UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS AERONÁUTICOS

Acceleration of an Unstructured
Compressible Flow Solver using Implicit
Methods and Graphics Cards

TESIS DOCTORAL

Jesús Pueblas Sánchez-Guerra

Ingeniero Aeronáutico

Madrid, 2017

Departamento de Motopropulsión y
Termofluidodinámica

Escuela Técnica Superior de Ingenieros Aeronáuticos

Acceleration of an Unstructured
Compressible Flow Solver using Implicit
Methods and Graphics Cards

Autor

Jesús Pueblas Sánchez-Guerra

Ingeniero Aeronáutico

Director

Roque Corral García

Doctor Ingeniero Aeronáutico

Madrid, Junio 2017

Escuela Técnica Superior de Ingenieros Aeronáuticos

UNIVERSIDAD POLITÉCNICA DE MADRID

Tribunal nombrado por el Magnífico y Excelentísimo Señor Rector de la Universidad Politécnica de Madrid. El día de de 2017.

Presidente: D. Javier Jiménez Sendin

Secretario: D. Oscar Flores Arias

Vocal: D. Alfredo Pinelli

Vocal: D. Luca di Mare

Vocal: D. Fernando Gisbert Cervera

Suplente: D. Manuel Antonio Burgos Olmos

Suplente: D. Raquel Gómez Miguel

Realizado el acto de defensa y lectura de la tesis el día 8 de Junio de 2017 en la E.T.S.I. Aeronáuticos.

El Presidente

Los Vocales

El Secretario

Acknowledgments

Esta tesis ha sido posible gracias al apoyo de mucha gente. Sin su ayuda constante este trabajo no hubiera alcanzado la madurez necesaria para cumplir los objetivos propuestos.

En primer lugar, quisiera agradecer a Roque Corral por ofrecerme la posibilidad de realizar este trabajo, mostrando plena confianza en mí. Gracias a él, he podido adquirir una serie de conocimientos en el área de la Mecánica de Fluidos Computacional, los cuales son de gran valor para el desarrollo de mi carrera profesional.

Sin duda, mi mayor gratitud es para los desarrolladores del Mu^2s^2T , el código CFD interno de ITP, en especial a Fernando, que me enseñó con gran paciencia la estructura del programa durante mis comienzos, por inculcarme su capacidad de trabajo y ser un indudable ejemplo a seguir. Las conversaciones con Jesús Contreras han sido vitales para resolver una gran serie de problemas numéricos, en especial cómo abordar las condiciones de contorno. Finalmente, y no menos importante, Javier Crespo me ha enseñado las técnicas de paralelización del código, las cuales son necesarias para poder realizar simulaciones en múltiples procesadores.

Además, agradezco también al resto de compañeros del departamento de Tecnología de Simulación de ITP. El desarrollo del resto de herramientas tanto de pre-proceso como de post-proceso han hecho posible que mi trabajo tenga sentido.

Los compañeros del proyecto europeo MAGPI también han contribuido en este trabajo. De ellos he aprendido a trabajar en el proceso de diseño y ensayos de una turbina de baja velocidad.

Por último, gracias a los miembros del tribunal por aceptar a valorar este trabajo.

Abstract

An implicit unstructured edge-based three-dimensional RANS equations solver which executes both in Central Processing Units (CPUs) and Graphics Processing Units (GPUs) has been implemented.

The goal of this work is to accelerate the convergence to the steady state as well as increase the robustness of an existing multigrid methods which uses a five-stage Runge-Kutta method as smoother. This type of methods tends to oscillate in turbomachinery including recirculation bubbles or shear flows, which are common in tip gaps, off-design conditions and cavity flows.

A two-dimensional Fourier analysis of different implicit techniques has been conducted in order to understand the behaviour of the method, and bench-mark it against an explicit Runge-Kutta scheme.

Three different methods have been implemented to solve the sparse system of linear equations resulting from the implicit method, namely: Jacobi, LU-SGS and GMRES. The latter is preconditioned either by the Jacobi or the LU-SGS methods to accelerate its convergence. An efficient implementation on GPUs devices has been achieved, obtaining a speed-up between an NVIDIA GeForce GTX780Ti GPU and an Intel Xeon Processor E5-1620 of about 41 and 36 for the Jacobi and the GMRES+LU-SGS solvers respectively, whereas for the existing Runge-Kutta solver the speed-up is about 26 for the same hardware..

The minimum convergence time for the Jacobi implicit solver is obtained for CFL numbers of about 20 and between 5 to 10 inner iterations. These figures have been corroborated empirically by means of three-dimensional simulations and theoretical studies. The parameters for the GMRES+LU-SGS solver are similar.

Several tests cases have been simulated to validate the code, paying attention to cavity flows. Finally, the new solver has been used to assess the influence of the sealing flows on the secondary flows of a low-pressure turbine. The experimental data were obtained at the Large Scale Turbine Rig at Technische Universität Darmstadt within the context

of the European project MAGPI. The effect of the sealing air is analysed, varying the cooling mass flow for two operating conditions. The penetration of the sealing flow in the main stream increases with the cooling flow, displacing the horseshoe and passage vortices towards the mid-span, and hence reducing the turbine efficiency.

This piece of work demonstrates for the first time the benefit of using together unstructured grids, implicit methods and GPUs in an industrial representative environment.

Resumen

Un método implícito basado en aristas para resolver numéricamente en mallas no estructuradas las ecuaciones tridimensionales de Navier-Stokes promediadas de Reynolds ha sido implementado. Dicho método está basado en la matriz Jacobiana aproximada de primer orden y es capaz de correr tanto en Unidades Centrales de Procesamiento (CPUs) como en Unidades Gráficas de Procesamiento (GPUs).

El objetivo de este trabajo es acelerar los tiempos de convergencia e incrementar la robustez de un código explícito existente de cinco pasos de Runge-Kutta preconditionado con el método de multi-malla, el cual tiende a oscilar para ciertas configuraciones de turbomaquinaria, tales como burbujas de recirculación o flujos de cortadura, los cuales son comunes en sellos, condiciones fuera de diseño y flujo en cavidades.

Un análisis bidimensional de Fourier de las diferentes técnicas implícitas se ha estudiado con el fin de entender el comportamiento del método en comparación con el esquema explícito Runge-Kutta.

Tres resolvedores de sistemas lineales han sido implementados: Jacobi, LU-SGS y GMRES. El último es preconditionado por los métodos de Jacobi o LU-SGS para acelerar su convergencia. La implementación satisfactoria para correr en GPUs se ha alcanzado, obteniendo unos factores de aceleración entre una GPU NVIDIA GeForce GTX780Ti y una CPU Intel Xeon Processor E5-1620 del orden de 41 y 36 para los métodos de Jacobi y GMRES+LU-SGS respectivamente, mientras que para el código existente de Runge-Kutta la aceleración es aproximadamente de 26.

El óptimo del tiempo de convergencia para el algoritmo implícito se observa entre un cierto rango del paso de tiempo y de iteraciones para resolver el sistema lineal. Cuando el sistema lineal se resuelve con el método de Jacobi, el CFL óptimo se encuentra alrededor de 20, necesitando entre 5 y 10 iteraciones lineales. Estos números han sido corroborados empíricamente con simulaciones tridimensionales y estudios teóricos. Los parámetros para el método GMRES+LU-SGS son muy similares.

Varios casos de prueba se han simulado para validar el código, prestando atención en flujo de cavidades. Finalmente, el nuevo resolvedor se ha usado para evaluar la influencia

de los flujos de sellado sobre los flujos secundarios de las turbinas de baja presión. Los datos experimentales obtenidos en una instalación de ensayos de una turbina a gran escala situada en la Universidad de Darmstadt se han usado para validar el método numérico. El efecto del aire de sellado es analizado, variando el flujo másico de refrigeración para dos condiciones de operación. La penetración del flujo de sellado en la corriente principal se incrementa con el flujo de refrigeración, desplazando los vórtices de herradura y de pasaje hacia la mitad del canal, y por consiguiente, reduciendo la eficiencia de la turbina.

Este trabajo demuestra por primera vez el beneficio de usar de forma conjunta mallas no estructuradas, métodos implícitos y GPUs en un entorno típico industrial.

Contents

List of Figures	xvi
List of Tables	xvii
Nomenclature	xix
1 Introduction	3
1.1 Goal and Thesis Contributions	6
1.2 Literature Review in Implicit Methods	8
1.3 Thesis Outline	10
2 3D Unstructured Base Solver	13
2.1 Introduction	13
2.2 Reynolds-Averaged Navier-Stokes equations	14
2.3 Turbulence Model	16
2.3.1 Algebraic Model	16
2.3.2 Wicox 2006 $k - \omega$ Turbulence Model	17
2.4 Dimensionless Reynolds-Averaged Navier-Stokes equations	18
2.5 Spatial Discretisation	19
2.5.1 Evaluation of the convective terms	20
2.5.2 Evaluation of the viscous terms	21
2.5.3 Artificial viscosity construction based on the pseudo-Laplacian (JST)	22

2.5.4	MUSCL approach	24
2.6	Boundary Conditions	25
2.6.1	Periodic Boundary Condition	26
2.6.2	1D steady non-reflecting inlet boundary condition	26
2.6.3	1D Steady Non-Reflecting Outlet Boundary Condition	29
2.6.4	Solid wall Boundary Conditions	30
2.7	Explicit Time Integration	31
2.7.1	Preconditioning	32
2.7.1.1	Local Time Step	32
2.7.1.2	Block-Jacobi Preconditioning	33
2.7.1.3	Multigrid Preconditioning	34
2.7.1.4	Low Mach number preconditioning	35
3	Implicit Algorithms	39
3.1	Introduction	39
3.2	Approximations of the Jacobian matrix	42
4	Linear Solvers	45
4.1	Introduction	45
4.2	Iterative Solvers	46
4.2.1	Splitting Methods	47
4.2.2	Krylov Methods	51
4.2.2.1	GMRES	51
4.2.2.2	Preconditioned GMRES	52
4.3	Block Linear Solvers	52
4.4	Implementation of the Linear Solvers	53

4.4.1	Introduction	53
4.4.2	Jacobi solver	54
4.4.3	LU-SGS solver	54
4.4.4	GMRES solver	56
4.4.5	Preconditioned GMRES solver	56
5	Linear stability and convergence analysis	59
5.1	Introduction	59
5.2	Temporal Discretisation	62
5.2.1	Explicit Integration	63
5.2.2	Implicit Integration based on the full Jacobian	63
5.2.3	Implicit Integration based on the first-order Jacobian	64
5.3	Stability of the Linear Advection Equation	65
5.3.1	Effect of the spatial discretisation	66
5.3.2	Effect of the flow angle	67
5.3.3	Effect of the inner iterations of the implicit method	68
5.3.4	Computational cost	69
5.4	Stability of the Euler equations	77
6	Multi-Core and Many-Core Implementation	83
6.1	Introduction	83
6.2	Serial edge loop	87
6.3	Distributed memory parallelisation	89
6.4	Shared memory parallelism	92
6.4.1	Gradient evaluation	96
6.4.2	Convective and viscous fluxes evaluation	98

6.4.3	Hybrid distributed-shared memory parallelism	98
6.5	Implicit Solver Parallelisation	100
6.5.1	Jacobi method parallelisation	101
6.5.2	LU-SGS method parallelisation	102
6.5.2.1	Shared memory LU-SGS parallelisation	102
6.5.2.2	LU-SGS distributed memory parallelisation	106
6.5.3	GMRES method parallelisation	106
7	Numerical Results	109
7.1	Introduction	109
7.2	Linear solvers results	110
7.2.1	Effect of the Linear Solver Parameters	112
7.3	Multi-core and many-core performances of the baseline solver	117
7.3.1	Single GPU benchmark	118
7.3.2	Network characterization	120
7.3.3	Multiple GPU Benchmark	121
7.3.4	Comparison against a CPU cluster	123
7.4	Multi-core and many-core performances of the implicit solver	125
7.4.1	Single GPU benchmark	125
7.4.2	Multiple GPU Benchmark	127
7.4.3	Comparison against a CPU cluster	128
7.5	Test Cavity Flows	129
7.5.1	Lid-Driven Cavity	129
7.5.2	Two coaxial rotating disks	132
7.5.3	Sealed Rotor-Stator Cavity	133
7.6	ADTURB Vane	134

8	CFD Analysis in a Low-Speed Turbine Rig	137
8.1	Introduction	138
8.2	Model Description	139
8.2.1	Computational Domain and Mesh	139
8.2.2	Simulation Matrix	143
8.2.3	Boundary Conditions	143
8.3	Multistage results	144
8.3.1	Seal Air Flows	144
8.3.2	Cavity Static Pressure	146
8.3.3	Stator 2 hub end-wall pressure distribution	147
8.3.4	Impact of the egress flow on the secondary structures	148
8.3.5	Stator 2 pressure distribution profiles	150
8.3.6	Overall Main Annulus Effects	151
8.4	Unsteady Flow Solution	153
8.5	Enhanced model to postdict experimental data	154
8.6	Conclusions	157
9	Conclusions and Further Work	159
	Bibliography	163
A	Boundary Conditions Derivatives	173
B	GMRES linear solver	177
C	Asymptotic analysis of the 2D Linear Advection amplification matrix	183
D	The non-dimensional Jacobian matrices for the Euler equations	189

List of Figures

1.1	Growth of supercomputers performance, based on data from top500.org site.	4
1.2	Evolution of the convergence time of Mu^2s^2T solver during the last 17 years for a 191000 point mesh.	6
2.1	Sketch of the simplified mixing length function.	17
2.2	Hybrid cell grid and associated dual mesh.	20
2.3	Hybrid boundary cell grid and associated dual mesh.	21
2.4	Piecewise parabolic reconstruction.	24
2.5	Periodic nodes and the dual cells. Cartesian (a) and asi-symmetric (b) periodicity.	27
2.6	Low Mach parameter ϵ variation with the Mach number.	37
4.1	Illustration of an edge contribution to the Jacobian matrix.	53
5.1	Eigenvalue footprints of the amplification matrix for the linear advection equation. $CFL_{explicit} = 3.5$. $CFL_{implicit} = 10^4$. $\Lambda = 1$. $\alpha = 1^\circ$. Exact implicit matrix inversion.	67
5.2	Eigenvalue footprints of the amplification matrix for the linear advection equation. $CFL_{explicit} = 3.5$. $CFL_{implicit} = 10^4$. $\Lambda = 1$. $\alpha = 45^\circ$. MUSCL scheme. Exact matrix inversion.	68
5.3	Eigenvalue footprints of the amplification matrix for the linear advection equation. Effect of the linear solver iterations. $CFL_{implicit} = 10^4$. $\Lambda = 1$. $\alpha = 45^\circ$. MUSCL scheme.	69

5.4	Effect of the Number of grid Points ($N_p = N_x = N_y$) on the spectral radius for $\alpha = 0$, (left) and $\alpha = 45^\circ$ (right).	70
5.5	Effect of the CFL and the number of Jacobi iterations on the spectral radius. Left: $\alpha = 0^\circ$. Right: $\alpha = 45^\circ$ ($N_p = 60$).	73
5.6	Effect of the CFL and the number of Jacobi iterations on the computational cost. Left: $\alpha = 0^\circ$. Right: $\alpha = 45^\circ$ ($N_{points} = 60 \times 60$).	74
5.7	Computational cost of a solver that could converge the linear system in 10 iterations to the same level than the Jacobi method with 100 iterations.	75
5.8	Influence of the time integration procedure on the eigenvalue footprint. $\Lambda = 1$, $\alpha = 45^\circ$, MUSCL scheme. (a) Explicit and $CFL = 3.5$. (b) Implicit with exact integration and $CFL = 10^4$. (c) Implicit with exact integration and $CFL = 20$. (d) Implicit with 10 iterations of Jacobi linear solver and $CFL = 20$	75
5.9	Phase velocity (top) and modulus (bottom) of the amplification matrix for 1D linear advection equation and different schemes.	76
5.10	Non-dimensional convection distance (top) and modulus (bottom) of the amplification matrix as a function of the CFL number for different number of the inner Jacobi iterations. for the 1D linear advection equation.	77
5.11	Influence of the spatial discretisation and time integration on the eigenvalue footprint. Top: Explicit ($CFL=3$). Bottom: Implicit ($CFL=20$) $M = 0.5$, $\Lambda = 1$, $\alpha = 0$	78
5.12	Eigenvalue footprint when the fluid is not aligned with the cell. MUSCL-Roe + LTS. $M = 0.5$, $\Lambda = 1$, $\alpha = 40$	80
5.13	Influence of the spatial discretisation and time integration on the eigenvalue footprint. $M = 0.05$, $\Lambda = 1$, $\alpha = 0$	81
5.14	Influence of the spatial discretisation and time integration on the eigenvalue footprint. $M = 0.5$, $\Lambda = 10^{-4}$, $\alpha = 0$	82
6.1	Reverse Cuthill-McKee ordering to minimize cache-misses.	88

6.2	Mesh split in 16 sub-domains using the ParMETIS library routines ([53]). .	92
6.3	Reverse Cuthill-McKee followed by an ordering by groups to avoid simultaneous memory access within a group.	94
6.4	Sequence of executions and data communications to compute an edge loop.	99
6.5	(a) Two-dimensional mesh with periodic upper and lower boundaries, with the original node numbering and the colour code after applying the multi-colouring algorithm. Non-zero pattern of the original Jacobian matrix (b) and after applying the multi-coloring algorithm (c).	103
6.6	Block-decomposed matrix for LU-SGS solver.	106
7.1	Surface mesh and dimensions of the circular bump channel test case (a) and Mach number colour contours (b) (13630 nodes and 68132 cells). . .	111
7.2	Convergence history for inviscid subsonic channel flow. Comparison between Mu^2s^2T code and Löhner et al. [60].	111
7.3	Convergence history of the linear system for two CFL numbers and different linear solvers. (Inviscid circular bump channel).	113
7.4	Convergence history of the linear system in terms of iterations (a) and CPU time (b) for different linear solvers, preconditioners and number of iterations of the GMRES+Jacobi solver for the inviscid circular bump channel test case.	114
7.5	Original matrix pattern of the linear system with the Cuthill-McKee ordering (a) and after applying multi-colouring reordering (b).	115
7.6	Effect of the multi-colouring reordering on the convergence history of GMRES+LU-SGS solver. Dimension of the Krylov subspace equals to 10.	115
7.7	Effect of the Krylov subspace dimension on the convergence history of GMRES+LU-SGS solver (a) and CPU time (b) for the inviscid circular bump channel test case.	116

7.8	Effect of the CFL number and inner iterations of the linear solver on the computational time to converge the simulation. Comparison between the implicit schemes (Jacobi linear solver (a) and GMRES+LU-SGS (b)) and explicit RK5 scheme for the circular bump channel test.	117
7.9	Influence of the number of grid points in the performance of the Mu^2s^2T solver for the NVIDIA GeForce GTX780Ti GPU and a single CPU core Intel Xeon E5-1620 @ 3.6 GHz.	118
7.10	Solver performance in different graphics cards.	120
7.11	Influence of the message size on the MPI communication data transfer rate. Communications between CPUs and between GPUs are presented.	121
7.12	Speed-up curves of the Mu^2s^2T solver for the T106C test case.	122
7.13	Speed-up curves of the Mu^2s^2T solver for problems of different sizes.	123
7.14	Execution time curves of the Mu^2s^2T solver for the T106C benchmark case.	124
7.15	Influence of the number of grid points in the performance for two processors (CPU: Intel Xeon E5-1620 @ 3.6 GHz, GPU: NVIDIA GeForce GTX780Ti) and different schemes: Explicit RK5, Jacobi with 10 linear iterations, and GMRES+LU-SGS with Krylov subspace dimension equal to 10.	126
7.16	Colour blocks groups after multi-colouring reordering for different problem sizes of the inviscid bump channel.	127
7.17	Speed-up curves of the Mu^2s^2T code for the bump test case on different platforms.	127
7.18	Speed-up curves of the Mu^2s^2T code for the bump benchmark case on different platforms.	129
7.19	Solution of the lid-driven cavity for different Reynolds numbers. Stream-lines (top), velocity profiles (Middle) and convergence histories (Bottom) for a mesh of 150×150 nodes.	131
7.20	Similar solution and Mu^2s^2T simulation results of the non-dimensional velocity profiles for a rotor-stator system of two coaxial disks (a) and convergence histories for different schemes (b).	133

