traditional method in terms runtime, being around twofold faster. However, the performance of these methods gradually worsen and are indeed slower than the traditional method. This happens for $N_{\mathrm{CPU}} > 1024$ in the $N = 512^3$ case and for $N_{\mathrm{CPU}} > 2048$ in the $N = 1024^3$. This is even clearer in Figure C.1b, where the curve for the traditional method is approximately a flat line (representing a perfect scaling), while the curves for methods with FFT decomposition clearly exhibit upward trends. Costa notes that this decrease in parallel efficiency happens due to the load per CPU becoming too small, and that the runtime in this saturation region is already very small – meaning that one might as well use less CPUs and save computer budget.

Table C.3 Speedups obtained by Costa [2] by using different FFT-based methods (x-decomposition), for lid-driven cavity flow. The values shown here are relative to the runtime corresponding to the $3\mathrm{D\ MG}$ method.

N	Method	N_{CPU}						
		64	128	254	512	1024	2048	4096
512^3	3D MG w/ FFT	1.72	2.39	1.67	1.47	1.03	0.74	0.41
	2D MG w/ FFT (P↔S)	1.90	2.52	1.77	1.78	–	–	–
1024^3	3D MG w/ FFT	–	1.60	1.78	1.66	1.50	0.95	
	2D MG w/ FFT (P↔S)	–	1.63	1.87	1.57	–	–	

Comparison of Estimated Computational Cost

Applying a multidimensional Fourier synthesis is also possible and indeed forms the backbone of another family of numerical methods – spectral methods – which are the *de facto* standard for DNS and, to a lesser degree, large-eddy simulation (LES) [5, 6]. These methods fall outside the scope of this work, but it is nonetheless interesting to compare the theoretical computational cost between different numerical approaches. Costa [2] gives some estimates on the computational cost of these different approaches.

The use of a multidimensional Fourier synthesis reduces the matrix bandwidth to such an extent where the use of a direct solver may become more affordable than an iterative solver. Specifically for the case of CD2, the original system with seven non-zero diagonals is reduced to a tridiagonal system,

for which efficient direct solvers methods (i.e. Gauss elimination) have long existed [1]. Using a direct solver with two-dimensional Fourier synthesis, the computational cost scales with $\mathcal{O}(N \log n_2 n_3)$, where $N \equiv n_1 n_2 n_3$ is the number of grid cells and n_2, n_3 are the number of grid cells along the two decoupling directions. On the other hand, using a geometric multigrid solver, the corresponding cost scales with only $\mathcal{O}(N)$.

However, Costa notes that, for Poisson problems with $N \sim 10^9$–10^{10}, FFT-based direct solvers have instead been reported to yield much better performance – on the order of 3–10 times. This happens mainly because of two reasons. First, the hidden constant for the computational cost of direct solvers is smaller than that of the iterative solvers. Second, the term $\log n_2 n_3$ grows quite slowly when compared to N.

Costa suggests that the now-ambitious problem sizes are still orders-of-magnitude smaller than the required sizes for iterative solvers to overperform direct solvers. Given the prohibitive memory requirement for such a problem size, the current Author believes that, save a revolutionary innovation in the area, it is unlikely that such a point will be achieved in the near-future – even with the arrival of exascale high-performance computing (HPC) in this current year of 2022 [7].

C.1.2 Treatment of Convection Terms

The aforementioned method has been derived – by, among others, Schumann and Sweet [3] and Costa [2] – only for Poisson's equation. As part of the current work, the Author will now present possible approaches to expanding the method to the convection-diffusion equation.

The approach that will be shown here is to solve implicitly for the diffusion terms only, and instead treat the convection terms explicitly. This may be understood as incorporating the convection terms into the source term, thus defining a modified source term $\tilde{Q}$. This allows matrix $\mathbf{A}$ to remain with the reduced bandwidth obtained by the FFT decomposition. However, since the Fourier synthesis only removes off-diagonal terms corresponding to the decoupling direction ζ, then only the convection terms in ζ The modified convection-diffusion equation may then be written as the following, where the decoupling direction was assumed to be x,

$$\frac{\partial}{\partial y}\left(v_y \varphi\right) + \frac{\partial}{\partial z}\left(v_z \varphi\right) + \nabla \cdot \left(\Gamma \nabla \varphi\right) = \widetilde{Q}(\varphi), \qquad \text{where} \qquad \widetilde{Q}(\varphi) \equiv Q - \frac{\partial}{\partial x}\left(v_x \varphi\right). \tag{C.10}$$

However, this means that this modified source term now depends on φ itself, and the resulting algebraic system will thus be nonlinear, $\mathbf{A}\boldsymbol{\varphi} = \mathbf{b}(\boldsymbol{\varphi})$. One may linearize the system using an estimate of $\boldsymbol{\varphi}$ and update the values of $\widetilde{Q}$ by performing additional outer iterations[1]. For an outer iteration n, the linearized system is then given by $\mathbf{A}\boldsymbol{\varphi}^n = \mathbf{b}(\boldsymbol{\varphi}^{n-1})$.