7.21	Comparison between experiments and Mu^2s^2T solver of the non-dimensional velocity profiles (a) and moment coefficient on the rotor (b) of a sealed rotor-stator cavity.	134
7.22	Mach number contours at the mid-span section and mesh detail for the ADTurB vane model. The mesh consists of 191000 nodes.	135
7.23	Convergence histories for a ADTurB vane model (191000 nodes). The CFL numbers for the different solvers: RK5=3.5, Jacobi=15 and GMRES+LU-SGS=60. Linear iterations for the implicit solvers: Jacobi=5 and GMRES+LU-SGS=1. Krylov subspace dimension for GMRES solver=5. . .	136
8.1	Sketch of the model geometry, SAS flows and instrumentation position (taken from Schrewe et al. [85]).	140
8.2	Model and mesh for steady state simulations (Pink: static frame, cyan: rotating frame).	141
8.3	Model and mesh for unsteady state simulations (Pink: static frame, cyan: rotating frame).	141
8.4	Mesh generation detail of the cooling hole, cavities and airfoils.	142
8.5	Convergence history for implicit Jacobi and RK5 solvers of the density L2 norm (left) and the total mass flow balance (right).	145
8.6	Effect of cooling mass flow on static pressure in the Stator 2 front cavity (a) and rear cavity (b) for OP1.	146
8.7	Streamlines within the Stator 2 front cavity colored with $\ \vec{v}\ /\omega r$ for OP1.	147
8.8	Stator 2 end-wall pressure distribution.	148
8.9	Effect of the egression and ingestion upstream seal flow on the hub secondary flows. Streamlines and total pressure contours downstream the Stator 2 (top). Swirl velocity on end-wall streamlines (bottom).	149
8.10	Profile pressure distribution at 5%, 20% and 50% span for MIP, HL and VHL cooling conditions and OP1.	150

8.11	Comparison of contours of slope swirl angle at MP04 between experiments (a) top and multistage CFD simulations (a) top. Circumferentially averaged incidence angle at MP02, MP03 and MP04 (b).	152
8.12	Comparison of contours of radial angle at MP04 between experiments (a) top and multistage CFD simulations (a) top. Circumferentially averaged radial angle at MP02, MP03 and MP04 (b).	153
8.13	Comparison of contours of total pressure loss coefficient at MP04 between experiments (a) top and multistage CFD simulations (a) top. Circumferentially averaged total pressure loss coefficient at MP04 (b). . . .	154
8.14	Snapshot of the entropy field in a time instant for a multi-row unsteady analysis. (a) 3D view of the meridional planes; (b) blade-to-blade plane view.	155
8.15	Contours of total pressure loss coefficient downstream the Stator 2 for experiments (a), multistage (b) and unsteady simulations (c). Circumferentially averaged total pressure loss coefficient downstream the Stator 2 (d).	155
8.16	Contours of total pressure loss coefficient downstream the Stator 2 for experiments (a), 0.3% ISS flow (b) and 1.4% ISS flow (c). Circumferentially averaged total pressure loss coefficient downstream the Stator 2 (d). . . .	156
8.17	Effect of ISS mass flow and static pressure reference on the Stator 2 pressure distribution profiles.	156
B.1	Interpretation of the Arnoldi process to create the Krylov subspace basis. .	178

List of Tables

5.1	Nomenclature for mode classification as a function of the wave numbers. . .	63
6.1	Comparison of representative properties of a modern CPU and a modern GPU.	85
8.1	Mesh size of every passage row (million of nodes).	142
8.2	Relative distances of the Measurement Planes. ($\leftarrow$: upstream the leading edge, $\rightarrow$: downstream the trailing edge).	143
8.3	Rig operating conditions.	143
8.4	Mass flow rates through the rim seals (% Main Annulus Flow).	144
8.5	Estimated, measured and computed mass flows comparisons.	145
9.1	Summary of the speed-up for the methods studied in this work with respect to the baseline solver without multigrid. 1 million mesh point running on a single NVIDIA GeForce GTX780Ti.	161
C.1	Comparison of the asymptotic spectral radius of the amplification matrix for different solvers.	187

Nomenclature

Abbreviations

<i>CFL</i>	Courant-Friedrich-Levy number
<i>CPUs</i>	Central Processing Units
<i>GMRES</i>	Generalised Minimal Residual
<i>GPUs</i>	Graphics Processing Units
<i>HL</i>	High Leakage
<i>IP</i>	Ingestion Point
<i>ISS</i>	Interstage seal
<i>ISS</i>	Interstage seal
ITP	Industria de Turbo Propulsores
JST	Jameson-Schmidt-Turkel scheme
JST	Jameson-Schmidt-Turkel scheme
<i>LPT</i>	Low Pressure Turbine
<i>LU – SGS</i>	Lower-Upper Symmetric Gauss-Seidel
<i>MIP</i>	Maximum Ingestion Point
MPI	Message Passing Interface
MUSCL	Monotonic Upstream-Centered Scheme for Conservation Laws

<i>NGV</i>	Nozzle Guide Vane
<i>OP</i>	Operating Point
RANS	Reyndols-Averaged Navier-Stokes
<i>SAS</i>	Secondary Air system
UPM	Universidad Polit�cnica de Madrid
<i>WK3</i>	Work Package 3

Greek Symbols

δ_{ij}	Kronecher delta	
γ	Ratio of specific heats	
k	Turbulent kinetic energy per unit mass	$[m^2/s^2]$
μ_l	Laminar viscosity	$[Pa \cdot s]$
μ_t	Turbulent viscosity	$[Pa \cdot s]$
Ω	Rotational speed	$[rad/s]$
ρ	Density	$[Kg/m^3]$
τ_{ij}	Viscous stress tensor	$[Pa]$
ϑ_c	Control volume	$[m^3]$
ω	Specific dissipation rate	$[1/s]$

Roman Symbols

a	Speed of sound	$[m/s]$
C_p	Specific heat capacity at constant pressure	$[m^2/(K \cdot s^2)]$
C_v	Specific heat capacity at constant volume	$[m^2/(K \cdot s^2)]$
E	Specific total energy	$[m^2/s^2]$

H	Specific relative total enthalpy	$[m^2/s^2]$
k	Thermal conductivity	$[W/(mK)]$
l_{mix}	Mixing length	$[m]$
LPT	Low Pressure Turbine	$[rad/s]$
M	Mach number	
OX	Abscissa axis	
p	pressure	$[Pa]$
Pr	Prandtl number	
q	Heat flux	$[K]$
r	radius	$[m]$
Re	Reynolds number	
R_g	Specific gas constant	$[J/(Kg \cdot K)]$
S_{ij}	Strain rate tensor	$[1/s]$
T	Temperature	$[K]$
t	Time	$[s]$
U	Conservative variables	
u, v, w	Velocity components	$[m/s]$

Chapter 1

Introduction

1.1 Goal and Thesis Contributions	6
1.2 Literature Review in Implicit Methods	8
1.3 Thesis Outline	10

Modern turbomachinery aerodynamic design is based on the systematic use of Computational Fluid Dynamics (CFD) solvers for the compressible Navier-Stokes (NS) equations. Accuracy, robustness and efficiency are demanded by CFD users in order to achieve more optimum designs in a shorter amount of time. Accuracy can be improved by means of increasing the order of the spatial discretisation and/or accounting for turbulence effects by means of more sophisticated models. Efficiency is enhanced by developing new numerical algorithms for the solution of the Navier-Stokes equations. Finally, robustness is achieved by guaranteeing the convergence of the PDE (Partial Differential Equations), which is based on numerical stability issues and well-posed boundary conditions. All of these aspects need to be kept in mind when a CFD solver is developed.

Geometry complexity and level of details are increasing every day in turbomachinery analysis. The inclusion of under platform and drum cavities, labyrinth seals or over-tip leakage flows, just to mention a few, is leading to larger problem sizes, and higher scale disparity, both in time and space, therefore increasing the difficulty to converge to a steady state solution. There are basically two mechanisms to address these requirements:

Figure 1.1: Growth of supercomputers performance, based on data from top500.org site.

- Firstly, the steady growth of the computational power allows to reduce the turnaround time for a constant number of degrees of freedom, or to run larger cases in the same elapsed time. Moore's law is an observation which establishes that the number of transistors in a dense integrated circuit doubles approximately every two years, which has been corroborated and reported in the TOP500 ranking since 1993 (see Figure 1.1), where supercomputers performance doubles roughly every 14 months. Nevertheless this projection is not valid for an infinite time. Moore himself foresaw a saturation of his own law for the next decade [73].
- Secondly there is continuous improvement on the algorithms used to solve the NS equations. More efficient methods are developed, reducing the number of iterations and the cost per iteration to deliver a numerical solution. This is especially true when the difficulty of the physical problem is increased in parallel. Indeed, algorithmic advances have been comparable with hardware improvements so far. Also the expectations for algorithmic development state that there is a potential factor for one or two orders of magnitude in convergence rates [35].

Therefore, the contribution to the improvement of CFD solvers relies on the investigation

of new numerical algorithms well suited to its implementation in the future hardware architectures. Efficient explicit methods for the resolution of the Navier-Stokes equations were developed during the early eighties, [2]. Later, these methods were improved by introducing acceleration techniques to speed up the convergence to the steady state such as residual smoothing ([1]), Block-Jacobi ([81], [68], [76]), low-Mach number ([37]) and multi-grid preconditioners. For example, Block-Jacobi and multi-grid can typically accelerate the convergence to the steady state by a factor of ten ([40]).

The increase of the computing capability is associated with hardware shifts, and hence the software needs to be re-implemented and adapted to the new architectures. For example, when multi-core processors appeared for the first time, algorithms had to be adapted to perform calculations in parallel, re-implementing the structure of the program using Message Passage Interface (MPI) communications. Frequently this hardware migration is not compatible with certain algorithms, and then the solver can not be re-implemented in the new architectures or its efficiency drops dramatically. Hence, both advances in hardware and algorithms must be considered simultaneously in order to keep improving CFD capabilities.

Finally, an evolution of the convergence time of the Mu^2s^2T solver during the last 17 years is represented in Figure 1.2, where the effect of the algorithmic and hardware improvements has been separated. The blue area represents the computational time to converge the case if no new algorithms would have been developed, and only hardware improvements are considered, whereas the green region takes into account both enhancements. Even though the hardware upgrades have led to a higher computational time reduction than the algorithmic developments, the latter are not negligible, and are essential to have a competitive solver for a state-of-the-art design environment. Between 2000 and 2005, hardware and algorithmic improvements were of the same order, while since 2009 the evolution of the Graphics Cards has exceeded the performance enhancements of the new numerical techniques, because the optimized numerical techniques in CPUs have not been adapted yet to the new architectures efficiently.

Figure 1.2: Evolution of the convergence time of Mu^2s^2T solver during the last 17 years for a 191000 point mesh.

1.1 Goal and Thesis Contributions

The objective of this work is to implement implicit algorithms in an existing state-of-the-art unstructured explicit RANS solver, called Mu^2s^2T [26, 78, 23, 27, 24, 40, 44], which is tailored for solving turbomachinery flows. Over the years, the Mu^2s^2T solver has been developed at ITP S.A. in collaboration with the UPM. Up until this work, the main features of the Mu^2s^2T solver are:

- Unstructured edge-based data structure to compute the convective and viscous fluxes.
- Finite volume formulation.
- Two different second order spatial discretisations:
 - JST scheme [2].
 - MUSCL scheme [93].
- Marched in time using an explicit five-stage Runge-Kutta scheme, where the viscous terms and the artificial viscosity are evaluated only in the odd stages [56].
- Low Mach number preconditioning to tackle the incompressible limit [37].
- Block-Jacobi preconditioning and multigrid to accelerate the steady-state convergence[68].
- Turbulence effects modeled with either a mixing length algebraic model [6] or the $k - \omega$ model of Wilcox 2006 [32].

- Dual time step technique for solving unsteady flows, such as vibrating blades or rotor-stator interactions.
- 1D or 2D [23][42] non-reflecting boundary conditions for inlets and outlets.
- Efficient sliding-plane and Phase-lagged boundary conditions for rotor-stator simulations [15].
- Parallelised using MPI for multiple processors.

Even though a well-refined CFD solver existed at the beginning of this thesis, and this was routinely used for turbine and compressor aerodynamic design, some shortcomings could still be encountered related with the convergence to reach the steady-state of complex flows. Firstly, due to the explicit nature of the solver physical hydrodynamic instabilities, such as vortex shedding, were difficult to eliminate and precluded the convergence to a truly steady state. This behaviour causes difficulties in the repeatability of the design work. This phenomenon is clearly seen as well in cavity flows, where typically counter-rotating vortical structures at low Mach number are present and tend to oscillate. Moreover, when the computational domain is not smooth and many details are included, sometimes the mesh quality obtained is not good enough. Under these circumstances, the agglomeration algorithm that forms an integral part of the multi-grid method, can lead to very distorted and coarse cells, making the multigrid solver prone to fail, and degrading the convergence rate.

Implicit methods were envisioned as a solution to overcome the aforementioned limitations and have been implemented in this thesis in the Mu^2s^2T code. The main requirement for the new solution strategy is to preserve exactly the same numerical results as the baseline solver, in order to maintain the code legacy. Hence, comparing the new results against the pre-existing code results, would be considered as a validation of the proposed method. Nevertheless, as a part of a European founded project, so called MAGPI (Main Annulus Gas Path Interactions), the CFD tool has been used to study the effects of the secondary air flows on the main stream and compared with new available experimental data. This validation is essential, because purge flows are not considered during a standard airfoil design and there is not enough experience with the performance of a RANS CFD solver

in this type of configurations. A CFD analysis is much more complicated when the cavities of a turbine are included in the computational domain, as different time scales are considered, such as the residence time of a particle in the main stream, the residence time within the cavity where the swirl velocity is much larger than the meridional velocity or the residence time of the flow through a labyrinth seal. This disparity of scales makes the problem very stiff, and a large number of time steps are needed to converge these cases with the baseline explicit solver. Finally, the implementation of an efficient implicit solver would make affordable during a typical aerodynamic design process the addition of complex parts of the turbine.

1.2 Literature Review in Implicit Methods

Implicit methods have been widely used and developed as well for CFD during the last three decades. Firstly, they were implemented on structured grids ([47]), where the LU decomposition was applied to the multi-dimensional Euler equations, deriving a system of tridiagonal matrices to be inverted. Next the Alternating Direction Implicit (ADI) scheme was applied to the Euler and Navier-Stokes equations in [98], in which implicit line treatment in each grid direction is performed. Later on, a multigrid method is combined with the LU implicit scheme and symmetric successive overrelaxation (SSOR) was applied to the Euler equations in [48]. Next year, this scheme was extended to the Navier-Stokes equations and LU-SGS scheme ([99]) was introduced. A multigrid relaxation scheme with LU-SGS solver is an alternative to the ADI scheme, which is very efficient in two dimensions but its performances are limited in three dimensions.

First applications of implicit methods on unstructured grids are detailed in [95], where a direct method was used to solve the system of linear equations, providing a robust solution technique but with a non-optimal computational complexity and high storage requirements. In the next decade, efficient implicit solvers were completely developed for unstructured grids. Venkatakrishnan and Mavriplis [96] introduced the GMRES as the iterative method to solve the first order linearized equations and the importance of a good preconditioner to be GMRES was pointed out as a key element to obtain

competitive solver. ILU, Jacobi and SSOR were investigated, being the former the most competitive. A good summary of implicit schemes on unstructured grids can be found in [94]. Both steady and unsteady problems are addressed and their parallel implementation issues are mentioned, corroborating the good performances of the implicit GMRES preconditioned by ILU. Matrix-free Newton-Krylov methods were successfully implemented and parallelised on unstructured grids in [7], where the matrix-vector product is approximated by finite differences. The LU-SGS scheme was firstly developed for unstructured grids in [87], where a grid reordering is proposed to make the code vectorizable. Next year, in 1998, this algorithm was used as a GMRES preconditioner in [60], showing a superior performance to GMRES+ILU scheme in terms of CPU time. Later on, many authors have been developing implicit methods [77, 66, 10, 20, 61, 36], with the common denominator of using a matrix-free GMRES technique.

Implicit methods have also been used in an unstructured solver applied to turbomachinery configurations (AU3D [83]), where the implicit first-order Jacobian linearised equations are solved with the Jacobi solver.

Recent works have been done to improve the implicit integration of the $\kappa - \omega$ equations ([67]), where the design of the Jacobian matrix is made to guarantee the positivity of the turbulence equation dependent variables for any time step.

A similar work done in this thesis has been carried out in [36], where implicit methods were investigated and implemented in DLR TAU code, which originally marched in time with a 3-stage Runge-Kutta scheme. Two different approaches were finally remarked as successful from a practical point of view. The first one is based on a novel formulation of the LU-SGS scheme with FAS multigrid and an approximate factorization which has identical memory requirements to the standard explicit Runge-Kutta scheme, yielding a factor of two in terms of CPU convergence time. The second proposal is the so called FOKI (First Order Krylov Implicit) method, which is based on a first order Jacobian and ILU preconditioning to GMRES. FOKI was shown to be 4-5 times faster than LU-SGS but consuming five times more memory and only was tested in 2D cases.

1.3 Thesis Outline

This thesis consists of two main parts. In the first part (from Chapter 2 to 7), a description of the implicit CFD solver is presented, as well as issues related with its implementation in GPUs and code benchmarking. In the second part (Chapter 8), the new implicit solver is used to simulate the experiments tested in a Low Speed Turbine Rig at Technische Universität Darmstadt (TUD) in order to validate the CFD solver and understand the interaction between the cavity leakages and the main annulus stream flows.

Firstly, there is a description of the baseline solver for the Navier-Stokes equations, which is the starting point for this thesis. The two different turbulence models used in the solver, namely: the Wilcox 2006 $\kappa - \omega$ two-equation turbulence model and an algebraic zero-equation model are outlined. The spatial and temporal discretisations are detailed, as well as the boundary condition treatment. Finally, the preconditioning techniques used in the solver are mentioned.

Secondly, in Chapter 3 the implicit solver is presented. The first-order Jacobian matrix construction and how its product with a vector is performed within the edge-based data structure are described. Additionally, the implicit treatment of the boundary conditions at the inlets and outlets is detailed.

Next, in Chapter 4 a summary of the linear solvers implemented in the code is described. Special attention is paid to the implementation of the different linear solvers within the edge-based data structure framework.

In Chapter 5 a throughout stability analysis is presented, comparing the implicit methods studied in this work against their pre-existing explicit Runge-Kutta scheme. The different non-dimensional parameters that appear in the problem are studied to understand the performances of the algorithms as a function of the flow regime, mesh stretching and artificial dissipation. From this analysis, an estimate of the non-dimensional computational time required to converge the numerical solution is derived, corroborating this results in Chapter 7. Moreover, the optimum numerical parameters to converge the RANS equations are obtained.

Chapter 6 describes the parallelisation strategy. Both, distributed and shared

memory parallelism of the solver are addressed. Distributed memory parallelisation is well documented in the literature. Typically for CFD applications the inter-processor communications are carried out via MPI. On the other hand, shared memory parallelisation, which is done when running a loop in multi-threads, is not frequent amongst the CFD community. The pre-existing explicit solver is implemented in a mixture of C++ and openCL to enable the execution in both, CPUs and GPUs ([44]). Most of the loops used in the implicit solver have been implemented in OpenCL following the same strategy as its explicit counterpart, except for the parallelisation of the LU-SGS scheme, where a multi-colouring reordering is done to obtain efficient parallelism of the algorithm in GPUs devices.

Chapter 7 presents the numerical performances of the implicit solver. Basic well-known models in the CFD community are used to assess the convergence rates of the implicit solver developed in this work. Furthermore, the GPU acceleration and parallel performances are studied, obtaining better scalability for the implicit than for the explicit counterpart.

Finally in Chapter 8, the code is validated with experimental data provided by TUD, as part of the EU FP6 MAGPI (Main Annulus Gas Path Interactions) programme. The goal here was to improve the understanding of the interactions between cooling and main stream flows in a two-stage low-speed turbine rig. Three-dimensional steady and unsteady simulations of the whole rig, including the stator-well cavities are performed, covering three types of sealing conditions: flow ingestion from the main stream to the cavities, marginal ingestion and very high purge flow from the cavities to the main stream. A side objective was to demonstrate the capability of the algorithm to deal with complex configurations.