While this approach of explicitly treating the convection terms preserves the smaller matrix bandwidth achieved by the FFT decomposition, it is very costly in terms of runtime. This happens because it requires the solution of several linear systems, which are inherently dependent on the system from a previous iteration. To make matters worse, the convergence rate of outer iterations – measured in terms of the iteration residual – becomes considerably slower with larger grid sizes. Therefore, for the considered cases, it is actually more efficient to forego the reduction of matrix bandwidth and implicitly solve for the convection terms.

[1] Denoted these as "outer iterations" since assuming that an iterative solver will be used for the linear system.

C.2 FFT Acceleration for FLS Schemes

With the procedure described in the previous section, the Author attempted to implement FFT acceleration for FLS schemes, in strictly regular Cartesian grids. First, consider the pure-diffusion version (i.e. with $\vec{v} = 0$) of the equation defining a FLS method:

$$\sum_{f\in\mathcal{F}(I)} \sum_{g\in\mathcal{G}(f)} w_{G_g} \Gamma \left(\vec{S}_f \cdot \nabla\right) \mathbf{d}_g \mathbf{P}_f \boldsymbol{\varphi}_{s(f)} = V \sum_{g\in\mathcal{G}(I)} w_{G_g} Q(\vec{x}_g) \tag{C.11}$$

Inner Face Stencils

Consider the contribution for a single inner face f_{in}, as given in (C.12). The summation in g is the same for all faces, and the pseudo-inverse matrix $\mathbf{P}_{s(f_{\text{in}})}$ of any inner face is equal, since the stencils of all inner faces are equivalent. Thus, these terms may be grouped into a single vector $\boldsymbol{\alpha}_{s(f_{\text{in}})}$, which depends only on the distances from the stencil cells to the target face f_{in}. As such, the values $\alpha = \alpha(\vec{x})$ are odd-symmetric about face f_{in} – i.e. $\alpha(\vec{x}\text{-}\vec{x}_{f_{\text{in}}}) = \text{-}\alpha(\text{-}\vec{x}\text{-}\vec{x}_{f_{\text{in}}})$.

$$\text{for face } f_{\text{in}}: \quad \sum_{g\in\mathcal{G}(f)} w_{G_g} \Gamma \left(\vec{S}_f \cdot \nabla\right) \mathbf{d}_g \mathbf{P}_{s(f_{\text{in}})} \boldsymbol{\varphi}_{s(f_{\text{in}})} \quad \Rightarrow \quad \boldsymbol{\alpha}_{s(f_{\text{in}})} \boldsymbol{\varphi}_{s(f_{\text{in}})} \tag{C.12}$$

The FLS2 method requires, for an inner face, a stencil of size $n_{s(f_{\text{in}})} = 2\times3\times3$. In terms of symmetry around target face f_{in}, there are three types of cells corresponding to the only unique values $\alpha_1, \alpha_2, \alpha_3$ of vector $\boldsymbol{\alpha}_{s(f_{\text{in}})}$. Therefore, expanding (C.12) for the case of FLS2 yields the expression shown in (C.13). In this expression, η, τ_a, τ_b are grid coordinates centered on the target face f_{in}, with η representing the normal direction and τ_a, τ_b the tangential directions.

$$\begin{aligned}
\boldsymbol{\alpha}_{s(f_{\text{in}})} \boldsymbol{\varphi}_{s(f_{\text{in}})} = \sum_{\delta=-\frac{1}{2}}^{\frac{1}{2}} (-1)^{\delta-\frac{1}{2}} \Bigg[& \alpha_1 \Big(\varphi_{\eta+\delta,\tau_a,\tau_b}\Big) \\
& + \alpha_2 \Big(\varphi_{\eta+\delta,\tau_a+1,\tau_b} + \varphi_{\eta+\delta,\tau_a,\tau_b+1} + \varphi_{\eta+\delta,\tau_a-1,\tau_b} + \varphi_{\eta+\delta,\tau_a,\tau_b-1}\Big) \\
& + \alpha_3 \Big(\varphi_{\eta+\delta,\tau_a+1,\tau_b+1} + \varphi_{\eta+\delta,\tau_a+1,\tau_b-1} + \varphi_{\eta+\delta,\tau_a-1,\tau_b+1} + \varphi_{\eta+\delta,\tau_a-1,\tau_b-1}\Big) \Bigg]
\end{aligned} \tag{C.13}$$