Chapter 2

3D Unstructured Base Solver

2.1	Introduction	13
2.2	Reynolds-Averaged Navier-Stokes equations	14
2.3	Turbulence Model	16
2.4	Dimensionless Reynolds-Averaged Navier-Stokes equations .	18
2.5	Spatial Discretisation	19
2.6	Boundary Conditions	25
2.7	Explicit Time Integration	31

2.1 Introduction

The aim of this chapter is to provide a brief introduction to the main features of an existing Reynolds-Averaged Navier-Stokes equations solver, which is the result of several works, mainly due to Dr. Gómez, to Dr. Gisbert and also to the personnel of the Technology and Methods Department at ITP [26, 78, 23, 27, 24, 40, 44]. The developments presented in this thesis, especially those related with implicit methods and its implementation in GPUs, are based on this previously existing solver.

The equations are solved in arbitrary domains which are discretised using unstructured grids. The data structure used to store the grid and the solution strongly influences the

development of the algorithms that are presented along this work. In the next sections, the discretisation of the Navier-Stokes PDEs in unstructured meshes and the edge-based data structure used to store the necessary data to solve them will be outlined. Then the explicit time marching algorithm is presented. Finally, the turbulence models used in the solver are introduced.

2.2 Reynolds-Averaged Navier-Stokes equations

The differential form of the Navier-Stokes equations in the rotating frame of reference along the OX axis with a velocity Ω can be written in terms of the relative velocity as:

$$\frac{\partial U}{\partial t} + \frac{\partial (F_c - F_v)}{\partial x_j} + S_\Omega = 0, \quad j = 1, 2, 3 \quad (2.1)$$

where the summation convection has been employed.

The vector U denotes the conservative variables vector:

$$U = \begin{Bmatrix} \rho \\ \rho u_j \\ \rho E \end{Bmatrix} \quad (2.2)$$

being ρ the fluid density, $u_j = \{u, v, w\}^T$ the cartesian relative velocity components and E is the specific relative total energy, which for a perfect gas is:

$$E = C_v T + \frac{u_k^2 - \Omega^2 r^2}{2} \quad (2.3)$$

being C_v the specific heat capacity at constant volume, $u_k^2 = u^2 + v^2 + w^2$ and r is the radius $\sqrt{y^2 + z^2}$.

The inviscid vector flux is:

$$F_c = \begin{Bmatrix} \rho u_j \\ \rho u_j u_i + \delta_{ij} p \\ \rho u_j H \end{Bmatrix} \quad (2.4)$$

where δ_{ij} is the Kronecker delta, p is the pressure. H denotes the specific relative total enthalpy, also known as the rothalpy, that for a perfect gas is:

$$H = E + \frac{p}{\rho} \quad (2.5)$$

The diffusion flux vector is given by:

$$F_v = \begin{pmatrix} 0 \\ \tau_{ij} \\ u_j \tau_{ij} - q_j \end{pmatrix} \quad (2.6)$$

The viscous stress tensor, τ_{ij} , can be written as:

$$\tau_{ij} = (\mu_l + \mu_t) \left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \delta_{ij} \frac{\partial u_k}{\partial x_k} \right] \quad (2.7)$$

where μ_l is the laminar viscosity and μ_t is the turbulent viscosity which is given by an additional turbulence model. The expression of the laminar viscosity is given by the Sutherland's law [88]:

$$\mu_l = \frac{a_s T^{3/2}}{T + b_s} \quad (2.8)$$

where the values of a_s and b_s constants for air are $1.458 \cdot 10^{-6} \text{ Nsm}^{-2} \text{ K}^{-1/2}$ and 110.4 K respectively.

The heat conduction term is given by the Fourier's Law:

$$q_j = -k \frac{\partial T}{\partial x_j} \quad (2.9)$$

being T the fluid temperature and k the thermal conductivity, which is defined as:

$$k = \left(\frac{\mu_l}{Pr_l} + \frac{\mu_t}{Pr_t} \right) C_p \quad (2.10)$$

The laminar and turbulent Prandtl numbers for air are 0.72 and 0.9 respectively. C_p is the specific heat capacity at constant pressure.

These equations are completed by the equation of state, that for an ideal gas is:

$$p = \rho R_g T \quad (2.11)$$

being R_g the specific gas constant, which can be related with the C_p as:

$$R_g = C_p \frac{\gamma - 1}{\gamma} \quad (2.12)$$

where γ is the ratio of specific heats.

$$\gamma = \frac{C_p}{C_v} \quad (2.13)$$

The pressure can be expressed as a function of the conservative variables as

$$p = \rho(\gamma - 1) \left(E - \frac{u_k^2 - \Omega^2 r^2}{2} \right) \quad (2.14)$$

Finally, the source terms S_Ω represents the Coriolis force $2\rho\Omega(0, -w, v)$ and the centripetal force $-\rho\Omega(0, y, z)$.

2.3 Turbulence Model

Turbulence effects are modelled using either Wilcox 2006 $k - \omega$ two-equation turbulence model or an algebraic zero-equation model.

2.3.1 Algebraic Model

The zero-equation algebraic model achieves the closure by relating the Reynolds stresses directly to mean-flow properties. The eddy viscosity is defined as:

$$\mu_{t,inner} = \rho l_{mix}^2 |\omega| \quad (2.15)$$

$|\omega|$ is the magnitude of the vorticity

$$|\omega| = \sqrt{\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial x} - \frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z} - \frac{\partial w}{\partial y}\right)^2} \quad (2.16)$$

l_{mix} is the mixing length which is defined by using a two-layer formulation:

$$l_{mix} = k \cdot \min(y_n, C_{pitch} \cdot Pitch) \quad (2.17)$$

Where y_n is the distance to the nearest wall (Fig. 2.1). The constant of the inner layer is $k = 0.4$.

This model does not compute the transition itself and it is set in a pre-processing step based on experimental correlations. Laminar regions are enforced by setting $l_{mix} = 0$.

Within this simplification, the inner layer only takes into account the log layer whereas the viscous sublayer is not considered to increase the robustness of the scheme. The model is validated by tuning the constant C_{pitch} in order to predict some overall parameters such as 2D or 3D losses.

Figure 2.1: Sketch of the simplified mixing length function.

2.3.2 Wicox 2006 $k - \omega$ Turbulence Model

$k - \omega$ turbulence model equations consists of the transport and diffusion of the turbulent kinetic energy k and the specific dissipation rate ω . These equations have a similar form to the Navier-Stokes equations, including some source terms:

$$\frac{\partial}{\partial t} \begin{Bmatrix} \rho k \\ \rho \omega \end{Bmatrix} + \frac{\partial}{\partial x_j} \begin{Bmatrix} \rho u_j k \\ \rho u_j \omega \end{Bmatrix} - \frac{\partial}{\partial x_j} \begin{Bmatrix} (\mu_l + \sigma_k \mu_t) \frac{\partial k}{\partial x_j} \\ (\mu_l + \sigma_\omega \mu_t) \frac{\partial \omega}{\partial x_j} \end{Bmatrix} = \begin{Bmatrix} P_k - \beta^* \rho k \omega \\ \alpha \frac{\omega}{k} P_k - \beta \rho \omega^2 + C_D \end{Bmatrix} \quad (2.18)$$

The closure coefficients are $\sigma_k = 0.6$, $\sigma_\omega = 0.5$, $\beta^* = 0.09$, $\beta = 0.0708$, $\alpha = 0.52$ and $\sigma_d = 0.125$.

The production term P_k is given by:

$$P_k = 2\mu_t S_{ij} \frac{\partial u_i}{\partial x_j} - \frac{2}{3}\mu_t \left(\frac{\partial u_k}{\partial x_k} \right)^2 - \frac{2}{3}\rho k \frac{\partial u_k}{\partial x_k} \quad (2.19)$$

where $S_{ij} = (\partial u_i / \partial x_j + \partial u_j / \partial x_i) / 2$ is the strain rate tensor.

The cross-diffusion term C_D is given by:

$$C_D = \sigma_d \frac{\rho}{\omega} \max \left(\frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j}, 0 \right) \quad (2.20)$$

This cross-diffusion term was introduced as a remedy for the original $k - \omega$ model's sensitivity to the free stream value of ω , the incorrect near-wall asymptotic behaviour is also eliminated. With the addition of this term the predictions for free shear flows are clearly improved.

The turbulent viscosity is:

$$\mu_t = \frac{\rho k}{\tilde{\omega}}, \quad \tilde{\omega} = \max \left(\omega, C_{lim} \sqrt{\frac{2S_{ij}S_{ij}}{\beta^*}} \right) \quad (2.21)$$

with $C_{lim} = 7/8$.

A realizability constraint for the turbulent viscosity is added to improve numerical robustness and to prevent anomalous growth of turbulent kinetic energy, i.e. that appears in stagnation regions. Also the non-physical turbulent viscosity production is avoided, yielding larger separation bubbles and improving incompressible and transonic flow predictions.

2.4 Dimensionless Reynolds-Averaged Navier-Stokes equations

Since the RANS equations are a system of the continuity, momentum and energy conservations, there is a disparity in several orders of magnitude between these equations. Therefore, it is convenient to non-dimensionalise the system in order to have similar quantities to reduce round-off errors.

The variables to non-dimensionalised are a reference length L , a reference density ρ_∞ , a reference speed of sound a_∞ and a reference viscosity μ_∞ . The different variables that appears in the problem in dimensionless form are:

$$\begin{aligned} x^* &= \frac{x}{L}, & t^* &= \frac{t}{L/a_\infty}, & \rho^* &= \frac{\rho}{\rho_\infty}, & u_j^* &= \frac{u_j}{a_\infty} \\ p^* &= \frac{p}{\rho_\infty a_\infty^2}, & E^* &= \frac{E}{a_\infty^2}, & \mu_l^* &= \frac{\mu_l}{\mu_\infty}, & \mu_t^* &= \frac{\mu_t}{\mu_\infty} \\ T^* &= \frac{T}{a_\infty^2/R_g}, & q^* &= \frac{q}{\mu_\infty a_\infty^2/L}, & a_s^* &= \frac{a_s}{\mu_\infty \sqrt{R_g/a_\infty}}, & b_s^* &= \frac{b_s}{a_\infty^2/R_g} \end{aligned}$$

Omitting the asterisks for the sake of clarity, the non-dimensional governing equations take the same structure as Equation 2.1. The dimensionless conservative variables after omitting the asterisks present the same form as Equation 2.2.

The specific total energy yields:

$$E = \frac{T}{\gamma - 1} + \frac{u_k^2}{2} \quad (2.22)$$

The convective vector flux remains unmodified after the non-dimensionalisation, whereas the viscous fluxes are:

$$F_v = \frac{M_\infty}{Re_\infty} \begin{pmatrix} 0 \\ \tau_{ij} \\ u_j \tau_{ij} - q_j \end{pmatrix} \quad (2.23)$$

The parameter that appears after the non-dimensionalisation is the ratio between the reference Mach number, M_∞ , and the Reference Reynolds number, Re_∞ :

$$\frac{M_\infty}{Re_\infty} = \frac{u_\infty/a_\infty}{\rho_\infty u_\infty L/\mu_\infty} = \frac{\mu_\infty}{\rho_\infty a_\infty L} \quad (2.24)$$

The expressions for the viscous stress tensor and the heat conduction are not modified after the non-dimensionalisation and the thermal conductivity is:

$$k = \left(\frac{\mu_l}{Pr_l} + \frac{\mu_t}{Pr_t} \right) \frac{\gamma}{\gamma - 1} \quad (2.25)$$

And finally, the state equation is simplified to:

$$p = \rho T \quad (2.26)$$

The only modification for the algebraic turbulence model is the dimensionless turbulent viscosity, yielding:

$$\mu_{t,inner} = \rho l_{mix}^2 |\omega| \frac{Re_\infty}{M_\infty} \quad (2.27)$$

Regarding the $k - \omega$ turbulence model, Equation 2.18 is non-dimensionalised as:

$$\frac{\partial}{\partial t} \begin{pmatrix} \rho k \\ \rho \omega \end{pmatrix} + \frac{\partial}{\partial x_j} \begin{pmatrix} \rho u_j k \\ \rho u_j \omega \end{pmatrix} - \frac{\partial}{\partial x_j} \begin{pmatrix} (\mu_l + \sigma_k \mu_t) \frac{\partial k}{\partial x_j} \\ (\mu_l + \sigma_\omega \mu_t) \frac{\partial \omega}{\partial x_j} \end{pmatrix} \frac{M_\infty}{Re_\infty} = \begin{pmatrix} P_k \frac{M_\infty}{Re_\infty} - \beta^* \rho k \omega \frac{Re_\infty}{M_\infty} \\ \alpha \frac{\omega}{k} P_k \frac{M_\infty}{Re_\infty} - \beta \rho \omega^2 \frac{Re_\infty}{M_\infty} + C_D \frac{M_\infty}{Re_\infty} \end{pmatrix} \quad (2.28)$$

where the non-dimensional turbulent kinetic energy and the specific dissipation rate are:

$$k^* = \frac{k}{a_\infty^2}, \quad \omega^* = \frac{\omega}{\rho_\infty a_\infty^2 / \mu_\infty}$$

The production term P_k after the non-dimensionalisation becomes:

$$P_k = 2\mu_t S_{ij} \frac{\partial u_i}{\partial x_j} - \frac{2}{3} \mu_t \left(\frac{\partial u_k}{\partial x_k} \right)^2 - \frac{2}{3} \rho k \frac{\partial u_k}{\partial x_k} \frac{Re_\infty}{M_\infty} \quad (2.29)$$

2.5 Spatial Discretisation

The solver uses hybrid unstructured grids to discretise the entire spatial domain with a single homogeneous data structure, in contrast with block-structured methods, and may

Figure 2.2: Hybrid cell grid and associated dual mesh.

contain cells with an arbitrary number of faces. The discrete equations are obtained from Equation 2.1, using a finite volume formulation, where the control volumes are constructed making use of the dual mesh (see Figure 2.2), that is, the mesh that results from connecting the centroids of the cells that surround a node. [57][78]. Then, the Navier-Stokes equations in conservative form for an arbitrary control volume may be written as

$$\frac{d}{dt} \int_{\vartheta} U d\vartheta + \int_{\Sigma} (F_c - F_v) \cdot n d\sigma \quad (2.30)$$

where ϑ is the flow domain, Σ is its boundary, n is the normal vector pointing outward to the boundary and $d\sigma$ is the differential area.

The solution vector is stored at the vertices of the cells. For the internal node i , the semidiscrete form of the system of non-linear Equation 2.30 can be written using a finite volume approach as

$$\frac{d(\vartheta_{c,i} U_i)}{dt} = R_i(U) = C_i(U) + D_i(U), \quad i = 1, \dots, \#nodes \quad (2.31)$$

where the residual has been split into the convective part, C , and the viscous and numerical diffusion terms, D . ϑ_i is the the control volume of the dual cell i .

2.5.1 Evaluation of the convective terms

The fluxes over the control volume faces are obtained computing the semi-sum of the fluxes in each node, which yields a second-order cell-centered scheme in the dual mesh.

Figure 2.3: Hybrid boundary cell grid and associated dual mesh.

Hence, the convective part is:

$$C_i = - \sum_{j=1}^{\#edges_i} \frac{1}{2} (F_{c,i} + F_{c,j}) n_{ij} \sigma_{ij} - \sum_{j=1}^{\#Bedges_i} F_{c,i} n_i \sigma_i \quad (2.32)$$

where σ_{ij} represents the area associated to the edge ij that joins the node i with its neighbours, and $\#edges_i$ and $\#Bedges_i$ are the number of edges and boundary edges that reach that node. Note that the flux over a boundary face of the dual control cell centered in the node i , is the flux of the node i itself (Figure 2.3). This is important to keep in mind to build the Jacobian matrix.

2.5.2 Evaluation of the viscous terms

Evaluating the viscous terms requires the computation of the gradient of the conservative variables:

$$\nabla U_i = \frac{1}{\vartheta_{c,i}} \left(\sum_{j=1}^{\#edges_i} \frac{1}{2} (U_i + U_j) n_{ij} \sigma_{ij} + \sum_{j=1}^{\#Bedges_i} U_i n_i \sigma_i \right) = \frac{1}{\vartheta_{c,i}} \sum_{j=1}^{\#edges_i} \frac{1}{2} (U_j - U_i) n_{ij} \sigma_{ij} \quad (2.33)$$

where the second expression is mathematically equivalent to the first, and the evaluation of a loop over the boundary edges is avoided.

The viscous fluxes are evaluated making use of the gradients, and hence the value of the gradient in the area where the flux is evaluated must be obtained. The simplest approach is taking the average value between the two points of the edge:

$$\overline{\nabla U_{ij}} = \frac{1}{2} (\nabla U_i + \nabla U_j) \quad (2.34)$$

but this solution does not prevent the odd-even decoupling of high frequency error modes. This is avoided by the numerical diffusion in zones where the viscous terms are not dominant, but in the boundary layer zone that contribution is not enough to prevent the appearance of high frequency modes. The gradient in the edge direction is then substituted by the difference of the variables along the edge:

$$\nabla U_{ij} = \overline{\nabla U_{ij}} - \left(\overline{\nabla U_{ij}} \cdot l_{ij} - \frac{U_j - U_i}{|x_j - x_i|} \right) l_{ij} \quad (2.35)$$

where l_{ij} is the unit vector:

$$l_{ij} = \frac{x_j - x_i}{|x_j - x_i|} \quad (2.36)$$

2.5.3 Artificial viscosity construction based on the pseudo-Laplacian (JST)

High frequency errors are neither damped nor propagated in a central difference scheme and some amount of numerical diffusion is required to stabilize the method. The artificial viscosity part of the diffusion term in Equation 2.31 based on the pseudo-Laplacian can be expressed as

$$D_{av,i} = - \sum_{j=1}^{\#edges_i} \frac{1}{2} |A_{ij}| \Delta U \sigma_{ij} \quad (2.37)$$

where A_{ij} is the Jacobian matrix of the inviscid fluxes:

$$A_{ij} = \frac{\left(\partial F_c|_i + \partial F_c|_j \right)}{\partial U} \cdot n_{ij} \quad (2.38)$$

The absolute value of the Jacobian matrix is defined as:

$$|A_{ij}| = T_{ij} |\Lambda_{ij}| T_{ij}^{-1} \quad (2.39)$$

The transformation matrices are detailed in [40], and the matrix $|\Lambda_{ij}|$ is the diagonal matrix which is made up of the absolute value of the Jacobian matrix eigenvalues:

$$|\Lambda_{ij}| = \begin{bmatrix} |v_{ij} \cdot n_{ij}| & 0 & 0 & 0 & 0 \\ 0 & |v_{ij} \cdot n_{ij}| & 0 & 0 & 0 \\ 0 & 0 & |v_{ij} \cdot n_{ij}| & 0 & 0 \\ 0 & 0 & 0 & |v_{ij} \cdot n_{ij} + a_{ij}| & 0 \\ 0 & 0 & 0 & 0 & |v_{ij} \cdot n_{ij} - a_{ij}| \end{bmatrix} \quad (2.40)$$

When the matrix $|\Lambda_{ij}|$ is simplified substituting the diagonal elements by the largest of them, Equation 2.39 becomes:

$$|A_{ij}n_{ij}| = (|v_{ij} \cdot n_{ij}| + a_{ij}) I_{d,5 \times 5} \quad (2.41)$$

which is the so-called scalar artificial formulation.

The term ΔU in Equation 2.37 is a blend of second and fourth differences [2]:

$$\Delta U = S_{ij} (U_i - U_j) - \frac{1}{2} (1 - \kappa) (1 - S_{ij}) (L_i - L_j) \quad (2.42)$$

The expression L_i stands for the pseudo-Laplacian operator:

$$L_i = \frac{1}{\#edges_i} \left[\sum_{j=1}^{\#edges_i} (U_j - U_i) + \sum_{j=1}^{\#edgesTBi} (U_i - U_j) \right] \quad (2.43)$$

The second term of the pseudo-Laplacian is applied to the inner edges touching a boundary node, in order to avoid asymmetry on the boundaries.

S_{ij} is a switch between the second and first-order schemes and is used to avoid spurious oscillations of the solution in the vicinity of shock waves, where high frequency harmonics are neither convected nor damping and wiggles tend to appear. In such regions, the fourth differences are disabled and only the second ones are used, activating a first order discretisation. This switch is based on the detection of the second derivative of the pressure by means of its pseudo-Laplacian:

$$S_{ij} = \min \left[1, \epsilon \left(\frac{L_i(p)}{p_i} + \frac{L_j(p)}{p_j} \right) \right] \quad (2.44)$$

where ϵ is a constant set to 8. Finally, κ is a constant to scale the fourth differences, which is set to 2/3 through this work.