To avoid the η, τ_a, τ_b grid system, it is useful to consider a pair of opposing inner faces $f_{\text{in}}^+, f_{\text{in}}^-$ of a cell I at i, j, k. Because of the odd-symmetry, the vectors from both faces are related by $\boldsymbol{\alpha}_{s(f_{\text{in}}^+)} = \text{-}\boldsymbol{\alpha}_{s(f_{\text{in}}^-)}$. Without loss of generality, let the face pair be along the x direction – i.e. $f_{\text{in}}^+, f_{\text{in}}^-$ are east and west faces, respectively. As such, east face f_{in}^+ is at $i{+}\frac{1}{2}, j, k$, and west face f_{in}^- is at $i{-}\frac{1}{2}, j, k$. Therefore, the contribution from this pair of faces may be written as

$$\begin{aligned}
\boldsymbol{\alpha}_{s(f_{\text{in}}^+)} \boldsymbol{\varphi}_{s(f_{\text{in}}^+)} + \boldsymbol{\alpha}_{s(f_{\text{in}}^-)} \boldsymbol{\varphi}_{s(f_{\text{in}}^-)} = & \alpha_1 \Big(\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j,k}\Big) \\
& + \alpha_2 \sum_{m,n=-1}^{1} \big(1 - \delta_{|m||n|}\big) \Big(\varphi_{i-1,j+m,k+n} - 2\varphi_{i,j+m,k+n} + \varphi_{i+1,j+m,k+n}\Big) \\
& + \alpha_3 \sum_{m,n=-1}^{1} |mn| \Big(\varphi_{i-1,j+m,k+n} - 2\varphi_{i,j+m,k+n} + \varphi_{i+1,j+m,k+n}\Big)
\end{aligned} \tag{C.14}$$

where δ_{ab} is the Kronecker delta in terms of a, b. For a shorter notation, let $\boldsymbol{\alpha}_{s(f_{\text{in}}^x)}\boldsymbol{\varphi}_{s(f_{\text{in}}^x)} \equiv \boldsymbol{\alpha}_{s(f_{\text{in}}^+)}\boldsymbol{\varphi}_{s(f_{\text{in}}^+)} + \boldsymbol{\alpha}_{s(f_{\text{in}}^-)}\boldsymbol{\varphi}_{s(f_{\text{in}}^-)}$. Spatially, the total stencil $s(f_{\text{in}}^x) \equiv s(f_{\text{in}}^+) \cup s(f_{\text{in}}^-)$ represents a $3\times3\times3$ cube centered on cell I. Additionally, let the set $\{\varphi_{i-1,j,k}; \varphi_{i,j,k}; \varphi_{i+1,j,k}\}$ be called a "pencil" for a given j, k.

The previous equation highlights how the form of the stencil contributions is similar to the CD2 scheme. As such, it should also be possible to apply the same kind of Fourier synthesis as done in Section C.1. Let x be the decoupling direction and $\mathcal{F}_x$ the corresponding Fourier-based discrete expansion operator. Then, applying the operator to a single pencil for a given j, k yields

$$\mathcal{F}\Big[\alpha_{jk}\big(\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j,k}\big)\Big] = \alpha_{jk}\Lambda_i\widehat{\varphi}_{i,j,k} \tag{C.15}$$

and applying the operator to the entire contribution of $f_{\text{in}}^+, f_{\text{in}}^-$ results in

$$\begin{aligned}\mathcal{F}\left(\boldsymbol{\alpha}_{s(f_{\text{in}}^x)}\boldsymbol{\varphi}_{s(f_{\text{in}}^x)}\right) = \alpha_1\Lambda_i\widehat{\varphi}_{i,j,k} \\ + \alpha_2\Lambda_i\big(\widehat{\varphi}_{i,j+1,k} + \widehat{\varphi}_{i,j-1,k} + \widehat{\varphi}_{i,j,k+1} + \widehat{\varphi}_{i,j,k-1}\big) \\ + \alpha_3\Lambda_i\big(\widehat{\varphi}_{i,j+1,k+1} + \widehat{\varphi}_{i,j+1,k-1} + \widehat{\varphi}_{i,j-1,k+1} + \widehat{\varphi}_{i,j-1,k-1}\big)\end{aligned} \tag{C.16}$$

Similar expressions as (C.14) may be derived for the face pairs along y (north-south) and along z (top-bottom), as given in (C.17) and (C.18). These expressions are identical to (C.14) except for a permutation of the index variations l, m, n.

$$\begin{aligned}\boldsymbol{\alpha}_{s(f_{\text{in}}^y)}\boldsymbol{\varphi}_{s(f_{\text{in}}^y)} = \alpha_1\Big(\varphi_{i,j-1,k} - 2\varphi_{i,j,k} + \varphi_{i,j+1,k}\Big) \\ + \alpha_2\sum_{l,n=-1}^{1}\big(1-\delta_{|l||n|}\big)\Big(\varphi_{i+l,j-1,k+n} - 2\varphi_{i+l,j,k+n} + \varphi_{i+l,j+1,k+n}\Big) \\ + \alpha_3\sum_{l,n=-1}^{1}\big|ln\big|\Big(\varphi_{i+l,j-1,k+n} - 2\varphi_{i+l,j,k+n} + \varphi_{i+l,j+1,k+n}\Big)\end{aligned} \tag{C.17}$$

$$\begin{aligned}\boldsymbol{\alpha}_{s(f_{\text{in}}^z)}\boldsymbol{\varphi}_{s(f_{\text{in}}^z)} = \alpha_1\Big(\varphi_{i,j,k-1} - 2\varphi_{i,j,k} + \varphi_{i,j,k+1}\Big) \\ + \alpha_2\sum_{l,m=-1}^{1}\big(1-\delta_{|l||m|}\big)\Big(\varphi_{i+l,j+m,k-1} - 2\varphi_{i+l,j+m,k} + \varphi_{i+l,j+m,k+1}\Big) \\ + \alpha_3\sum_{l,m=-1}^{1}\big|lm\big|\Big(\varphi_{i+l,j+m,k-1} - 2\varphi_{i+l,j+m,k} + \varphi_{i+l,j+m,k+1}\Big)\end{aligned} \tag{C.18}$$

Since these face pairs do not line up with the decoupling direction x of $\mathcal{F}_x$, the expressions above need to be rearranged to highlight the form $\big(\varphi_{i-1,j+m,k+n} - 2\varphi_{i,j+m,k+n} + \varphi_{i+1,j+m,k+n}\big)$:

$$\begin{aligned}\boldsymbol{\alpha}_{s(f_{\text{in}}^y)}\boldsymbol{\varphi}_{s(f_{\text{in}}^y)} = -2\Big[\alpha_2\big(\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j,k}\big) + \big(\alpha_1 + 2\alpha_2\big)\varphi_{i,j,k}\Big] \\ + \sum_{m,n=-1}^{1}|mn|\Big[\alpha_3\big(\varphi_{i-1,j+m,k+n} - 2\varphi_{i,j+m,k+n} + \varphi_{i+1,j+m,k+n}\big) + \big(\alpha_2 + 2\alpha_3\big)\varphi_{i,j+m,k+n}\Big] \\ - 2\sum_{n=-1}^{1}|n|\Big[\alpha_3\big(\varphi_{i-1,j,k+n} - 2\varphi_{i,j,k+n} + \varphi_{i+1,j,k+n}\big) + \big(\alpha_2 + 2\alpha_3\big)\varphi_{i,j,k+n}\Big] \\ + \sum_{m=-1}^{1}|m|\Big[\alpha_2\big(\varphi_{i-1,j+m,k} - 2\varphi_{i,j+m,k} + \varphi_{i+1,j+m,k}\big) + \big(\alpha_1 + 2\alpha_2\big)\varphi_{i,j+m,k}\Big]\end{aligned} \tag{C.19}$$