Figure 2.4: Piecewise parabolic reconstruction.

2.5.4 MUSCL approach

MUSCL scheme (Monotonic Upstream-Centered Scheme for Conservation Laws) is a second-order scheme which is achieved by using upwind-biased interpolations of the solution U , U_i^+ and U_j^- (see Figure 2.4), yielding:

$$C_i + D_{av,i} = -\frac{1}{2} \sum_{j=1}^{\#edges_i} \left[\left(F_c(U_i^+) + F_c(U_j^-) \right) n_{ij} - \left| A_{ij}(U_i^+, U_j^-) n_{ij} \right| (U_j^+ - U_i^-) \right] \sigma_{ij} - \sum_{j=1}^{\#Edges_i} F_{c,i} n_i \sigma_i$$

where the interpolated variables are defined as:

$$\begin{aligned} U_i^+ &= U_i + \frac{S_i}{4} \left[(1 - \kappa_2 S_i) \Delta_i^- + (1 + \kappa_2 S_i) (U_j - U_i) \right] \\ U_j^- &= U_j - \frac{S_j}{4} \left[(1 - \kappa_2 S_j) \Delta_j^+ + (1 + \kappa_2 S_j) (U_j - U_i) \right] \end{aligned}$$

being $\kappa_2 = 1/3$. The operators Δ_i^- and Δ_j^+ are:

$$\begin{aligned} \Delta_i^- &= 2\nabla U_i \cdot (x_j - x_i) - (U_j - U_i) \\ \Delta_j^+ &= 2\nabla U_j \cdot (x_j - x_i) - (U_j - U_i) \end{aligned}$$

Finally, the values S_i and S_j are the limiters to avoid oscillations next to the shock waves and make the scheme TVD (Total Value Diminishing). The Van Albada 2 limiter is used:

$$\begin{aligned} S_i &= \max \left[0, \frac{2\Delta_i^-(U_j - U_i) + \delta}{(\Delta_i^-)^2 + (U_j - U_i)^2 + \delta} \right] \\ S_j &= \max \left[0, \frac{2\Delta_j^+(U_j - U_i) + \delta}{(\Delta_j^+)^2 + (U_j - U_i)^2 + \delta} \right] \end{aligned}$$

where δ is a small value to prevent the division by zero in smooth zones.

2.6 Boundary Conditions

In order to cope with the complexity, several types of boundary conditions are implemented in the code: periodic, inlet, outlet, solid wall, mixing plane, sliding plane and phase-lagged periodicity.

The solid wall boundary conditions are imposed making use of the volumes associated to the wall nodes. For viscous cases with a fixed temperature at the wall, only the mass equation is marched in time imposing that the mass flux at the solid wall is null. If the heat flux is imposed then the energy equation is marched in time making use of this condition. In any case the momentum equation is not used since it is substitute by the condition that the velocity vanishes (or is known) at the wall. For inviscid cases all the equations are marched in time at the solid walls by imposing that the net flux of mass and energy is null and specifying that the contribution of the wall to the momentum equations is only due to the static pressure.

For internal flows, the characteristics theory is used to specify the boundary conditions. For steady cases the radial distribution of stagnation pressure, stagnation temperature and flow angles are imposed at the inlet while the static pressure is specified at the exit. This may be done in a point-wise basis (1D boundary conditions) or as a mean (2D exact steady, non-reflecting boundary conditions [23], [42]). The rest of information is computed making use of the semi-volumes associated to the nodes. 1D and 2D unsteady non-reflecting boundary conditions are also available.

For a cascade vibrating in a travelling-wave mode or a rotor/stator configuration with different number of vanes and blades, phase-lagged boundary conditions are applied in the steady periodic boundaries to perform unsteady computations using a single passage [17].

The different boundary conditions used in the code are detailed in the following subsections. Regarding inlets and outlets, only the formulation for 1D steady non-reflecting boundary conditions are described.

2.6.1 Periodic Boundary Condition

The fluxes of the periodic nodes need to be completed with the fluxes of their homologous nodes. This is depicted in Figure 2.5, where the lower periodic node should have below the dual cell of its corresponding upper periodic node. In the case of axi-symmetric periodicity, the homologous dual cell must be rotated an angle equal to the sector pitch α . Moreover, the homologous fluxes are computed with the velocity and normal areas rotated as well. This is depicted in Figure 2.5 (b). The only components of the fluxes which need to be corrected to account for the axi-symmetry of the boundary condition are the second and third components of the momentum equation, assuming that the axis of symmetry coincides with the first axis direction. The final lower flux, according with Figure 2.5 (b) is:

$$\begin{pmatrix} R_{1,l} \\ R_{2,l} \\ R_{3,l} \\ R_{4,l} \\ R_{5,l} \end{pmatrix}^c = \begin{pmatrix} R_{1,l} \\ R_{2,l} \\ R_{3,l} \\ R_{4,l} \\ R_{5,l} \end{pmatrix} + \begin{pmatrix} R_{1,u} \\ R_{2,u} \\ R_{3,u} \cdot \cos \alpha + R_{4,u} \cdot \sin \alpha \\ -R_{3,u} \cdot \sin \alpha + R_{4,u} \cdot \cos \alpha \\ R_{5,u} \end{pmatrix} \quad (2.45)$$

Once, the flux in the lower node is correct, it can be rotated to recover the proper flux in the upper node:

$$\begin{pmatrix} R_{1,u} \\ R_{2,u} \\ R_{3,u} \\ R_{4,u} \\ R_{5,u} \end{pmatrix}^c = \begin{pmatrix} R_{1,l} \\ R_{2,l} \\ R_{3,l} \cdot \cos \alpha - R_{4,l} \cdot \sin \alpha \\ R_{3,l} \cdot \sin \alpha + R_{4,l} \cdot \cos \alpha \\ R_{5,l} \end{pmatrix}^c \quad (2.46)$$

2.6.2 1D steady non-reflecting inlet boundary condition

Subsonic Inlet

In the case of a subsonic inlet, there are four waves which are entering into the computational domain through the boundary and only one wave which is coming out.

Figure 2.5: Periodic nodes and the dual cells. Cartesian (a) and asi-symmetric (b) periodicity.

The philosophy of the 1D boundary condition is to assume locally a one-dimensional flow in the normal direction to the boundary and use an estimate of the tangential component of the velocity to the boundary. The stagnation pressure, stagnation temperature, and flow angles are imposed in all the mesh points of the inlet. To complete the information of the node, the outcoming characteristic is used:

$$R^- = v_n - \frac{2a}{\gamma - 1}, \quad (2.47)$$

where v_n is the normal component of the velocity to the inlet surface, $v_n = \mathbf{v} \cdot \mathbf{n}$, and the speed of sound, a :

$$a = \sqrt{\gamma \frac{P}{\rho}}$$

The invariant R^- is calculated from the inner scheme transformed into its characteristic form. Indeed, if the differential equation after the spatial discretisation for the node i is of the form:

$$V_{c,i} \frac{\partial U_i}{\partial t} = R_i, \quad (2.48)$$

the time derivative in terms of the invariants is:

$$\vartheta_{c,i} \frac{\partial U}{\partial \phi^{inlet}} \frac{\partial \phi_n^{inlet}}{\partial t} = R_i \quad (2.49)$$

being ϕ^{inlet} a set of variables to define the inlet:

$$\phi^{inlet} = \begin{pmatrix} P_t \\ T_t \\ \alpha_s \\ \alpha_r \\ R^- \end{pmatrix} \quad (2.50)$$

Inverting the matrix $\partial u / \partial \phi^{inlet}$, the ordinary differential equation for the invariant R^- is:

$$\vartheta_{c,i} \frac{\partial R^-}{\partial t} = \sum_{k=1}^5 \frac{\partial R^-}{\partial u_k} R_{i,k} \quad (2.51)$$

In Equation 2.51, R_i represents the residual of the node i without applying the boundary condition, which in general is not zero when the convergence is reached because there is an external forcing to satisfy the boundary condition itself. Nevertheless, the right hand side of Equation 2.51 must be cancelled in the convergence, which means that the solution is independent of the time step used.

The turbulent kinetic energy, k , and the specific dissipation rate, ω , need to be specified at the inflow for the $k - \omega$ equations. The turbulent kinetic energy is fixed at the inlet in terms of the turbulent intensity I :

$$k_{inlet} = \frac{3}{2} u_k^2 I^2$$

and the inlet specific dissipation rate is fixed at the inlet in terms of the flow conditions and a factor ω_{fact} which varies between 100 and 500:

$$\omega_{inlet} = \frac{\omega_{fact}}{\beta^*} \sqrt{k}$$

Supersonic Inlet:

When the inlet is supersonic, there are not waves leaving the computational domain. Then, the flow must be defined by external information of the computational domain. Typically, the static pressure is used to complete the set of variables, coupled with the four variables described for a subsonic inlet.

2.6.3 1D Steady Non-Reflecting Outlet Boundary Condition

Subsonic Outlet

Only one external value, that represent one invariant, can be imposed from outside. Typically the static pressure is imposed in subsonic outlets for turbomachinery applications. The two first one-dimensional invariants, which are the entropy and the left-running Riemann invariant:

$$s = \frac{p}{\rho^\gamma} \quad (2.52)$$

$$R^+ = |\vec{v}| + \frac{2a}{\gamma - 1} \quad (2.53)$$

are calculated from the original equations. Additionally, the two remaining invariants are approximated by the two components of the tangential velocity. This condition is frequently impose by retaining from the inner scheme the two angles of the outgoing velocity:

$$\alpha_s = \frac{v_y}{|\vec{v}|} \quad (2.54)$$

$$\alpha_r = \frac{v_z}{|\vec{v}|} \quad (2.55)$$

Similar to the invariant R^- in the subsonic inlet case, the ordinary differential equations to be solved in a subsonic outlet are:

$$\vartheta_{c,i} \frac{\partial s}{\partial t} = \frac{\partial s}{\partial u} R_i \quad (2.56)$$

$$\vartheta_{c,i} \frac{\partial R^+}{\partial t} = \frac{\partial R^+}{\partial u} R_i \quad (2.57)$$

$$\vartheta_{c,i} \frac{\partial \alpha_s}{\partial t} = \frac{\partial \alpha_s}{\partial u} R_i \quad (2.58)$$

$$\vartheta_{c,i} \frac{\partial \alpha_r}{\partial t} = \frac{\partial \alpha_r}{\partial u} R_i \quad (2.59)$$

Supersonic Outlet

When the outlet becomes supersonic, there are not any waves that are coming into the computational domain. Therefore, no additional information is needed from outside.

2.6.4 Solid wall Boundary Conditions

Inviscid wall

The no-penetration boundary condition is imposed in weak form by imposing in the convective fluxes that the normal component of the velocity is null. i.e.: $\mathbf{v} \cdot \mathbf{n} = 0$, yielding:

$$F_c^{slip-wall} = \begin{Bmatrix} 0 \\ p \cdot n_x \\ p \cdot n_y \\ p \cdot n_z \\ 0 \end{Bmatrix} \quad (2.60)$$

Viscous wall

The no-slip wall condition is imposed for a viscous wall, where the velocity of the fluid must be the same as the wall velocity:

$$\vec{v} = \vec{v}_{wall} \quad (2.61)$$

The boundary condition is imposed in strong form, the momentum equation is discarded and replaced by:

$$\rho \mathbf{v} = (\rho \mathbf{v})_{wall} \quad (2.62)$$

The energy equation requires the imposition of one boundary at solid wall. Frequently the wall temperature is imposed:

$$T = T_{wall} \quad (2.63)$$

Similarly to the non-slip velocity condition, this boundary condition is imposed in strong form::

$$\rho E = \rho \left(\frac{T_{wall}}{\gamma - 1} + \frac{\mathbf{v}_{wall}^2}{2} \right) \quad (2.64)$$

and the energy equation is discarded. However when the heat is imposed the boundary condition is imposed in weak form.

For the $k - \omega$ equations, a values for The turbulent kinetic energy, k , and the specific dissipation rate, ω , need to be specified at the wall when the turbulence model is activated.

The turbulent kinetic energy is set to zero at the wall:

$$k_{wall} = 0, \quad (2.65)$$

whereas the wall specific dissipation rate is set to :

$$\omega_{wall} = \frac{6}{0.0708} \frac{\mu_l}{\rho y_w^2} \quad (2.66)$$

where y_w is the distance of the first cell to the wall. Additionally, if $y^+ < 2.5$, the normal derivative of ω is set to zero at the wall:

$$\frac{d\omega}{dy_n} = 0 \quad (2.67)$$

2.7 Explicit Time Integration

To obtain the steady state solution, the spatially discretised equations (2.31) are integrated in time. The code Mu^2s^2T marches in time using an explicit five-stage Runge-Kutta scheme, where the artificial viscosity and the viscous terms are evaluated only in the odd stages [56].

Local time stepping, is used to enhance the convergence acceleration when JST scheme is selected, which guarantees that disturbances will reach the inlet and outlet boundaries in a fixed number of steps proportional to the number of cells between the inner and outer boundaries. Nevertheless, the scalar preconditioner is completely ineffective to solve the convective modes but nearly succeeds in the two acoustic modes. Block-Jacobi preconditioner provides optimal damping for all high frequency modes. Furthermore, in stretched cells of a viscous boundary layer, the block-Jacobi preconditioner also provides adequate solution of convective modes that are high frequency in one mesh direction and low frequency in the other. However, acoustic pressure modes that are high frequency in the streamwise direction and low frequency across the boundary layer are not damped effectively. This effect is particularly important when the multigrid technique is used to improve the convergence rate. Multigrid benefits are based on the ability of the smoother to damp high frequency errors. If this ability is disabled somehow, the whole algorithm fails to converge, since high frequency errors are aliased to low frequency ones when moving to coarser grids, and the smoother does not damp them at all. This causes the

convergence rate to degrade because multigrid can not cope with that high frequency appropriately. This behaviour is avoided using a semi-coarsening multigrid strategy. the grid is not coarsened in the streamwise direction, therefore aliasing in that direction is avoided and the whole scheme recovers one of its fundamental goals, which is not pass high frequency errors to coarser grids.

However, Block-Jacobi preconditioner can not address the problems encountered in the limit of low Mach numbers, i.e. convergence deteriorates as the Mach number diminishes due to a large condition number of the continuous system of equations and accuracy is lost due to a poor scaling of the artificial dissipation terms. Low-Mach number preconditioner modifies the physical eigenvalues, and makes all of them of the order of the flow velocity.

2.7.1 Preconditioning

The time derivative can be scaled to accelerate the convergence to the steady state, without modifying the numerical fluxes, and therefore not altering the final solution. Three global preconditioners are used in the Mu^2s^2T code: local time step, Block-Jacobi matrix, Low-Mach number and multigrid. All those preconditioning techniques are detailed in [40] and here just a brief overview is presented.

2.7.1.1 Local Time Step

Local time step is a scalar preconditioning that adjusts the time step of each cell in the discretised domain according to its stability limit, giving by the spectral radius of the Jacobians of residuals. The parabolic and hyperbolic part are separated to compute the final local time step as:

$$\Delta t_i^{-1} = \Delta t_{H_i}^{-1} + \Delta t_{P_i}^{-1} \quad (2.68)$$

The hyperbolic time step for an edge-based data structure is given by:

$$\Delta t_{H_i}^{-1} = \frac{1}{CFL \cdot \vartheta_i} \left[\sum_{j=1}^{\#ed_i} (|v_{ij} \cdot n_{ij}| + a_{ij}) \sigma_{ij} + \sum_{j=1}^{\#Bed_i} (|v_{ij} \cdot n_{ij}| + a_{ij}) \sigma_{ij} \right] \quad (2.69)$$

Analogously, the parabolic part of the time step is:

$$\Delta t_{P_i}^{-1} = \frac{1}{CFL \cdot \vartheta_i} \left[\sum_{j=1}^{\#ed_i} \frac{\gamma}{\rho_{ij}} \left(\frac{\mu_{ij}}{Pr} \right)_{eq} \frac{4}{|x_j - x_i|} \sigma_{ij} \right] \quad (2.70)$$

The hyperbolic and parabolic parts correspond to the convection and diffusion fluxes respectively. The CFL number is a factor to scale the local time step and speedup the convergence. The limit of the CFL is given by stability criteria.

2.7.1.2 Block-Jacobi Preconditioning

Block-Jacobi preconditioning consists in pre-multiplying the time term derivative by the central node matrix that results from the first-order residual fluxes:

$$\Delta t^{-1} = \frac{1}{CFL \cdot \vartheta_i} \left[\sum_{j=1}^{\#ed_i} \left(\frac{1}{2} |A_{ij}| + B_{ij} \frac{1}{|x_j - x_i|} \right) \sigma_{ij} + \sum_{j=1}^{\#Bed_i} \frac{1}{2} |A_{ij}| \sigma_{ij} \right] \quad (2.71)$$

The matrices $|A_{ij}|$ and B_{ij} are defined in [40].

Block-Jacobi preconditioning scales properly the phase velocity the entropy, vorticity and acoustic waves when the flow is aligned with the grid. As consequence, the stiffness of the high aspect ratio cells is reduced when using block-Jacobi preconditioning instead of local time step. If scalar version of the artificial viscosity is used, the block-Jacobi matrix reduces to the local time step, since $|A_{ij}|$ becomes $(|v_{ij} \cdot n_{ij}| + a_{ij}) \cdot I$.

When $k - \omega$ turbulence model is used, instead to treat the global system as blocks of 7×7 matrices, the additional two turbulent equations are preconditioned separately with their local time step, which is $|v_{ij} \cdot n_{ij}|$.

Additionally, when the viscous terms are negligible and the Mach number is close to 0 or 1, the block-Jacobi matrix becomes singular. To overcome this issue, the eigenvalues are limited by introducing the so-called entropy fix, that consists of setting the minimum allowed eigenvalue of the Roe's Matrix $|A_{ij}|$ as:

$$|\bar{\lambda}_{ij}|^* = \begin{cases} |\bar{\lambda}_{ij}|, & |\bar{\lambda}_{ij}| \geq \epsilon_{ij} \\ \frac{|\bar{\lambda}_{ij}|^2 + \epsilon_{ij}^2}{2\epsilon_{ij}}, & |\bar{\lambda}_{ij}| < \epsilon_{ij} \end{cases} \quad (2.72)$$

being $\epsilon_{ij} = a_{ij}/4$ and $|\bar{\lambda}_{ij}| = \max(|\lambda_i + \lambda_j|/2, K|\lambda_j - \lambda_i|)$, where K is a constant set to

2.

2.7.1.3 Multigrid Preconditioning

Multigriding is widely used to enhance the convergence rate of the Euler and Navier-Stokes equations, which usually undergo a differential convergence of the high and low frequency error modes. High frequency errors are appropriately damped in few iterations. On the other hand, low frequency errors have much lower damping rates. The multigrid idea is to quickly smooth the low frequency errors in a mesh with the cell spacing increased, where those modes correspond to high frequency. The solutions are transferred between fine and coarse grids by using restriction and prolongation operators [69, 1, 63].

The implementation of multigrid method in Mu^2s^2T is detailed in [40]. where the coarsening strategy in unstructured grids is described. Basically control volumes are merged to form an agglomerated volume. The nodes and edges of the fine grid are used to construct a coarser grid. The high-stretched cells near the viscous walls or wakes are agglomerated along the normal direction in order to reduce its aspect ratio and hence alleviating the numerical stiffness associated to it.