$$\begin{aligned}\boldsymbol{\alpha}_{s(f_{\text{in}}^z)}\boldsymbol{\varphi}_{s(f_{\text{in}}^z)} = &-2\Big[\alpha_2\big(\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j,k}\big) + \big(\alpha_1 + 2\alpha_2\big)\varphi_{i,j,k}\Big] \\ &+ \sum_{m,n=-1}^{1} |mn|\Big[\alpha_3\big(\varphi_{i-1,j+m,k+n} - 2\varphi_{i,j+m,k+n} + \varphi_{i+1,j+m,k+n}\big) + \big(\alpha_2 + 2\alpha_3\big)\varphi_{i,j+m,k+n}\Big] \\ &+ \sum_{n=-1}^{1} |n|\Big[\alpha_2\big(\varphi_{i-1,j,k+n} - 2\varphi_{i,j,k+n} + \varphi_{i+1,j,k+n}\big) + \big(\alpha_1 + 2\alpha_2\big)\varphi_{i,j,k+n}\Big] \\ &- 2\sum_{m=-1}^{1} |m|\Big[\alpha_3\big(\varphi_{i-1,j+m,k} - 2\varphi_{i,j+m,k} + \varphi_{i+1,j+m,k}\big) + \big(\alpha_2 + 2\alpha_3\big)\varphi_{i,j+m,k}\Big] \quad \text{(C.20)}\end{aligned}$$

Applying operator $\mathcal{F}_x$ to the obtained expressions lead to the transformed contributions of north-south pairs (C.21) and top-bottom pairs (C.22).

$$\begin{aligned}\mathcal{F}\left(\boldsymbol{\alpha}_{s(f_{\text{in}}^y)}\boldsymbol{\varphi}_{s(f_{\text{in}}^y)}\right) = &-2\Lambda_i\big(\alpha_1 + 3\alpha_2\big)\widehat{\varphi}_{i,j,k} \\ &+ \Lambda_i\big(\alpha_1 + 3\alpha_2\big)\big(\widehat{\varphi}_{i,j+1,k} + \widehat{\varphi}_{i,j-1,k}\big) - 2\Lambda_i\big(\alpha_2 + 3\alpha_3\big)\big(\widehat{\varphi}_{i,j,k+1} + \widehat{\varphi}_{i,j,k-1}\big) \\ &+ \Lambda_i\big(\alpha_2 + 3\alpha_3\big)\big(\widehat{\varphi}_{i,j+1,k+1} + \widehat{\varphi}_{i,j+1,k-1} + \widehat{\varphi}_{i,j-1,k+1} + \widehat{\varphi}_{i,j-1,k-1}\big) \quad \text{(C.21)}\end{aligned}$$

$$\begin{aligned}\mathcal{F}\left(\boldsymbol{\alpha}_{s(f_{\text{in}}^z)}\boldsymbol{\varphi}_{s(f_{\text{in}}^z)}\right) = &-2\Lambda_i\big(\alpha_1 + 3\alpha_2\big)\widehat{\varphi}_{i,j,k} \\ &- 2\Lambda_i\big(\alpha_2 + 3\alpha_3\big)\big(\widehat{\varphi}_{i,j+1,k} + \widehat{\varphi}_{i,j-1,k}\big) + \Lambda_i\big(\alpha_1 + 3\alpha_2\big)\big(\widehat{\varphi}_{i,j,k+1} + \widehat{\varphi}_{i,j,k-1}\big) \\ &+ \Lambda_i\big(\alpha_2 + 3\alpha_3\big)\big(\widehat{\varphi}_{i,j+1,k+1} + \widehat{\varphi}_{i,j+1,k-1} + \widehat{\varphi}_{i,j-1,k+1} + \widehat{\varphi}_{i,j-1,k-1}\big) \quad \text{(C.22)}\end{aligned}$$

Adding the contributions from all three inner face pairs yields the final equation for an inner cell I, representing row I of the transformed linear system $\widehat{\mathbf{b}} = \mathcal{F}\left(\mathbf{A}\boldsymbol{\varphi}\right)$, as given by

$$\begin{aligned}\widehat{b}_I = &-\Lambda_i\big(3\alpha_1 + 6\alpha_2\big)\widehat{\varphi}_{i,j,k} \\ &+ \Lambda_i\big(\alpha_1 + 2\alpha_2 - 6\alpha_3\big)\big(\widehat{\varphi}_{i,j+1,k} + \widehat{\varphi}_{i,j-1,k} + \widehat{\varphi}_{i,j,k+1} + \widehat{\varphi}_{i,j,k-1}\big) \\ &+ \Lambda_i\big(2\alpha_2 + 7\alpha_3\big)\big(\widehat{\varphi}_{i,j+1,k+1} + \widehat{\varphi}_{i,j+1,k-1} + \widehat{\varphi}_{i,j-1,k+1} + \widehat{\varphi}_{i,j-1,k-1}\big) \quad \text{(C.23)}\end{aligned}$$

Boundary Conditions and Related Issues

Due to the initial assumptions of the previous derivation, equation (C.23) is only valid for cells whose face stencils do not contain boundary faces. As such, it can only be applied to the entire system if *all* boundary conditions – including those of dimensions that are not the decoupling direction – are periodic. As listed in Table C.1, the direct operator for a periodic boundary condition is `DFT`. This, however, presents an additional difficulty, because this operator represents a real-to-complex transform – i.e. even when given a pure-real input, the output will always be complex. Thus, the underlying solver must be able to deal with complex numbers. Given the timeframe, the Author was unable to successfully implement FFT with periodic boundary conditions. The implementation of this feature is thus left for future work.

The use of FFT in an FLS scheme with Dirichlet and/or Neumann boundary conditions is up-to-now not clear. One could attempt to decouple only the inner face stencils, and, for stencils with boundary faces, take advantage of the linearity propriety of $\mathcal{F}$ by transforming only the φ values – e.g. $\mathcal{F}\big(\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j,k}\big) = \widehat{\varphi}_{i-1,j,k} - 2\widehat{\varphi}_{i,j,k} + \widehat{\varphi}_{i+1,j,k}$ – thus not reducing the number of stencil points. While this approach – at the time of writing – has not yet worked, it appears to be a promising strategy, specially since, for large grid sizes. The number of total faces in a grid is on the same order of magnitude as

the number of cells, $N_f \sim N_C \equiv N_x N_y N_z$ where the N_x, N_y, N_z are the number of cells along each dimension. On the other hand, the number of stencils that contain a boundary face is on the order of the outer grid area, $N_{s(f)} - N_{s(f_{\text{in}})} \sim 2(N_x N_y + N_y N_z + N_z N_x)$. Therefore, as the grid size increases, the number of boundary-face-containing stencils becomes negligible in relation to inner-face stencils, and the overall benefit given by the reduction of the matrix bandwidth is then recouped.

C.3 Deferred-Correction Approach

Some high-order methods may sometimes be unstable, unlike their lower-order counterparts. Therefore, it might be beneficial to combine the two to take advantage of the stability of low-order methods and the accuracy of high-order methods. One such approach is the so-called "deferred correction", which has been traditionally used in the context of flux limiters [8].

Consider the linear systems created from two numerical methods, one corresponding to a low-order method (C.24) and another to a high-order method (C.25). Now, assuming that the system will be solved using an iterative method, one may define $\boldsymbol{\varphi}^n$ as the solution vector at iteration n. Combining a previous-iteration version of (C.24) and (C.25) into the current-iteration version of both (C.24) yields the deferred-correction linear system (C.26). Assuming that the system converges, then, as the solver iterates, $\boldsymbol{\varphi}^{n-1}$ approaches $\boldsymbol{\varphi}^n$ and, in the limit of convergence, the deferred-correction system becomes equivalent to the high-order system.

$$\mathbf{A}_L \boldsymbol{\varphi} = \mathbf{b}_L \tag{C.24}$$

$$\mathbf{A}_H \boldsymbol{\varphi} = \mathbf{b}_H \tag{C.25}$$

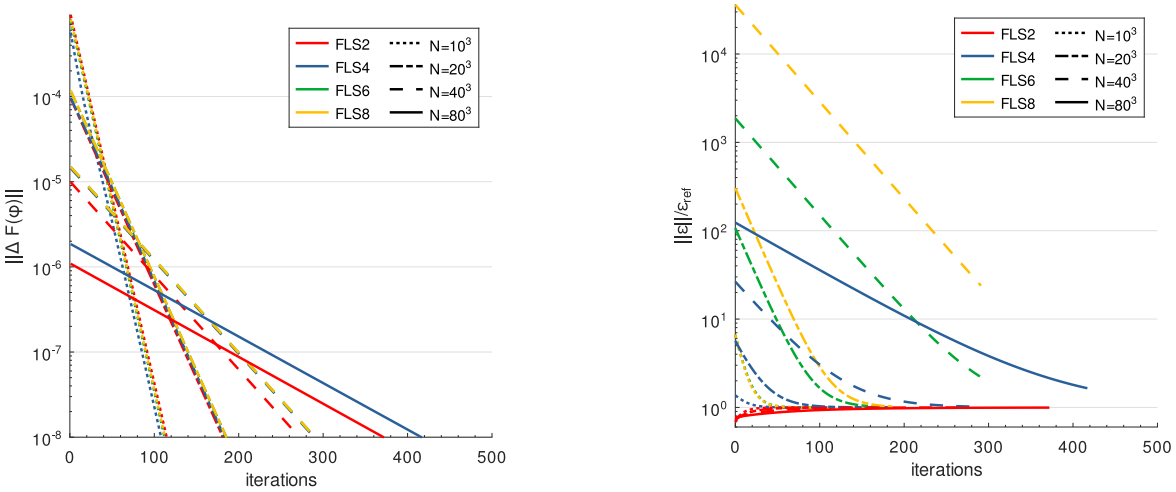
$$\mathbf{A}_L \boldsymbol{\varphi}^n = \mathbf{b}_H + (\mathbf{A}_L - \mathbf{A}_H)\, \boldsymbol{\varphi}^{n-1} \tag{C.26}$$