The same preconditioning framework used for the explicit method is used in the implicit algorithms developed here, including multigrid. Full approximation scheme (FAS) is implemented to accelerate the convergence of the discrete non-linear problem in the fine grid:

$$R_h(U_h^s) = 0 \quad (2.73)$$

where U_h^s is the exact solution. After performing n iterations in the above equation by any iterative technique, an approximate solution in the fine grid U_h^n is obtained. Since U_h^n does not satisfy Equation 2.73, it is obtained:

$$R_h(U_h^n) = r_h \quad (2.74)$$

where r_h is called the residual, which tends to zero during the convergence process. The objective of multigrid is to provide a correction to the approximate solution such that the exact solution is achieved:

$$U_h^s = U_h^n + \Delta U_h \quad (2.75)$$

Substituting Equation 2.75 into Equation 2.73 and subtracting Equation 2.74, we obtain:

$$R_h (U_h^n + \Delta U_h) - R_h (U_h^n) = -r_h \quad (2.76)$$

The coarse grid equation equivalent to Equation 2.76 can be written as:

$$R_H (I_H^h U_h^n + \Delta U_H) - R_H (I_H^h U_h^n) = -I_H^h r_h \quad (2.77)$$

where I_H^h represents the restriction operator which interpolates the field of the residuals or solution from the fine to the coarse grids. Defining a new coarse grid variable as:

$$U_H = I_H^h U_h^n + \Delta U_H \quad (2.78)$$

the coarse grid equation can be rewritten as:

$$R_H (U_H) = f_H \quad (2.79)$$

where the right hand side has been defined as:

$$f_H = R_H (I_H^h U_h^n) - I_H^h r_h \quad (2.80)$$

Equation 2.79 is a system of equations for the coarse grid which is analogous to that of Equation 2.73 with a new forcing term f_H . Therefore, similar techniques to solve the fine grid can be used to solve the coarse grid. After m iterations performed in the coarse grid to obtain an approximated smooth solution, the solution in the fine grid can be updated as:

$$U_h^{n,m} = U_h^n + I_h^H \Delta U_H^m = U_h^n + I_h^H (U_H^m - I_H^h U_h^n) \quad (2.81)$$

2.7.1.4 Low Mach number preconditioning

In principle, the compressible equations can model flow at all Mach numbers. In the limit of $M \rightarrow 0$ the solution of the compressible Navier-Stokes equations converge to the solution of the incompressible Navier-Stokes equations. But compressible schemes have problems when used in situations where there is a large region of low Mach number. In particular if $M < 0.1$ throughout the flow domain, then two problems are encountered:

- Slow convergence to steady state solution: at small Mach numbers, convective and acoustic eigenvalues are vastly different. The ratio between the maximum

and minimum eigenvalues are: $1 + 1/M$. During computations, the time step is restricted by the largest acoustic speed and this can require a lot of time iterations to converge, which is also referred as stiff problem. Block-Jacobi preconditioning attempts to equalize the eigenvalues and the steady state is reached quickly.

- Inaccurate computation of pressure: asymptotic analysis of the Navier-Stokes equations at low Mach numbers yields that the leading order pressure perturbations are of the order of M^2 , whereas the asymptotic analysis of Finite Volume scheme shows that upwind schemes like MUSCL scheme lead to pressure solution of the form $p(x, t) = p_0(t) + Mp_1(x, t) + \mathcal{O}(M^2)$. The leading order pressure perturbations scale wrongly with the Mach number. This is a spurious solution and leads to large error in pressure. Due to this wrong scaling, upwind schemes add too much dissipation at low Mach numbers.

Low-Mach number preconditioning matrix is chosen such that the eigenvalues of preconditioned system of equations are close to one another. The preconditioning matrix takes a particularly simple form if we express the problem in symmetrised variables (p, u, v, w, s) :

$$P = \begin{bmatrix} \epsilon & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.82)$$

where the parameter ϵ is

$$\epsilon = \min \left(1, 3 \frac{\beta}{a^2} \right) \quad (2.83)$$

and the parameter β is defined as

$$\frac{\beta}{a^2} = \max(M^2, \epsilon_\beta)$$

the limit ϵ_β is set to 0.000125. In Figure 2.6 is depicted the values of the parameter ϵ against the Mach number. Taking $\epsilon \propto M^2$ ensures that the acoustic wave speeds are similar to those of convective waves, which are proportional to flow velocity. When $\epsilon = 1$ the unpreconditioned formulation is recovered, which is the case if the Mach number is

Figure 2.6: Low Mach parameter ϵ variation with the Mach number.

high enough $M > \sqrt{1/3}$.

The eigenvalues of the preconditioned equations are:

$$\begin{aligned}\lambda_{1,2,3} &= v_n \\ \lambda_{4,5} &= \frac{1}{2} (1 + \epsilon) v_n \pm \frac{\tau}{2}\end{aligned}$$

being $\tau = \sqrt{(1 - \epsilon)^2 v_n^2 + 4\epsilon a^2}$.

In the limit of $M \rightarrow 0$, the acoustics eigenvalues are $\lambda_{4,5} = \frac{1 \pm \sqrt{5}}{2} v_n$, being all the eigenvalues of the same order.

Chapter 3

Implicit Algorithms

3.1	Introduction	39
3.2	Approximations of the Jacobian matrix	42

3.1 Introduction

After discretising in space the Reynolds-Averaged Navier-Stokes equations, the resulting system of coupled ordinary differential equations can be written in compact form as:

$$\vartheta_c \frac{dU}{dt} = \mathbf{R}(U) \quad (3.1)$$

where a mass-lumping approach has been followed, and a static mesh has been assumed. ϑ_c is a diagonal matrix containing the volumes each cell.

This system of coupled ordinary differential equations can be solved explicitly with linear multi-step methods, but although the coefficients of this Runge-Kutta schemes can be tailored to yield large CFL number, the convergence to the steady state is usually unacceptable slow. As a consequence either multigrid or implicit methods have to be used to accelerate the convergence to the steady state. In this thesis the latter approach has been pursued.

Without loss of generality the time derivative can be discretised to first order as

$$\frac{dU}{dt} = \frac{U^{n+1} - U^n}{\Delta t}, \quad (3.2)$$

although second order accurate time steppers can be easily implemented in case that time accuracy is sought. Implicit methods evaluate the residual at instant at the next time level, $n + 1$, leading to the solution of the following system of algebraic non-linear equations to obtain, U^{n+1} ,

$$\frac{\vartheta_c}{\Delta t} (U^{n+1} - U^n) = R(U^{n+1}) \quad (3.3)$$

The right hand side is non-linear, and therefore the solution U^{n+1} can not be explicitly solved. One procedure to solve the non-linear algebraic system of equations is by casting the problem as a new time dependent problem adding a pseudo derivative

$$\frac{dU}{d\tau} + \frac{\vartheta_c}{\Delta t} (U - U^n) = R(U) \quad (3.4)$$

and use all the standard acceleration techniques derived for explicit methods, to obtain U^{n+1} . This approach, due to Jameson , is known as the dual time step method (DTS).

Alternatively, the residual can be linearised about the time level n , yielding:

$$\frac{\vartheta_c}{\Delta t} (U^{n+1} - U^n) = R(U^n) + \left(\frac{\partial R}{\partial U} \right)^n (U^{n+1} - U^n) \Rightarrow \left[\frac{\vartheta_c}{\Delta t} - \left(\frac{\partial R}{\partial U} \right)^n \right] \Delta U = R(U^n) \quad (3.5)$$

where $(\partial R / \partial U)^n$ is the Jacobian matrix, and $\Delta U = U^{n+1} - U^n$ the vector of the increments of the conservative variables. This second approach has been followed in this work. Equation 3.5 represents a large nonsymmetric linear system of equations which needs to be solved at each time step. As Δt tends to infinity the method reduces to the Newton's method.

The Jacobian matrix is a large sparse matrix, whose sparsity pattern depends on the stencil of the residual and the mesh topology. In the case of the Navier-Stokes equations, the Jacobian's structure takes the form of a sparse block-matrix, where the non-zero entries correspond to 5x5 blocks.

Due to storage considerations, only lower order representations of the Jacobian are usually considered, $(\partial R / \partial U)^{low}$. The stencil of a first-order discretisation of the convective fluxes involves only the nearest neighbours of each node, i.e., for every edge connecting the nodes i and j , there are two non-zero block sub-matrices in the Jacobian at (i, j) and (j, i) . Moreover, the block diagonal matrix is non-zero, due to the contribution of each node. The number of non-null terms, $N_{non-Null}$, for the first-order Jacobian matrix of the

convective fluxes is finally

$$N_{non-Null} = 25 \times (n_{nodes} + 2n_{edges}) \quad (3.6)$$

where n_{edges} and n_{nodes} are, respectively, the number of edges and nodes of the mesh. In the particular case of 3D semi-unstructured grids, the number of neighbouring edges is 8, and the storage requirement is about three times higher than that of an explicit solver. For tetrahedral meshes where the $n_{edges} \simeq 6n_{nodes}$, this factor can be up to five.

A second-order discretisation of the Jacobian, $(\partial R/\partial U)^{high}$, involves the distance of two neighbours of each node, that for a 3D semi-unstructured mesh are approximately 34, whereas for a tetrahedral mesh is about the double, that is about five times more than for the first order approximation. The memory storage requirements of an implicit method could be about one order of magnitude higher than that of an explicit solver if the Jacobian were actually stored. For this reason, second-order linearisations are not typically used in CFD applications. Nevertheless, they are the base of Newton's method, in which the temporal derivatives are not considered, the Jacobian matrix is calculated in its exact form, and the resulting system of linear equations fully converged. One of the main concerns of the Newton's method is that the initial solution needs to be close enough to the final solution to ensure the convergence, and some starter procedure is usually required. In the proximity of the exact solution, quadratic convergence is obtained, which means that only a few iterations (typically three or four) are enough to reach the solution of the problem.

The construction of a second-order approximation of the Jacobian matrix is not only expensive, as it has been commented before, but make this not block diagonally dominant, and most of the iterative linear solvers either are unstable, or show poor convergence rates. As a result, the cost per iteration of the Newton's method becomes unaffordable making the method not suitable for realistic applications.

3.2 Approximations of the Jacobian matrix

As it has been previously pointed out, the computation and storage of a second-order discretisation of the Jacobian can be unaffordable. Therefore, one the most common technique in the context of implicit methods consists of using a Jacobian based on the linearisation of the first-order discrete operators, in contrast with the full Jacobian methods. It is important to recall that this method, which has some similarities with the so called linearised defect-correction schemes, retains however the second-order spatial accuracy, and only affects to the convergence rate. This can be realised by noting that the resulting scheme

$$\left[\frac{\vartheta_c}{\Delta t} - \left(\frac{\partial R}{\partial U} \right)_{low}^n \right] \Delta U = R_{high}(U^n) \quad (3.7)$$

ensures that $R_{high}(U) = 0$, when $\Delta U \rightarrow 0$.

This scheme can be seen as a preconditioned technique, where the preconditioned matrix is given by $\left[\frac{\vartheta_c}{\Delta t} - \left(\frac{\partial R}{\partial U} \right)_{low}^n \right]$.

The advantages of approximating the first-order Jacobian to the resolution of the Navier-Stokes equations by means of linear implicit methods are the low storage requirements, the diagonal dominance of the resulting linear system, and the direct correspondence between the Jacobian matrix and the grid graphs, which in turn enables the construction of the Jacobian, or the matrix-vector product using an edge-based data structure.

Linear implicit methods can then be classified according to the approximation of the Jacobian matrix as exact implicit methods or implicit methods based on the first-order Jacobian. Jacobian matrices can in turn be computed making use of its exact form, or using some additional approximations, such as replacing the Roe's matrix by its spectral radius in the well-known Roe's Flux function [79].

In this work only linear implicit methods based on the first-order Jacobian, J_{1st} , are studied. Matrix-vector products are directly computed using an edge-based data structure

as:

$$\begin{aligned} J_{1st} \Delta U &= -\frac{1}{2} \sum_{j=1}^{n_{edges}} \left[\left(\frac{\partial F_c(U_i)}{\partial U_i} \Delta U_i + \frac{\partial F_c(U_j)}{\partial U_j} \Delta U_j \right) n_{ij} + |A_{ij}(U_i, U_j) n_{ij}| (\Delta U_i - \Delta U_j) \right] \sigma_{ij} \\ &\quad - \sum_{j=1}^{\#Edges_i} \frac{\partial F_c(U_i)}{\partial U_i} \Delta U_i n_i \sigma_i \end{aligned} \quad (3.8)$$

The 5x5 block diagonal of the linear system matrix is:

$$D_i = \frac{\vartheta_c}{\Delta t} I_d + \frac{1}{2} \sum_{j=1}^{n_{edges}} \left[\frac{\partial F_c(U_i)}{\partial U_i} n_{ij} + |A_{ij}(U_i, U_j) n_{ij}| \right] \sigma_{ij} - \sum_{j=1}^{\#Edges_i} \frac{\partial F_c(U_i)}{\partial U_i} n_i \sigma_i \quad (3.9)$$

It is worth noting that for the inner points of the domain, only the temporal terms and the numerical diffusion terms contribute to the block diagonal matrix. The non-zero off-diagonal entries of the linear system matrix are given by the edge that connects the nodes “i” and “j”. These contributions can be split by the lower and upper triangular matrices. Assuming that the node “i” has a lower index than the node “j”, the elements of the lower triangular L and upper triangular U matrices are:

$$U_{ij} = \frac{1}{2} \left[\frac{\partial F_c(U_j)}{\partial U_j} n_{ij} - |A_{ij}(U_i, U_j) n_{ij}| \right] \sigma_{ij} \quad (3.10)$$

$$L_{ji} = \frac{1}{2} \left[-\frac{\partial F_c(U_i)}{\partial U_i} n_{ij} - |A_{ij}(U_i, U_j) n_{ij}| \right] \sigma_{ij} \quad (3.11)$$

where it has been taken into account that the external normal vectors of the common face centered in the nodes “i” and “j” have opposite sign:

$$n_{ji} = -n_{ij}$$

In [60], the Roe’s matrix is replaced by its spectral radius, λ_{ij} , the block matrices can be simplified as:

$$|A_{ij}(U_i, U_j) n_{ij}| \rightarrow \left(|\vec{v}_{ij} \cdot \vec{n}_{ij}| + a_{ij} + \frac{\gamma}{\rho |x_j - x_i|} \left(\frac{\mu_l}{Pr_l} + \frac{\mu_t}{Pr_t} \right) \frac{M_\infty}{Re_\infty} \right) I_d \quad (3.12)$$

where I_d is the identity matrix. The advantage of this approximate Jacobian matrix is the low computational cost required to build the matrix. Although the number of time steps may increase, the reduction of the cost per iteration usually compensates this penalty.

Chapter 4

Linear Solvers

4.1	Introduction	45
4.2	Iterative Solvers	46
4.3	Block Linear Solvers	52
4.4	Implementation of the Linear Solvers	53

4.1 Introduction

The efficient implicit time integration of the discretised RANS equations leads to the linearisation of the governing equation system, and the resolution of a system of linear equations. There is a wide spectrum of algorithms for solving a system of linear equations, but only few of them have shown to be efficient for CFD applications.

There are a number of requirements that an algorithm for the solution of a linear system of equations must fulfill to be considered for its use in a CFD solver. The most important are related to the memory usage, which needs to be as little as possible in order that the implicit solver remain competitive with respect explicit solvers, and its efficiency. Efficient linear solvers for large systems of equations employ iterative algorithms, this is essential not only to reduce the amount of required memory, but the number of iterations needed to reach the converged state. Furthermore, the linear solver must present a good scalability in many-core processors.

A linear system of equations can be written as:

$$Ax = b \tag{4.1}$$

where A is the coefficient matrix of size $N \times N$, the right-hand side vector b is the independent term, and x is the unknown vector. As it has been described in Chapter 3, this is a sparse linear system, where the matrix A is a non-symmetric matrix. Thus, algorithms which are based on certain properties of the matrix can not be applied here. This is the case of the Conjugate Gradient method, which is one of the best iterative methods for solving sparse Symmetric Positive Definite linear systems, usually stemming from elliptic problems.

Algorithms for the resolution of linear systems can be classified into two main types: direct and iterative methods. Direct methods attempt to solve the problem by a finite sequence of operations, whereas iterative methods converge for a certain number of iterations which depends on certain properties of the matrix and the algorithm itself. Moreover, convergence is not always guaranteed for iterative methods, and special attention has to be paid to the design of this algorithms. On the other hand, although direct methods always provide a solution of the system, for large systems can become prohibitively expensive. since the number of operations required to solve the problem is $O(N^3)$, where N is the number of unknowns. For this reason, direct methods are seldom considered for CFD applications.

The most relevant iterative solvers are presented in the following lines, with special emphasis on those exhibiting high generality, robustness and efficiency [82, 96, 60].

4.2 Iterative Solvers

Iterative solvers are based on the relaxation of the problem from a given initial solution. Each iteration reduces the error of the components of the unknown vector until the convergence is reached.

Two groups of iterative solvers have been considered in this work: splitting and Krylov subspace methods. The former decomposes the coefficient matrix into two matrices. The

first matrix multiplies the unknown vector at the current step, while the second operates over the vector of unknowns of the next iteration, resulting in a simpler linear system which can be easily solved by a direct method. Krylov methods project the solution over a prescribed number of vectors of the Krylov subspace and optimizes the residual.

4.2.1 Splitting Methods

These methods consist of decomposing the coefficient matrix into two parts such as:

$$A = M + N \quad (4.2)$$

The matrix M is taken to be the coefficient matrix of the next iteration, $k + 1$, whereas matrix N is applied to the solution of the previous iteration, k , leading to:

$$Mx^{k+1} + Nx^k = b \quad (4.3)$$

whose solution is:

$$x^{k+1} = M^{-1} (b - Nx^k) \quad (4.4)$$

The matrix M must be invertible, that is, $\det(M) \neq 0$. This iterative algorithm is stable if the approximate solution x^k tends to the solution of the system:

$$x_s = A^{-1}b \quad (4.5)$$

Defining the error as the difference between the approximate solution and the exact solution, $\epsilon^k = x^k - x_s$, from Equation 4.4, the expression of the error can be derived:

$$x^{k+1} = M^{-1} ((M + N)x_s - Nx^k) = x_s - M^{-1}N(x^k - x_s) \Rightarrow \epsilon^{k+1} = -M^{-1}N\epsilon^k \quad (4.6)$$

The resulting amplification matrix is $G = -M^{-1}N$. After k iterations, the error is:

$$\epsilon^k = G^k \epsilon^0 \quad (4.7)$$

and then, the method is stable if the spectral radius of the matrix G is less than 1:

$$\rho(G) < 1,$$

where the spectral radius, $\rho(G)$, is defined as the maximum modulus of the eigenvalues of matrix G , and defines the convergence rate of the scheme.

Depending on the selection of the matrices M and N , different methods are obtained:

Jacobi method:

This method is the simplest algorithm to solve a linear system using an iterative method. The matrix M is decomposed in a diagonal matrix, denoted as D , and an off-diagonal matrix, N , , which can be expressed as $N = L + U$, that is, the sum of the strict lower-triangular matrix L , and the strict upper-triangular matrix U . The amplification matrix for the Jacobi method becomes:

$$G_J = -D^{-1} (L + U) \quad (4.8)$$

Jacobi method has been applied to the resolution of elliptic PDEs problems. The spectral radius corresponding to a two-dimensional mesh of $n \times n$ points of the Poisson equation for large n is:

$$\rho_J (G) \simeq 1 - \frac{\pi^2}{2n^2} \quad (4.9)$$

The number of iterations to reduce the overall error by a factor of 10^{-p} is $\frac{1}{2}pn^2$. The spectral radius approaches unity as the grid size increases, and the convergence rate decreases quadratically with the element mesh size.

Forward Gauss-Seidel:

In this case, the matrix M is formed by the diagonal matrix, D , plus the strict lower-triangular matrix L , whereas the matrix N is the strict upper-triangular matrix U . The amplification matrix for the forward Gauss-Seidel is:

$$G_{FGS} = -(D + L)^{-1}U \quad (4.10)$$

Backward Gauss-Seidel:

Here the matrix M is formed by the diagonal matrix D plus the strict upper-triangular matrix U , whereas the matrix N is the strict lower-triangular matrix L . The amplification matrix for the forward Gauss-Seidel is:

$$G_{BGS} = -(D + U)^{-1}L \quad (4.11)$$

Following the particular case of elliptic PDEs problems, the spectral radius corresponding to a two-dimensional mesh of $n \times n$ points of the Poisson equation is:

$$\rho_{GS}(G) \simeq 1 - \frac{\pi^2}{n^2} \quad (4.12)$$

which is lower than the spectral radius obtained for the Jacobi method (Equation 4.9).

In general the larger the diagonal matrix is, compared with the off-diagonal matrix, the smaller the amplification matrix is for the Jacobi and Gauss-Seidel methods. The convergence of the Jacobi and Gauss-Seidel methods is guaranteed if the coefficient matrix A is strictly diagonally dominant or an irreducibly diagonally dominant matrix [82].

A matrix is said to be strictly diagonally dominant if:

$$|a_{jj}| > \sum_{\substack{i=1 \\ i \neq j}}^n |a_{ij}| \quad j = 1, \dots, n \quad (4.13)$$

A matrix, A , is said to be irreducibly diagonally dominant if A is irreducible, and:

$$|a_{jj}| \geq \sum_{\substack{i=1 \\ i \neq j}}^n |a_{ij}| \quad j = 1, \dots, n \quad (4.14)$$

with strict inequality for at least one j

It can be anticipated that for the linearised RANS equations, lower CFL numbers lead to a better conditioned linear system. Indeed, the time step only contributes to the diagonal matrix of the system, and therefore, reducing the CFL number, the matrix is more diagonally dominant.

Successive Over Relaxation Successive Over Relaxation (SOR) methods consists of correcting the residual error by a relaxation factor ω and setting $M = D + U$. The solution of the system 4.4 can be rearranged as:

$$x^{k+1} = x^k + M^{-1} (b - Ax^k) \quad (4.15)$$

and adding the over relaxation parameter yields:

$$x^{k+1} = x^k + \omega M^{-1} (b - Ax^k) \quad (4.16)$$

The method is stable for the range $0 < \omega < 2$. The optimum parameter ω is obtained empirically and depends on the PDE problem and the grid resolution. For the Laplace equation $\omega_{opt} \simeq 1.8$.

Symmetric Gauss-Seidel (SGS): This method consists of performing a forward sweep followed by a backward sweep of the Gauss-Seidel algorithm. The matrices M and N are:

$$M_{SGS} = (D + L) D^{-1} (D + U) \quad (4.17)$$

$$N_{SGS} = -LD^{-1}U \quad (4.18)$$

and finally, the amplification matrix is:

$$G_{SGS} = (D + U)^{-1} L (D + L)^{-1} U \quad (4.19)$$

Lower-Upper Symmetric Gauss-Seidel: Lower-upper Symmetric Gauss-Seidel (LU-SGS) method was introduced for the resolution of the RANS in structured grids by Jameson and Yoon [99] and extended to unstructured grids by [87]. This method consists of neglecting the matrix N_{SGS} of Equation 4.18, yielding:

$$(D + L) D^{-1} (D + U) \cdot x = b \quad (4.20)$$

which can be solved in two sweeps as:

$$(D + L) x^* = b$$

$$(D + U) x = D \cdot x^*$$

The linear system is approximately factored, since the matrix A is approximated by the product of a lower $(D + L)$ and an upper $(I + D^{-1}U)$ matrices, with an approximation error $-LD^{-1}U$. LU-SGS method can be used as a good smoother for multi-grid solvers or preconditioner for other iterative methods such as Krylov methods.

The solution of this system is exactly the same result as one iteration of the SGS method

with the initial guess set to 0, but this approach is computationally more efficient since there is an extra triangular matrix multiplication for SGS method.

4.2.2 Krylov Methods

Krylov methods belong to the family of projection methods, whose main idea is to extract an approximate solution, x_m , from an affine subspace $x_0 + \mathcal{K}_m$ of $\mathbb{R}^n$, with dimension m , and such that:

$$b - Ax_m \perp \mathcal{L}_m \quad (4.21)$$

where $\mathcal{L}_m$ is another subspace of dimension m , and x_0 represents an initial guess of the solution. The Krylov methods are those for which the subspace $\mathcal{K}_m$ is the Krylov subspace:

$$\mathcal{K}_m(A, r_0) = \text{span} \{r_0, Ar_0, A^2r_0, \dots, A^{m-1}r_0\} \quad (4.22)$$

being $r_0 = b - Ax_0$ the initial residual vector.

Different choices of the subspace $\mathcal{L}_m$ lead to different methods of Krylov subspace methods. For the sake of brevity, $\mathcal{K}_m(A, r_0)$ will be denoted by $\mathcal{K}_m$. The most popular method used in the CFD community due to the efficiency is the GMRES (Generalized Minimum Residual), which is based on taking the subspace $\mathcal{L}_m = A\mathcal{K}_m$.

4.2.2.1 GMRES

GMRES is a projection method to solve linear systems 4.1, which belongs to the family of Krylov subspace methods. This is a generalization of the conjugate gradient method for solving a linear system where the coefficient matrix is not symmetric and/or positive definite. The idea of the algorithm is to minimise the residual over the Krylov subspace, i.e., seeks a vector $x_0 + V_m y$ that minimises the residual, being V_m a matrix formed with the Arnoldi vectors as columns.

To generate the orthogonal basis of the Krylov subspace there are several versions, such as the Householder Arnoldi or Arnoldi-Modified Gram-Schmidt. The latter is used in this work since it is more reliable in presence of round-off errors compared with the standard Arnoldi's procedure [82]. A deep description of GMRES algorithm is detailed in B.

4.2.2.2 Preconditioned GMRES

GMRES solvers usually need many iterations to converge large linear systems, and frequently tend to stagnate when the dimension of the Krylov subspace is much lower than the dimension of the system. To overcome this weakness, GMRES is used together with other solvers which act as preconditioners. The idea of the preconditioner is to pre-multiply the original system by a precondition matrix, P , that approximates the inverse of the matrix of the system, $P \simeq A^{-1}$, and that at the same time PA is inexpensive to evaluate, and P is a sparse matrix with low memory requirements:

$$PAx = Pb \quad (4.23)$$

Obviously, the closer the matrix P to A^{-1} is, the better the preconditioning is.

Splitting methods preconditioning:

The splitting methods presented earlier can be used as GMRES preconditioners. Indeed, from Equation 4.7, it is derived that:

$$x^k - x^0 = (I - G^k) A^{-1} (b - A \cdot x^0) \quad (4.24)$$

Therefore, performing k iterations of a splitting method, is equivalent to precondition the original system by the matrix $P = (I - G^k) A^{-1}$. If the linear system is converged completely, the preconditioning matrix derives into the inverse of the matrix A .

LU-SGS has been used as an efficient preconditioner for GMRES. Löhner proposed preconditioning GMRES with this approach improving the best existing results using implicit methods by that time [60].

4.3 Block Linear Solvers

In the particular case of CFD, every node contributes with 5 or 7 of unknowns when the RANS equations are solved with the algebraic or $k - \omega$ turbulence models respectively. Therefore, the derived linear system consists of block matrices of size 5×5 or 7×7 depending on the turbulence model. However, the linearised $k - \omega$ turbulence model

Figure 4.1: Illustration of an edge contribution to the Jacobian matrix.

is solved uncoupled from the Navier-Stokes equations. Therefore, taking into account the structure of the coefficient matrix of the linear system to be solved, the algorithms described in the previous sections can be modified by its block versions.

4.4 Implementation of the Linear Solvers

4.4.1 Introduction

The implicit methods implemented in this work are based on the first-order Jacobian matrix, that is, the linearisation is done using an approximate residual with first-order artificial viscosity. In this case only the nearest neighbours of each node contribute to the Jacobian reducing the number of non-zero entries of the matrix of the system. Indeed, the edge that connects the nodes i and j contributes to the elements a_{ij} and a_{ji} of the matrix. This is sketched in Figure 4.1, where the contributions of the edges of a dual cell are represented.

The algorithms implemented in the code are described below. They are based on the product between the entire Jacobian matrix or some block of it, and the increment of the

Algorithm 4.1 Jacobi algorithm with and edge-based data structure.

1. *Initial guess* : $\Delta U = 0$
 2. *for* $k = 0, \dots, \text{linear iterations}$
 3. *Compute* $O = (L + U) \cdot \Delta U$:
 4. *for* $j = 0, \dots, \#edges$
 5. $O_{node1} = O_{node1} - \frac{1}{2} \left[\frac{\partial F_c(U_{node2}) n_{edgej}}{\partial U_{node2}} - |A_{edgej}| \right] \cdot \Delta U_{node2}^k \cdot \sigma_{edgej}$
 6. $O_{node2} = O_{node2} - \frac{1}{2} \left[-\frac{\partial F_c(U_{node1}) n_{edgej}}{\partial U_{node1}} - |A_{edgej}| \right] \cdot \Delta U_{node1}^k \cdot \sigma_{edgej}$
 7. *end*
 8. *for* $i = 0, \dots, \#nodes$
 9. $\Delta U_i = D_i^{-1} [R_i^n - O_i]$
 10. *end*
 11. *end*
-

conservative variables, ΔU . Since the Jacobian is not stored, these products are performed by means of an edge loop similar to that used for the computation of the convective fluxes.

4.4.2 Jacobi solver

The Jacobi solver iteration applied to the edge-based data structured is:

$$\Delta U_i^{k+1} = D_i^{-1} \left\{ R_i^n + \frac{1}{2} \sum_{j=1}^{\#edges_i} \left[\frac{\partial F_c(U_j) n_{ij}}{\partial U_j} - |A_{ij}(U_i, U_j) n_{ij}| \right] \cdot \Delta U_j^k \cdot \sigma_{ij} \right\} \quad (4.25)$$

where the nodal diagonal block-matrix D_i is defined in 3.9.

The off-diagonal terms of the vector of unknowns is computed using a standard edge-based loop, but neglecting the contribution of the node itself. This is detailed in Algorithm 4.1. The inverse of the block-diagonal matrix is pre-computed and stored for every node, whereas the matrix $|A_{edgej}|_{5 \times 5}$ is stored for every edge. There are two main loops for this algorithm: the first one consists of computing the off-diagonal matrix times the unknown vector in a loop over edges, and the second one updates the unknown vector in a loop over nodes. The simplicity of the algorithm allows to parallelise the solver efficiently, obtaining good scalability results, as it will be shown.

4.4.3 LU-SGS solver

The LU-SGS solver iteration applied to the edge-based data structured is:

Algorithm 4.2 Forward sweep for LU-SGS algorithm with and edge-based data structure.

1. Solve $(D + L) \cdot \Delta U^{1/2} = R_i^n$:
 2. for $i = 0, \dots, \#nodes$
 3. for $j = 0, \dots, \#edges[i] : node_2 < node_1, node_1 = i$
 4. $L_i = L_i - \frac{1}{2} \left[\frac{\partial F_c(U_{node2}) n_{edgej}}{\partial U_{node2}} - |A_{edgej}| \right] \cdot \Delta U_{node2}^{1/2} \cdot \sigma_{edgej}$
 5. end
 6. $\Delta U_i^{1/2} = D_i^{-1} [R_i^n - L_i]$
 7. end
-

Algorithm 4.3 Backward sweep for LU-SGS algorithm with and edge-based data structure.

1. Solve $(D + U) \cdot \Delta U = D \cdot \Delta U_i^{1/2}$:
 2. for $i = \#nodes, \dots, 0$
 3. for $j = 0, \dots, \#edges[i] : node_2 > node_1, node_1 = i$
 4. $U_i = U_i - \frac{1}{2} \left[\frac{\partial F_c(U_{node2}) n_{edgej}}{\partial U_{node2}} - |A_{edgej}| \right] \cdot \Delta U_{node2} \cdot \sigma_{edgej}$
 5. end
 6. $\Delta U_i = \Delta U_i^{1/2} - D_i^{-1} U_i$
 7. end
-

- Forward sweep:

$$\Delta U_i^{k+1/2} = D_i^{-1} \left\{ R_i^n + \frac{1}{2} \sum_{j=1:j < i}^{\#edges_i} \left[\frac{\partial F_c(U_j) n_{ij}}{\partial U_j} - |A_{ij}(U_i, U_j) n_{ij}| \right] \cdot \Delta U_j^{k+1/2} \cdot \sigma_{ij} \right\} \quad (4.26)$$

Backward sweep:

$$\Delta U_i^{k+1} = \Delta U_i^{k+1/2} + D_i^{-1} \left\{ \frac{1}{2} \sum_{j=1:j > i}^{\#edges_i} \left[\frac{\partial F_c(U_j) n_{ij}}{\partial U_j} - |A_{ij}(U_i, U_j) n_{ij}| \right] \cdot \Delta U_j^{k+1} \cdot \sigma_{ij} \right\} \quad (4.27)$$

where the nodal diagonal block-matrix D_i is defined in 3.9.

In this case, instead of performing a loop over edges, like in the computation of the fluxes and the off-diagonal times of the Jacobi method, a loop over nodes is done. Moreover, for the forward seep, an additional loop over edges is performed, but considering only those edges connecting nodes with lower index than the node i (see Algorithm 4.2). In this way the lower triangular matrix is solved without storing the elements of the matrix. Analogously, the backward seep algorithm is showed in Algorithm 4.3, but taking into account only the nodes with indexes higher than the node i , in order to solve the upper triangular matrix.

4.4.4 GMRES solver

Summarising the GMRES method, the algorithm is showed in 4.4.

4.4.5 Preconditioned GMRES solver

The modifications of the Algorithm 4.4 due to the preconditioning are:

- Line 2: $r^0 = P(b - A \cdot x^0)$
- Line 7: $v_{j+1} = P \cdot A \cdot v_j$

Line 2 is obtained directly by running a certain number of iterations of the splitting method, as described in Equation 4.24.

Line 7 is obtained just using the same routines as Line 2, but setting the vectors $x^0 = 0$ and $b = A \cdot v_j$.

Algorithm 4.4 GMRES algorithm with practical implementation.

1. *Initial guess* : x_0
 2. $r_0 = b - Ax_0$
 3. $\gamma_1 = \|r_0\|$
 4. $v_1 = \frac{r_0}{\gamma_1}$
 5. $j = 1$
 6. *while* ($j \leq m$ and $\gamma_j > tol$)
 7. $v_{j+1} = Av_j$
 8. *for* $i = 1, \dots, j$
 9. $h_{i,j} = \langle v_{j+1}, v_i \rangle$
 10. $v_{j+1} = v_{j+1} - h_{i,j}v_i$
 11. *end for*
 12. $h_{j+1,j} = \|v_{j+1}\|$
 13. $v_{j+1} = \frac{v_{j+1}}{h_{j+1,j}}$
 14. *Givens rotation* :
 15. *for* $i = 1, \dots, j - 1$
 16. $h_{i,j}^* = c_i h_{i,j} + s_i h_{i+1,j}$
 17. $h_{i+1,j} = -s_i h_{i,j} + c_i h_{i+1,j}$
 18. $h_{i,j} = h_{i,j}^*$
 19. *end for*
 20. $\mu = \sqrt{h_{j,j}^2 + h_{j+1,j}^2}$
 21. $c_j = h_{j,j}/\mu$
 22. $s_j = h_{j+1,j}/\mu$
 23. $h_{j,j} = c_j h_{j,j} + s_j h_{j+1,j}$
 24. $h_{j+1,j} = 0$
 25. $\gamma_j^* = c_j \gamma_j + s_j \gamma_{j+1}$
 26. $\gamma_{j+1} = -s_j \gamma_j + c_j \gamma_{j+1}$
 27. $\gamma_j = \gamma_j^*$
 28. *End Givens rotation*
 29. $j = j + 1$
 30. *end while*
 31. *Solve the least square problem* :
 32. $j = j - 1$
 33. $y_j = \gamma_j / h_{j,j}$
 34. *for* $i = j - 1, \dots, j = 1$
 35. $y_i = \gamma_i$
 36. *for* $k = i + 1, \dots, j$
 37. $y_i = y_i - h_{i,k} y_k$
 38. *end for*
 39. $y_i = y_i / h_{i,i}$
 40. *Update the solution*
 41. $x = x_0 + V_j y_j$
-

Chapter 5

Linear stability and convergence analysis

5.1	Introduction	59
5.2	Temporal Discretisation	62
5.3	Stability of the Linear Advection Equation	65
5.4	Stability of the Euler equations	77

5.1 Introduction

The stability analysis described here, known as Von Neumann analysis, is based on a Fourier decomposition of the linearised equations. This technique consists in studying the development of small perturbations about a uniform solution. To tackle this analysis, the flow variables, U , are decomposed into the exact solution, U_e , and a small perturbation, u , such that $U = U_e + u$, with $u \ll U_e$. Injecting this expansion in the spatially discretised governing equations:

$$\vartheta \frac{\partial (U_e + u)}{\partial t} = R(U_e + u) \quad (5.1)$$

and neglecting higher than first order terms, the first order linearised equations for the perturbations are obtained:

$$\vartheta \left(\frac{\partial u}{\partial t} + \frac{\partial U_e}{\partial t} \right) = R(U_e) + \left(\frac{\partial R}{\partial U} \right)_e u \quad (5.2)$$

Taking into account that U_e satisfies the discrete steady RANS equations, then $\partial U_e / \partial t = 0$ and $R(U_e) = 0$. Defining the Jacobian matrix as $J = \partial R(U_e) / \partial U$, the equations for the perturbations reduce finally to:

$$\vartheta \frac{\partial u}{\partial t} = Ju \quad (5.3)$$

Assuming a two-dimensional equispaced structured mesh, taking into account that Equation 5.3 is linear, and assuming periodic boundary conditions, we can expand the solution in Fourier series in space

$$u(x, y, t) = \sum_{m=0}^{N_x/2} \sum_{n=0}^{N_y/2} \hat{u}_{mn}(t) e^{i(mx+ny)} \quad (5.4)$$

Replacing 5.4 in Equation 5.3 we obtain:

$$\sum_{m=0}^{N_x/2} \sum_{n=0}^{N_y/2} \left[\vartheta \frac{d\hat{u}_{mn}}{dt} - Z_{mn} \hat{u}_{mn} \right] e^{i(mx_i+ny_j)} = 0 \quad (5.5)$$

The governing equations are discretised at the mesh points, $x_i = i\Delta x$ and $y_j = j\Delta y$, with $i = 1, \dots, N_x$ and $j = 1, \dots, N_y$. Z_{mn} is the discrete spatial operator in the Fourier space, which in general is a complex number. For the sake of clarity, from now on, the sub-indexes mn will be omitted, i.e., for instance, the Z_{mn} operator is renamed as Z . This operator is formed by the contributions of the convective, artificial viscosity, viscous and source terms:

$$Z = Z_c + Z_{av} + Z_{vis} + Z_P \quad (5.6)$$

In this chapter two physical models are studied in order to understand and separate the effects of the non-dimensional parameters of the problem in the stability of the discretised RANS equations.

1. The linear advection equation, which is useful to understand the effects of the cell aspect ratio and the time step.
2. The linearised Euler equations. This model is useful to assess compressibility effects.

The spatial eigenvalues resulting from the discretisation of the convective terms using a centered scheme, and first-order and third-order artificial dissipation terms (JST scheme) formulated using an edge-based data structure are:

$$\begin{aligned} Z_c^{JST} &= -i\Delta x [\Lambda A_x \sin(m\Delta x) + A_y \sin(n\Delta y)] \\ Z_{av}^{JST} &= \Delta x [\Lambda |A_x| (\cos(m\Delta x) - 1) + |A_y| (\cos(n\Delta y) - 1)] \cdot \\ &\quad \cdot \left[S - \frac{(1-\kappa)(1-S)}{4} (\cos(m\Delta x) + \cos(n\Delta y) - 2) \right] \end{aligned} \quad (5.7)$$

and for MUSCL scheme:

$$\begin{aligned} Z_c^{MUSCL} &= -i\Delta x \left\{ \Lambda A_x \left[\sin(m\Delta x) + \frac{S(1-\kappa S)}{4} (2\sin(m\Delta x) - \sin(2m\Delta x)) \right] + \right. \\ &\quad \left. + A_y \left[\sin(n\Delta y) + \frac{S(1-\kappa S)}{4} (2\sin(n\Delta y) - \sin(2n\Delta y)) \right] \right\} \end{aligned} \quad (5.8)$$

$$\begin{aligned} Z_{av}^{MUSCL} &= \Delta x \left\{ \Lambda A_x \left[(1 - \kappa S^2) (\cos(m\Delta x) - 1) - \frac{S(1-\kappa S)}{4} (\cos(2m\Delta x) - 1) \right] + \right. \\ &\quad \left. + A_y \left[(1 - \kappa S^2) (\cos(n\Delta y) - 1) - \frac{S(1-\kappa S)}{4} (\cos(2n\Delta y) - 1) \right] \right\} \end{aligned} \quad (5.9)$$

where Λ is the cell aspect ratio, $\Lambda = \Delta y / \Delta x$, k a constant fixed to $1/3$, and S a switcher between the first and second order discretisation. For this study, second-order solutions are considered, i.e. the switch parameter S is equal to 0 for JST scheme whereas is set to 1 for MUSCL scheme. The Jacobian matrices A_x , A_y , $|A_x|$ and $|A_y|$ are different for every model.