It is important to note that the deferred-correction approach as described in (C.26) is significantly less computationally efficient than using a single method, since it requires the computation and storage of two $N \times N$ – albeit sparse – matrices.

C.3.1 Without FFT decomposition

To test the deferred-correction approach described above, the same was applied to combinations of CD2 and FLS as, respectively, the low- and high-order methods. The results are summarized in Figure C.2, which plots, for different grid sizes, the iteration residual norm $\|\Delta\widehat{\varphi}^n\|$ and the numerical error norm $\|\varepsilon\|$ in terms of outer iterations.

The norm type considered here is the weighted l_1-norm, defined according to (C.27) for an arbitrary vector υ of length N. Equations (C.28) and (C.29) define the iteration residual norm and the numerical error norm, where $\boldsymbol{\varphi}, \boldsymbol{\varphi}'$ are the analytical and numerical solutions, respectively. The reference error $\|\varepsilon_{\text{ref}}\|$ – used to normalize the error norm in Figure C.2b – is the actual converged error value, obtained



(a) Iteration error, $\|\Delta\boldsymbol{\varphi}^n\|$

(b) Normalized error, $\frac{\|\boldsymbol{\varepsilon}\|}{\varepsilon_{\text{ref}}}$

Figure C.2 Error measurements for deferred-correction approach with CD-2 and FLS-n as the low- and high-order methods, for different grid sizes and convergence criteria of $\|\Delta\boldsymbol{\varphi}^n\| > 10^8$.

by solving the FLS scheme implicitly without deferred correction.

$$\|\boldsymbol{v}\| \equiv \frac{1}{N}\sum_{i=1}^{N}|v_i| \tag{C.27}$$

$$\|\Delta\widehat{\varphi}^n\| \equiv \left\|\widehat{\boldsymbol{\varphi}}^n - \widehat{\boldsymbol{\varphi}}^{n-1}\right\| \tag{C.28}$$

$$\|\boldsymbol{\varepsilon}\| \equiv \|\boldsymbol{\varphi} - \boldsymbol{\varphi}'\| \tag{C.29}$$

It is clear from Figure C.2a (and also, to a lesser extent, from Figure C.2b) that the convergence rate of the deferred correction is not dependent on the order of the chosen methods, but instead on the grid size. Figure C.2b shows the clear trend of "diminishing returns" as the results converge. This illustrates something common to all numerical methods, which is the difficulty and importance of choosing an adequate convergence criteria, considering that $\|\varepsilon^n\|$ is unknown *a priori*, in general.

Note that, in Figure C.2b, the results from FLS2 are the only that show a normalized error of less than one, i.e. $\|\boldsymbol{\varepsilon}\| < \|\boldsymbol{\varepsilon}_{\text{ref}}\|$; this happens due to the CD2 method having a lower numerical error than FLS2, meaning that the deferred-correction approach has to converge to a larger error than its initial value.

An unexpected result from these tests was that, using CD2 as the implicit lower-order method, converged solutions were obtained for all FLS schemes, including the sixth- and eight-order variants. This goes against what was reported by Vasconcelos [9]. However, his results were done with a purely one-dimensional implementation, while the current tests were obtained with a fully three-dimensional implementation.

C.3.2 With FFT decomposition

The same deferred-correction approach could be applied – in theory, at least – for methods with FFT decomposition. The implicit method could be chosen to be the FFT-accelerated CD2, while the explicit high-order method could be a FLS scheme. With this, one manages to avoid developing an FFT-based

FLS scheme. However, the Author was unable to successfully implement this approach. This is specially unnerving because, since both the deferred-correction approach (without FFT) and the FFT-based CD2 have been correctly implemented, the combination of these two should have been fairly straightforward.

C.4 Final Remarks

Regarding computational cost, the Author believes that – in the considered context of FFT acceleration – the explicit treatment of convection terms and the deferred-correction approach are extremely lacking. For the former, there is no gain in runtime when compared to the implicitly solving for the convection terms. However, these two strategies remain relevant for their original purposes, which is to provide additional stability to the numerical methods.

Regarding implementation, most of the issues encountered were not regarding the formal derivation of the FFT-based methods. Instead, the most troublesome issue was in programming with and conciliating the two distinct numerical library used: FFTW [4] and PETSc [10]. In specific, these two take different approaches to – among others – the definition of a complex variable type and the decomposition of parallel (distributed) arrays. The former was the main obstacle in implementing a FFT-based method with periodic boundary conditions, while the latter was the main reason why the current work only considers serial and not parallel computation.

Overall, although some proposed approaches were unsuccessful, the underlying strategy has been shown to be promising, for the considered case of regular Cartesian grid. It is doubtful, however, that such can be implemented for unstructured grids – for which the FLS schemes were originally developed – since the Fourier synthesis requires that the grid spacing along the decoupling dimension be uniform.

Regarding future work on this topic, it would first be relevant to solve the implementation issues for an FFT-accelerated FLS2 (section C.2). This enables a more solid comparison between the two versions of FLS2, specially in regards to parallel performance. Second, for the case of CD2, it would be interesting to compare how much of the FFT-acceleration benefit is lost when implicitly solving for convection terms.

References

[1] Costa P. “A FFT-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows”. In: *Computers & Mathematics with Applications* 76.8 (2018), pp. 1853–1862. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2018.07.034.

[2] Costa P. “A FFT-accelerated multi-block finite-difference solver for massively parallel simulations of incompressible flows”. In: *Computer Physics Communications* 271 (2022), p. 108194. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2021.108194.

[3] Schumann U, Sweet RA. “Fast Fourier transforms for direct solution of Poisson’s equation with staggered boundary conditions”. In: *Journal of Computational Physics* 75.1 (1988), pp. 123–137. ISSN: 0021-9991. DOI: 10.1016/0021-9991(88)90102-7.

[4] Frigo M, Johnson S. “The Design and Implementation of FFTW3”. In: *Proceedings of the IEEE* 93.2 (2005), pp. 216–231. DOI: 10.1109/JPROC.2004.840301.

[5] Canuto C, Hussaini MY, Quarteroni A, Zang Jr TA. *Spectral Methods*. Vol. 2: *Evolution to Complex Geometries and Applications to Fluid Dynamics*. Springer, 2007. ISBN: 978-3-540-30727-3.

[6] Berselli LC, Iliescu T, Layton WJ. *Mathematics of Large Eddy Simulation of Turbulent Flows*. Springer, 2006. ISBN: 978-3-540-26316-6.

[7] *ORNL's Frontier First to Break the Exaflop Ceiling*. TOP500. 2022. URL: https://www.top500.org/news/ornls-frontier-first-to-break-the-exaflop-ceiling (visited on 06/2022).

[8] Ferziger JH, Perić M, Street RL. *Computational Methods for Fluid Dynamics*. 4th ed. Springer, 2020. ISBN: 978-3-319-99693-6. DOI: 10.1007/978-3-319-99693-6.

[9] Vasconcelos AGR. "A Very High-Order Finite Volume Method Based on Weighted Least Squares for the Solution of Poisson Equation on Unstructured Grids". MSc thesis. Instituto Superior Técnico, Universidade de Lisboa, 2017. FENIX: 1972678479053984 [MEAER].

[10] Balay S, Abhyankar S, Adams MF, Benson S, Brown J, Brune P, Buschelman K, Constantinescu E, Dalcin L, Dener A, Eijkhout V, Gropp WD, Hapla V, Isaac T, Jolivet P, Karpeev D, Kaushik D, Knepley MG, Kong F, Kruger S, May DA, McInnes LC, Mills RT, Mitchell L, Munson T, Roman JE, Rupp K, Sanan P, Sarich J, Smith BF, Zampini S, Zhang H, Zhang H, Zhang J. *PETSc/TAO Users Manual*. Tech. rep. ANL-21/39 - Revision 3.17. Argonne National Laboratory, 2022.