Equation 5.5 is fulfilled when the coefficients within the square brackets are cancelled, that is:

$$\vartheta \frac{d\hat{u}}{dt} = Z\hat{u}. \quad (5.10)$$

Equation 5.10 represents the temporal evolution of the (m, n) harmonic. This equation can be preconditioned by the matrix P^{-1} , without affecting the steady state. P^{-1} can represent either local time step or block-Jacobi preconditioning. Equation 5.10 then becomes:

$$P^{-1} \vartheta \frac{d\hat{u}}{dt} = Z\hat{u} \quad (5.11)$$

The matrix P^{-1} is typically derived from the first-order Block-diagonal matrix of the Jacobi iteration:

$$\vartheta \frac{P^{-1}}{\Delta t} = \sum_{j=1}^{\#ed_i} |A_{ij}| \sigma_{ij} = \frac{2\Delta x}{CFL} (|A_x| \Lambda + |A_y|) \quad (5.12)$$

where CFL is the Courant-Friedrichs-Lewy number, which scales the time step.

5.2 Temporal Discretisation

Equation 5.11 needs to be integrated in time. Implicit integration, which is the subject of this work, is compared against its explicit counterpart. Moreover, implicit integration can be based on either the first-order or the full Jacobian. Additionally, the implicit integration needs the solution of a large sparse linear system, which is performed by means of iterative methods. In particular, the Jacobi linear solver will be studied and compared with the exact matrix inversion. The existence of various options, both for the Jacobian approximation and the solution of the linear system, leads to a number of combinations which will be presented in the following subsections.

Numerical time integration gives rise to the amplification matrix concept:

$$\hat{u}^{t+\Delta t} = G \hat{u}^t \quad (5.13)$$

which defines the relationship between two consecutive time steps. The numerical scheme is said stable when the eigenvalues of the amplification matrix have a modulus lower than one. Therefore, the eigenvalue footprints are represented in the complex plane to assess the damping of every mode. Moreover, high and low frequency modes are represented separately in order to identify which errors need more iterations to converge and which errors are properly damped. Following this approach, an assessment on the efficiency of a multigrid method can also be done. The nomenclature for the classification of the modes as a function of the spatial wave numbers is given in Table 5.1, where H_ξ and L_ξ represent the high and low frequency modes along the ξ direction respectively.

	$m\Delta x$	$n\Delta y$
HxHy	$[\pi/2, 3\pi/2]$	$[\pi/2, 3\pi/2]$
HxLy	$[\pi/2, 3\pi/2]$	$[-\pi/2, \pi/2]$
LxHy	$[-\pi/2, \pi/2]$	$[\pi/2, 3\pi/2]$
LxLy	$[-\pi/2, \pi/2]$	$[-\pi/2, \pi/2]$

Table 5.1: Nomenclature for mode classification as a function of the wave numbers.

5.2.1 Explicit Integration

The family of Runge-Kutta schemes is selected as representative class of explicit time marching methods. Integrating Equation 5.11 by a n th-order Runge-Kutta method the following amplification factor is obtained:

$$G^{RK-nth} = \left(I + \sum_{k=1}^{n_{stage}} \frac{(\tilde{Z})^k}{k!} \right) \quad (5.14)$$

where $\tilde{Z} = \Delta t PZ/\vartheta$ and n_{stage} the number of stages of the RK. In this chapter, an efficient 5-stage Runge-Kutta scheme is studied which is representative of the type of RK schemes that can be found in RANS codes designed with acceleration mechanisms to obtain the steady state.

5.2.2 Implicit Integration based on the full Jacobian

Implicit Euler integration has been used throughout this work to solve Equation 5.11. When the linear system is solved using direct methods, that is, the matrix of the system is inverted, the amplification matrix is of the form:

$$G^{exact, Full-J} = \left(I - \tilde{Z} \right)^{-1}, \quad (5.15)$$

but when an iterative method is used to solve the linear system, the amplification matrix depends on the number of iterations. In particular, for the Jacobi method, the matrix of the system $I - \tilde{Z}$ is split into the diagonal, D , and the off-diagonal, O , terms. The Jacobi iteration can then be expressed as:

$$(\hat{u}^{n+1})^{j+1} = D^{-1} \left(\hat{u}^n - O(\hat{u}^{n+1})^j \right) \quad (5.16)$$

with $(\hat{u}^{n+1})^0 = \hat{u}^n$. The amplification matrix for this iterative process, is:

$$G^{jacob_i, Full-J} = M^j \quad (5.17)$$

where the matrix M^j is obtained iteratively after j iterations:

$$\begin{aligned} M^{j+1} &= D^{-1} (I - OM^j) \\ M^0 &= I \end{aligned} \quad (5.18)$$

5.2.3 Implicit Integration based on the first-order Jacobian

Similar to the previous case, an implicit Euler integration is applied to solve Equation 5.11. Instead of inverting the full second-order Jacobian matrix, the residual is linearised in time using a simplified first-order Jacobian matrix,

$$R(U^{n+1}) = R(U^n) + \left(\frac{\partial R}{\partial U} \right)_{1st}^n (U^{n+1} - U^n) \quad (5.19)$$

then, linearising the residual, $R(U^n)$, and Fourier transforming we obtain:

$$Z\hat{u}^{n+1} = Z\hat{u}^n + Z^{1st} (\hat{u}^{n+1} - \hat{u}^n) \quad (5.20)$$

Finally, including the first-order time implicit integration we arrive at:

$$(I - \tilde{Z}^{1st}) \hat{u}^{n+1} = [I + \tilde{Z} - \tilde{Z}^{1st}] \hat{u}^n \quad (5.21)$$

where $\tilde{Z}^{1st}$ is obtained taking into account only the first order artificial viscosity terms, (setting $S = 1$) in the discretisation of the artificial viscosity fluxes. When the linear system given by 5.21 is solved inverting the matrix of the system, the amplification factor becomes:

$$G^{exact, 1st-J} = (I - \tilde{Z}^{1st})^{-1} [I + \tilde{Z} - \tilde{Z}^{1st}] \quad (5.22)$$

Similarly to the previous section, the Jacobi iteration for this system is:

$$(\hat{u}^{n+1})^{j+1} = (D^{1st})^{-1} \left([I + \tilde{Z} - \tilde{Z}^{1st}] \hat{u}^n - O^{1st} (\hat{u}^{n+1})^j \right) \quad (5.23)$$

The amplification matrix can then be defined as:

$$G^{jacob_i, 1st-J} = M_{1st}^j \quad (5.24)$$

where the matrix M_{1st}^j is obtained iteratively after j iterations as:

$$\begin{aligned} M_{1st}^{j+1} &= (D^{1st})^{-1} \left(I + \tilde{Z} - \tilde{Z}^{1st} - O^{1st} M_{1st}^j \right) \\ M_{1st}^0 &= I \end{aligned} \quad (5.25)$$

5.3 Stability of the Linear Advection Equation

The two-dimensional linear advection equation, propagated with a velocity $v > 0$ and an angle $\alpha \in [0, \pi/2]$, is formulated as:

$$\frac{\partial \phi}{\partial t} + \nabla (v \cos \alpha, v \sin \alpha) = 0 \quad (5.26)$$

Only the convective and artificial viscosity terms are considered here. The Jacobian matrices that appear in this problem reduce to:

$$A_x = |A_x| = v \cos \alpha \text{ and } A_y = |A_y| = v \sin \alpha \quad (5.27)$$

The spatial operator times the time step, $\tilde{Z} = \Delta t Z / \vartheta$, depends only on the non-dimensional parameters CFL , Λ and α , that is,

$$\tilde{Z} = f(\alpha, \Lambda, CFL, m\Delta x, n\Delta y) \quad (5.28)$$

For the particular case of the JST scheme, this expression is:

$$\begin{aligned} \tilde{Z} &= \frac{CFL}{2(\Lambda \cos \alpha + \sin \alpha)} \left\{ -i \left[\Lambda \cos \alpha \sin(m\Delta x) + \sin \alpha \sin(n\Delta y) \right] + \right. \\ &\quad + \left[\Lambda \cos \alpha (\cos(m\Delta x) - 1) + \sin \alpha (\cos(n\Delta y) - 1) \right] \cdot \\ &\quad \cdot \left[S - \frac{(1-\kappa)(1-S)}{4} (\cos(m\Delta x) + \cos(n\Delta y) - 2) \right] \Big\} \end{aligned} \quad (5.29)$$

It can be shown that

$$\tilde{Z}(\alpha, \Lambda) = \tilde{Z}\left(\pi/2 - \alpha, \frac{1}{\Lambda}\right) \quad (5.30)$$

This represents a rotation of the mesh and the velocity angle. Additionally, there are two limit solutions: when $\Lambda \cos \alpha \gg \sin \alpha$ or $\Lambda \cos \alpha \sim \sin \alpha$. The third limit, $\Lambda \cos \alpha \ll \sin \alpha$, is equivalent to the former making use of relation 5.30. Therefore, both limits will be studied here in order to assess the performance of the numerical scheme.

The stability of the implicit integration based on the full Jacobian is not presented here, because it is well known that when $CFL \rightarrow \infty$, the Newton's method and a quadratic convergence are recovered. It is important to point out that the resulting matrix from the full Jacobian does not fulfill the stability requirements imposed by a Jacobi linear solver, and other solution methods have to be pursued. Therefore, from now on, only an explicit 5th-order Runge-Kutta scheme and implicit first-order time integration using the first-order Jacobian will be considered and compared in detail. The terms 'Implicit' and 'Explicit' will be used from now on to refer to these two time integration schemes.

5.3.1 Effect of the spatial discretisation

Figure 5.1 compares the effect on the stability of the JST and MUSCL schemes for both, the explicit and implicit integration schemes. The angle is set to 1° , ($\alpha \ll 1$) to separate the different families of eigenvalues. The main conclusion is that similar performances in terms of iterations are obtained for JST and MUSCL schemes., It can be noticed that the eigenvalues of the JST scheme are spread in different families whereas the MUSCL scheme exhibits a kind of unique collapsed family. This is due to the coupling between the x and y directions created by the pseudo-Laplacian operator, which would not be seen in a structured code, but that is difficult to avoid in an unstructured code. It is worth mention as well that in the $\alpha = 0$ case, both schemes reduce exactly to the one-dimensional case.

The explicit solver damps quickly high frequency errors, meanwhile there is a family of low frequency errors which need many iterations to be converged. On the contrary, the implicit solver is more efficient damping low frequency than high frequency errors. Additionally, the implicit scheme needs much less number of time steps to converge than its explicit counterpart. Note that the implicit solution has been obtained inverting the matrix of the resulting linear system, hence this comparison cannot be readily extended to computing time, since the implicit iterations are much more costly than the explicit ones. The effect of the inner iterations needed to invert the linear system will be studied along the next sub-sections, and then an idea about the computational cost will be obtained. Nevertheless, this is the asymptotic limit in terms of number of iterations of an implicit method based on the first-order Jacobian.

Figure 5.1: Eigenvalue footprints of the amplification matrix for the linear advection equation. $CFL_{explicit} = 3.5$. $CFL_{implicit} = 10^4$. $\Lambda = 1$. $\alpha = 1^\circ$. **Exact implicit matrix inversion.**

It is important to recall that the above discussion is based solely in the effect of the damping in the errors. In practice, errors are convected also downstream, and the relaxation time, or the number of iterations, for convergence depends also on the through-flow time based on the numerical phase velocity, which is proportional to the CFL number. We will come back later on to this point to complete the discussion.

Since there are not significant differences between JST and MUSCL schemes, from now on, only the MUSCL approach will be considered for the sake of brevity.

5.3.2 Effect of the flow angle

When the velocity is not aligned with the cell direction, it is worth to mention which is the amplification of the spatial eigenvalues. The eigenvalue footprints, when flow angle is 45° and the cell aspect ratio is one, are shown in Figure 5.2. This corresponds with

Figure 5.2: Eigenvalue footprints of the amplification matrix for the linear advection equation. $CFL_{explicit} = 3.5$. $CFL_{implicit} = 10^4$. $\Lambda = 1$. $\alpha = 45^\circ$. MUSCL scheme. Exact matrix inversion.

the second limit mentioned ($\tan \alpha \sim \Lambda$). Clearly, the implicit method for angles near to 45° is not as good damping modes as for $\tan \alpha \ll \Lambda$, because there are low frequency errors which are not properly damped. To see which modes are those that lie on the real axis near to the poor damping region, an asymptotic analysis has been performed (see Appendix C) where can be concluded that the spectral radius of the implicit solver when $\tan \alpha = \Lambda$ is approximating to one with the square number of grid points.

5.3.3 Effect of the inner iterations of the implicit method

Up to now, the stability analysis of the implicit methods has been performed using the exact inversion of the linear system. In practice, this approach is seldom used since direct methods are not practical for industrial applications, and typically iterative methods with an incomplete convergence are used. For the sake of simplicity, the Jacobi solver is used to assess the effect of the inner iterations of the linear solver in the pattern of the eigenvalues of the amplification matrix. The evolution of the eigenvalue footprints with the number of the linear solver iterations for $CFL = 10^4$, is shown in Figure 5.3. It can be observed that not only many Jacobi iterations are needed to converge the linear system, but also to stabilise the time integration. This is because the CFL number is quite large and the matrix of the system is not diagonal dominant. In this case the Jacobi solver shows a very poor convergence rate. The quasi one-dimensional high-frequency modes, $H_x L_y$ and $L_y H_y$, are converged quickly, meanwhile both, high-high and low-low frequency modes are

Figure 5.3: Eigenvalue footprints of the amplification matrix for the linear advection equation. Effect of the linear solver iterations. $CFL_{implicit} = 10^4$. $\Lambda = 1$. $\alpha = 45^\circ$. MUSCL scheme.

unstable even with 1000 linear iterations. This is not practical and the optimum values of the CFL and the linear solver iterations need to be estimated.

5.3.4 Computational cost

The computational cost of the global problem is the time needed to converge the partial differential equations. This time is the product between the number of iterations and the time required to compute a single iteration. The number of iterations to converge the problem is given by the spectral radius, if convection effects are neglected, which is the maximum eigenvalue, λ_{max} , of the amplification matrix. That is, during the convergence process, the smallest eigenvalues are smoothed quickly, while the maximum eigenvalue is the last to be converged. When the relative error ϵ_r , which is defined as the ratio between the actual error and the initial error, reaches a certain value, typically the round-off machine error, it is said that the problem is converged. The number of time steps needed to reach a given convergence level is:

$$n_{steps} = \frac{\ln \epsilon_r}{\ln \lambda_{max}} \quad (5.31)$$

The largest eigenvalue depends on the number of grid points, the CFL number and the number of the linear iterations used to solve the linear system in the implicit method.

Firstly, the effect of the number of grid points is shown in Figure 5.4 for both limit cases: $\alpha = 0^\circ$ and $\alpha = 45^\circ$. It has been found that the spectral radius for large number of points follows the law:

$$\lambda_{max} = 1 - \beta N_p^a \quad (5.32)$$

Figure 5.4: Effect of the Number of grid Points ($N_p = N_x = N_y$) on the spectral radius for $\alpha = 0$, (left) and $\alpha = 45^\circ$ (right).

where N_p is the number maximum number of grid points, in the x and y directions, $N_p = \max(N_x, N_y)$ and, β and a two constants.

For the explicit method it can be seen that the slope for large N_p is $a = -4$. From an asymptotic analysis of the amplification factor for very low wave numbers ($\hat{m} = m\Delta x \ll 1$), see Appendix C. It can be concluded that in this case:

$$G \simeq 1 - \mu_4(m\Delta x)^4 \quad (5.33)$$

this is as it could be expected since for $\hat{m} \ll 1$, the temporal integration is nearly exact and the damping is due to the artificial viscosity terms, which in the long-wave length limit scales as $\hat{m}^4$. Actually, if the same exercise is performed with a 1st-order scheme, then it can be shown that

$$G \simeq 1 - \mu_2 \text{CFL} (m\Delta x)^2 \quad (5.34)$$

and hence in this case the slope is $a = -2$. This actually means that temporal integration scheme does not participate in the damping of this modes. It should be recall that these modes are in any case convected with a numerical phase velocity which is very close to the physical one.

For the implicit integration scheme the situation is somewhat different. It can be seen that the slope of λ_{max} with N_p for $N_p \gg 1$ is $a = -2$. Conducting an asymptotic ($\hat{m} \ll 1$)

analysis of the amplification factor of the implicit method it can be proved as well that

$$G \simeq 1 - \text{CFL}^2 (m\Delta x)^2 \quad (5.35)$$

Conceptually the main difference between the explicit and the implicit method is that, while the damping of the long-wave lengths in the explicit method is due to the spatial discretisation, in the implicit method the damping is caused by the temporal scheme. It should be emphasised that these long-wave length modes are also very well propagated by the scheme.

When the flow angle is $\alpha = 45^\circ$ the situation is quite different to that previously outlined, as can be seen in Figure 5.39 (b). The explicit Runge-Kutta scheme has the same asymptotic behaviour as the case $\alpha = 0^\circ$ for large number of points, N_p , but the trends of the implicit solver are significantly different. For moderate values of the CFL, (i.e., $\text{CFL} \sim O(1)$) it can be shown (see Appendix C) that:

$$G \simeq 1 - \frac{\mu_4}{4} \text{CFL} (m\Delta x)^4 \quad (5.36)$$

this actually means that the asymptotic behaviour of both, the explicit and implicit schemes are very similar. The underlying reason is that the asymptotic behaviour of the implicit scheme for the $\alpha = 0^\circ$ was control by the imaginary part of the eigenvalue. However, for the $\alpha = 45^\circ$ case, there are modes ($m\Delta x = -n\Delta y$) for which the imaginary part of the eigenvalue is zero, and therefore the asymptotic behaviour is totally different. This behaviour can be seen in Figure 5.39 (b) where the implicit method for $\text{CF} = 3.5$ and 10 exhibit a slope very close to the explicit method ($a = -2$). The modes associated to the largest eigenvalues are long wave-length modes ($m\Delta x \ll 1$).

The trend for large CFL numbers ($\text{CFL} \gg 1$) is different, as can be noticed in Figure 5.39 (b) for $\text{CFL}=10^4$. In this case it can be shown that

$$G \simeq 1 - \mu_4 \text{CFL} (m\Delta x)^2 \quad (5.37)$$

and therefore the slope of the curve is $a = -2$. In this case it can be shown that the modes controlling the convergence of the system are high frequency modes. In the same line it can be demonstrated that for $\alpha = 0^\circ$ and $\text{CFL} \gg 1$, the amplification factor is:

$$G \simeq 1 - 2\mu_4. \quad (5.38)$$

This actually means that the largest eigenvalue is independent of the number of points. This limit case corresponds to the horizontal line in Figure 5.39 (a) for $CFL=10^4$. It can be concluded that the trends presented in this subsection are well understood and supported by a relatively simple analysis.

The asymptotic analysis delivers as well very valuable information about the trends of λ_{max} with the CFL number, which is different for the explicit and the implicit methods. The effect of the CFL on the spectral radius is shown in Figure 5.5. It has been found that the spectral radius satisfies the following law with the CFL number:

$$\lambda_{max} = 1 - \gamma CFL^{b_2} \quad (5.39)$$

for low enough CFLs ($\hat{m} CFL \ll 1$), which is consistent with the previously outlined asymptotic analysis that retains the effect of the CFL. The trend is valid for both, the explicit and the implicit solvers, being the slope equal to $b = -1$ for the former, independently of the flow angle, whereas for the implicit solver this slope is $b = -1$ and $b = -2$ for flow angles of 45° and 0° respectively. When the CFL is large enough the largest eigenvalue reaches a maximum threshold, which is $1 - 2\mu_4$ for $\alpha = 0^\circ$ and $1 - \mu_4 (m\Delta x)^2$ for $\alpha = 45^\circ$ as it has been demonstrated in Appendix C.

When the implicit method is solved using the Jacobi solver, the maximum eigenvalue depends on the number of the inner iterations of the linear system, as it was shown in Figure 5.3. Moreover, when the number of the linear iterations is fixed, there is a maximum CF beyond which the global problem becomes unstable. This does not mean that the Jacobi method is unstable, but the number of iterations of the Jacobi solver is not enough to approximate the eigenvalues obtained for a exact matrix inversion. In other words, the linear solver is not sufficiently converged to guarantee the stability of the integration of the original PDE. From Figure 5.5, this CFL maximum can be derived as:

$$CFL_{max} \simeq 2.5 \cdot n_{inner} \quad (5.40)$$

being n_{inner} the number of the Jacobi iterations for the linear system.

The time to compute one iteration is proportional to the number of evaluations of the residual, and the time required to compute the residual is proportional to the grid size. In summary, the computational time non-dimensionalised by the time to evaluate the

Figure 5.5: Effect of the CFL and the number of Jacobi iterations on the spectral radius. Left: $\alpha = 0^\circ$. Right: $\alpha = 45^\circ$ ($N_p = 60$).

residual can be estimated as:

$$\frac{t_c}{t_r} = n_{steps} \times n_{inner} = \frac{\ln \epsilon_r}{\ln \lambda_{max}} \times n_{inner} \quad (5.41)$$

being t_c the time needed to reduce the relative error a magnitude of ϵ_r , t_r the time to evaluate the residual, n_{inner} the inner iterations of the time integrator (five for 5th-order Runge-Kutta and the number of linear iterations for the Jacobi linear solver).

Equation 5.41 is plotted in Figure 5.6 taken the values from Figure 5.5. The most important result from Figure 5.6 is that the minimum time for the Jacobi implicit solver is found for CFL values between 10 and 30, using 10 inner iterations for both limit cases, $\alpha = 0^\circ$ and $\alpha = 45^\circ$. For larger CFL values, more number of the inner iterations of the linear solver are needed, and the overall time is increased drastically. Nevertheless, the Jacobi solver improves the convergence time by a factor of 3 with respect the explicit 5-stage Runge-Kutta method in the worst case, i.e., when the flow angles is $\alpha = 45^\circ$. When $\alpha = 0^\circ$, the Jacobi implicit method is extremely efficient compared with the Runge-Kutta scheme, three orders of magnitude faster.

Considering the worst case, $\alpha = 45^\circ$, the only way to reduce the computational cost of the implicit method is to accelerate the convergence of the linear system using more efficient linear solvers. If there was a linear solver which were able to converge the linear system in less number of iterations than the Jacobi method, the resulting curve of the

Figure 5.6: Effect of the CFL and the number of Jacobi iterations on the computational cost. Left: $\alpha = 0^\circ$. Right: $\alpha = 45^\circ$ ($N_{\text{points}} = 60 \times 60$).

computational cost would be equivalent to one the Jacobi method shifted down to the corresponding curve with the equivalent number of iterations. This means that higher CFL numbers could be used with the new linear solver and the computational cost further reduced as it is shown in Figure 5.7, where a hypothetical solver able to converge the linear system using one order of magnitude less iterations than the Jacobi solver has been used. Using this new solver, the computational cost could be improved by a factor of 20 with respect the original explicit solver.

Once the computational cost has been assessed, It is possible to compare the baseline 5-stage Runge-Kutta scheme and the implicit solver with its optimum parameter setting, that is, an implicit method based on the first-order Jacobian matrix solved with the Jacobi linear solver with a $CF=20$ and 10 linear iterations. This is shown in Figure 5.8. The best convergence rate is obtained with the implicit solver and CFL numbers as large as possible (Figure 5.8 (b)). If the CF number is reduced to 20, there is a considerable penalty in the convergence rate (Figure 5.8 (c)). There is a second degradation if the linear solver is not solved completely and only 10 iterations are used (Figure 5.8 (d)). Nevertheless, the convergence rate is better than the baseline explicit method (Figure 5.8 (a)).

Finally, it is worth to mention that all the previous analysis establish the rate at which different modes of the amplification matrix are damped. Moreover it is important to consider the phase velocity of the modes, that for a non-periodic domain, determines

Figure 5.7: Computational cost of a solver that could converge the linear system in 10 iterations to the same level than the Jacobi method with 100 iterations.

Figure 5.8: Influence of the time integration procedure on the eigenvalue footprint. $\Lambda = 1$, $\alpha = 45^\circ$, MUSCL scheme. (a) Explicit and $CFL = 3.5$. (b) Implicit with exact integration and $CFL = 10^4$. (c) Implicit with exact integration and $CFL = 20$. (d) Implicit with 10 iterations of Jacobi linear solver and $CFL = 20$.

the rate at which different waves are expelled out of the domain, and therefore is directly linked to the convergence speed of the method. Considering the one dimensional problem, that is, $\alpha = 0^\circ$ and $n\Delta y = 0$, the non-dimensional phase velocity of an error mode is:

$$\frac{v_{phase}}{v} = \frac{\arctan(G_I/G_R)}{CFL \cdot m\Delta x} \quad (5.42)$$

Figure 5.9 represents the non-dimensional phase velocity and the modulus of the amplification matrix for the explicit and implicit solvers as a function of the non-dimensional wave-number for different CFLs. It can be concluded that high frequency errors are poorly convected, but they are properly damped by both the explicit and

Figure 5.9: Phase velocity (top) and modulus (bottom) of the amplification matrix for 1D linear advection equation and different schemes.

implicit schemes. For the explicit solver, the low frequency errors are barely damped, and they are convected proportionally to the CFL number, which represents the non-dimensional time step. This means convection is an essential mechanism for the convergence of the explicit solver. Regarding the implicit solver, for large CFL numbers, both the damping and the propagation of the low frequency errors are extremely high. Nevertheless, for low CFL numbers, of the order of 10, the implicit solver is able to damp most of the modes. Only a very small set of low frequency modes are slightly damped, but they are effectively convected nevertheless.

Finally, the spectral radius of the amplification matrix and the numerical phase velocity are represented in Figure 5.10 as a function of the CFL number for different number of inner iterations of the Jacobi solver. It is worth noting that the vertical scale of Figure 5.10 (top) represents the numerical phase velocity times the CFL, because this is a measure of the propagation of the different waves per time step, and hence of the efficiency of the scheme to expel errors out.

At low CFL numbers, the phase velocity is close to the nominal and errors are propagated proportionally to CFL number. This is especially important since for low CFL numbers low frequency errors are weakly and the least damped. In this range, the implicit solver converge faster than the explicit one because the difference in the CFL number. Only when the CFL number is greater than 20, the least damped mode becomes a high frequency

Figure 5.10: Non-dimensional convection distance (top) and modulus (bottom) of the amplification matrix as a function of the CFL number for different number of the inner Jacobi iterations. for the 1D linear advection equation.

mode, and is not convected anymore. At this point the convergence rate is controlled by the damping, and increasing the CFL number does not make any difference. Figure 5.10 (top) displays the non-dimensional convection distance as a function of the CFL and a steep drop to zero can be seen around $\text{CFL}=20$. Simultaneously, the damping of the least stable mode reach a maximum. This is the optimum CFL since further increasing the CFL does not help neither to propagate nor damp the modes and since increasing the CFL requires more inner iterations to converge the linear system, this is the optimum CFL in terms of computational cost.

5.4 Stability of the Euler equations

The Euler equations are derived from the RANS equations neglecting the viscous terms. The analysis performed here, as it has been mentioned before, is based on the linearisation of the Euler equations about a mean uniform flow using periodic boundary conditions.

Figure 5.11: Influence of the spatial discretisation and time integration on the eigenvalue footprint. Top: Explicit (CFL=3). Bottom: Implicit (CFL=20) $M = 0.5$, $\Lambda = 1$, $\alpha = 0$.

The spatial operator times the time step, $\tilde{Z}_{mn}$ depends on:

$$\tilde{Z}_{mn} = f(M, \alpha, \Lambda, CFL)$$

The main difference with respect the scalar advection equation is the compressibility effect, taking into account through the Mach number. Both, the JST and the MUSCL schemes, with or without Block-Jacobi preconditioning, and with scalar artificial dissipation or matricial artificial dissipation are considered. Due to the large number of combinations of numerical schemes and parameters, just a selection of representative cases is presented. Moreover, since it has been concluded from the analysis of the linear advection problem, that the optimum set of parameters for the Jacobi solver are $CFL \simeq 20$ and 10 inner iterations of the Jacobi solver. These values will be kept constant from here on for the analysis of the implicit method.

As the Euler equations contain five families of waves, the eigenvalue footprints will have to reflect this reality. In this particular analysis, where the solution is periodic in the

z direction and the velocity is contained in the xy plane, only four families appear in the problem. There are different approaches to add artificial viscosity to every family. The first approach consists of adding the same amount of artificial viscosity to all the waves, scaling with the maximum eigenvalue to increase the robustness and guarantee the convergence. This approach is easy to implement and requires a small amount of storage since the manipulation of 5×5 sub-block matrices is avoided. The second approach is based on scaling every wave with its own eigenvalue by means of Roe's formulation of the numerical fluxes. The numerical problem discretised with the Roe's matrix can be integrating just with the local time step or preconditioned with the Block-Jacobi matrix. This three combinations are represented in Figure 5.11 for both the explicit and implicit methods. Analogously to the linear advection problem the eigenvalue footprint patterns are obtained, but in this case three families corresponding with the entropy, vorticity and acoustic waves can be seen. The JST scheme using scalar formulation of the artificial viscosity is shown in Figures 5.11 (a) and (d) for both, the explicit and implicit method, respectively. The high frequency modes are well clustered in the maximum damping area, meanwhile the low frequency modes are near origin and are slightly damped. This means that a multi-grid smoother will work properly if it is implemented. When the matricial formulation is used, some high frequency modes are moved away from the real axis which means that are better convected and less damped in both methods (Figures 5.11 (b) and (e)). Two high frequency families are moved to areas with lower damping and will induce a deterioration of a hypothetical multi-grid preconditioner, especially the family LxHy, which represents low frequency modes in the x direction and high frequency in y direction. This effect disappears when the flow is not aligned with the cell as it is represented in Figure 5.12. When the Block-Jacobi is activated (Figures 5.11 (c) and (f)), the eigenvalues are moved a little bit farther away from the real axis. Some LxHy modes are not concentrated in the Fourier space origin and only few of them are still there. Therefore, when Roe's artificial formulation is used, it is recommended to use the Block-Jacobi preconditioner. This is a standard technique use both in explicit an implicit solvers.

Regarding the computational cost, the spectral radius (λ_{max}) is an indication of to the

Figure 5.12: Eigenvalue footprint when the fluid is not aligned with the cell. MUSCL-Roe + LTS. $M = 0.5$, $\Lambda = 1$, $\alpha = 40$.

time required to converge the simulation if the CFL is constant, since high frequency modes, which are not effectively transported, are eliminated by the numerical damping. Indeed, since the Jacobi solver uses twice the evaluation of the RK5 method, the time to converge the problem according with Equation 5.41 is inversely proportional to the logarithm of the spectral radius. That is, the closer to the unity the spectral radius is, the larger the convergence time. Analogously to the linear advection problem, implicit methods exhibit lower values of the spectral radius than its explicit counterpart.

The only physical parameter which is retained in the Euler equations is the Mach number. Low Mach number flows are quite common in turbomachinery applications, especially in the secondary system. The spatial eigenvalue footprints of the different spatial discretisations at low Mach number are shown in Figure 5.13. For all the cases, the entropy and vorticity waves are collapsed into the real axis, because their convection velocity is low. This is an undesirable scenario because these waves are overdamped and the steady state solution is not well resolved. The stiffness of the low Mach number flows is overcome with the use of Block-Jacobi and low Mach number preconditioning.

Finally, the effect of the cell aspect ratio is studied. Stretched cells are typically used to solve properly boundary layer and wake regions, therefore the limit $\Lambda \rightarrow 0$ is also studied. The result for this case is shown in Figure 5.14. Entropy and vorticity waves are poorly convected by the JST scheme, due to the disparity of scales between the acoustic and convective waves, that is reinforced by the cell aspect ratio. When Roe's matrix formulation and local time step preconditioning is used, this stiffness is even more critical,

Figure 5.13: Influence of the spatial discretisation and time integration on the eigenvalue footprint. $M = 0.05$, $\Lambda = 1$, $\alpha = 0$.

because those modes are clustered in the region of minimum damping (see Figure 5.14(e)). Additionally, there is a family of $LxHy$ acoustic modes which are poorly damped, which means that a high-frequency mode in the perpendicular direction of the stretching is not properly damped, and then multi-grid method will not work in that scenario. Adding Block-Jacobi preconditioning, which means that the time step of each family of modes is scaled with its own eigenvalue, this family is moved far away for that region. Regarding the implicit scheme, although the CFL number is large, the Block-Jacobi preconditioning still produces an important benefit for this kind of configurations because in this case $CFL \times \Lambda \ll 1$.

Figure 5.14: Influence of the spatial discretisation and time integration on the eigenvalue footprint. $M = 0.5$, $\Lambda = 10^{-4}$, $\alpha = 0$.

Chapter 6

Multi-Core and Many-Core Implementation

6.1	Introduction	83
6.2	Serial edge loop	87
6.3	Distributed memory parallelisation	89
6.4	Shared memory parallelism	92
6.5	Implicit Solver Parallelisation	100

6.1 Introduction

The use of Graphics Processing Units (GPUs) for general purpose computing is becoming increasingly popular because of the outstanding computing performance of GPUs and the popularisation of programming languages that ease their programming.

GPUs contain a large number of processor elements, of the order of thousands, which are connected to the GPU memory through a data bus that provides a high bandwidth when transferring data between the memory and the processor. While the amount of data residing in the processor is small, the access to additional data is very fast. CPUs, however, feature a number of processors not as large as that of GPUs, and data transfer between

memory and processors occurs at a much lower rate. These limitations are partially overcome by the existence of the processor's cache memory, which allows an efficient reuse of a certain amount of already accessed data. GPUs, on the contrary, have a much reduced cache memory compared with that of CPUs. A rough comparison of the characteristics of a modern CPU and a GPU is presented in table 6.1. These differences in the processor architecture have had a direct translation in the programming languages which are most suitable for each device. GPU programming traditionally required the knowledge of specific programming languages with an intricate nomenclature that was hard to translate to general purpose applications. This situation changed substantially with the release of CUDA ([72]) by NVIDIA in 2006. CUDA is an extension to the C programming language that makes the execution of a general purpose program in an NVIDIA GPU much easier. Two years later, another programming language named OpenCL ([55]) appeared as a joint effort of the industry to provide a standard, non-proprietary programming language for heterogeneous multi-processor platforms. Programs written in OpenCL are not only able to be executed on GPUs, but on any multi-processor platform, such as Intel multi-core CPUs, IBM Cell processors, etc. This paradigm offers a clear advantage with respect to CUDA; different multi-core platforms are supported with exactly the same code, ensuring its usability despite future changes in the prevalent architecture. As a result of these developments, the effort to write and execute a code on CPUs or GPUs has become comparable, triggering the appearance of more applications specifically designed for GPUs. By the time CUDA was released to the public domain, the computing performance of GPUs was roughly one order of magnitude greater than that of CPUs, and therefore the number of CFD applications developed for GPUs ever since has grown substantially (see for instance [29, 13, 28, 31, 9, 19, 80]). The use of GPUs in fields closely related with CFD such as level set methods ([52]) and novel CFD methods ([90]) is not negligible either. A comprehensive review of various uses of GPUs in Computational Fluid Dynamics, as well as some implementation guidelines, can be found in [70].

With the release of the next generation of multi-core CPUs like the Intel Xeon Phi co-processor, the performance gap with current GPUs has been reduced ([33]), making a future choice of computing hardware less obvious than it is nowadays. In any case, it

Table 6.1: Comparison of representative properties of a modern CPU and a modern GPU.

	Intel Xeon E5-1660 v4	NVIDIA Tesla P100	GTX 780i
Number of processors	8	3584	2304
Bandwidth (GB/s)	76.8	732	336
L2 Cache Size	2 MB	4 MB	1.5 MB
L3 Cache Size	20 MB	-	-

is mandatory for the present and future CFD solvers to efficiently run in highly parallel environments to fully exploit current and future hardware capabilities ([54]).

This Chapter presents the steps followed to re-design an existing RANS solver, known as Mu^2s^2T ([16]), to increase its parallel performance by executing it on a variety of multi-core platforms using both explicit and implicit methods.

The primary concern of the re-design is to exploit the increased parallel capabilities of existing GPUs, but since the choice of the most suitable parallel hardware is far from being a closed subject, the ability to switch between different hardware platforms is also considered critical. By the time the development was initiated, the easiest way of fulfilling the first requirement was to use either CUDA or OpenCL as the programming language to code the RANS solver and execute it on GPUs. These two languages provided tools mature enough to optimise the code execution and debug on many-core platforms, such as GPUs. By choosing CUDA, the target hardware is limited to NVIDIA GPUs, while other platforms such as AMD or Intel would be currently left out; for this reason, OpenCL was chosen as the coding language, which is not only supported by NVIDIA but as well by many other vendors. Additionally, clusters of CPUs ought not to be disregarded, thus the same code executed on a GPU should also be able to be executed on a CPU with minimum programming effort or performance penalty. This is accomplished by using C++ as a wrapper for the OpenCL code, since it provides enough flexibility to naturally integrate the source code written in OpenCL as part of the C++ classes that manage the simulation. This, combined with the use of compiler directives, allows the generation of a unified source code to be executed both on CPUs and GPUs, avoiding the need to have separate solvers for different architectures. This choice has in turn several positive aspects: (i) this monolithic approach ensures that differences in the results due to algorithm modifications between CPUs and GPUs do not exist. Variations in the results

are solely due to the use of different hardware and are equivalent to those that can be seen executing the same code with different compilers or different compiler options and (ii) the resulting code can be executed on existing CPU clusters that would otherwise become obsolete. Other authors have developed and validated code translators ([28]) or high level directives to express the code parallelism ([9]) with remarkable results. One of the most popular tools is OpenACC ([75]), a set of compiler directives used to express parallelism. It has been reported that OpenACC can achieve similar performances to those of CUDA or OpenCL, especially for large enough cases ([70]), at a much lower implementation cost. For modest size cases, however, CUDA or OpenCL are still the best choice. The use of automatic translators is understandable if there is a large amount of legacy frozen code that needs to be quickly translated but is undesirable for codes that need to be maintained and are involved in a continuous improvement process. The use of translators introduces additional complexity and forces the developer to follow strict programming rules in order to keep pace with the translator rules, a situation that is avoided in the present approach.

A weakness of the use of GPUs as computing devices is the bottleneck that poses the communication between the GPU and the CPU used to control it. The data transfer rate between the GPU memory and the processor is one order of magnitude higher than the transfer rate between GPU and CPU memory. To overcome this limitation, most of the computations must be carried out on the GPU in order to minimize the data exchange between GPU and CPU. Indeed, this is the approach followed in this work. All the routines needed to perform either a Runge-Kutta or an implicit iteration are programmed in OpenCL. The block-Jacobi preconditioning matrix, the inviscid and viscous fluxes, the boundary conditions and the update of the conservative variables are entirely computed on the GPU. The only exception to this is the process of exchanging the sub-domain frontier data between the GPU and CPU to perform the GPU to GPU communication, that need to be performed when several GPUs are used in parallel, since there is not any supported OpenCL method to exchange data directly among the GPUs without relying on the CPUs that control them. In other words, all the data transferred between two GPUs necessarily need to pass through the CPU-GPU bottleneck. This is not the case anymore if the OpenCL 2.0 standard ([55]) is used, but this language specification is not

currently supported by NVIDIA GPUs, hence the newest OpenCL features have not been exploited in this work. To minimize the penalty associated with this issue, a proper design of CPU to GPU communications has been devised and will be explained in section 6.4.3.

Access to computational resources is always difficult in engineering companies due to budget restrictions, and therefore the affordability of the simulations is a major concern for the deployment of high-fidelity methods. In this work, we have followed the route of using commodity GPUs, which are not only widely available, but much less costly than GPUs specifically designed for computing, such as the Tesla graphics cards from Nvidia, which are installed on most of the supercomputers. Commodity GPUs, such as the GTX card, have higher bandwidths (see Table 6.1) but lack of (i) ECC memory to prevent data corruption, (ii) hyper-Q, to enable multiple CPU threads or processes to launch work on a single GPU simultaneously, (iii) RDMA, to easily port communications to Infiniband, and (iv) double precision functionalities. In our experience, in practice, only data corruption due to hardware malfunction is relevant in our case, since the queuing of jobs in the GPU is implicitly handled by the algorithm, whereas the communication among GPUs is performed through dedicated Infiniband cards. Moreover, the use of single-precision arithmetic for aerodynamic applications has been used in our group for many years without any obvious practical consequences (see for instance [16]).

6.2 Serial edge loop

The computation of the gradient (Equation (2.33)), block-Jacobi preconditioning matrix (Equation (2.71)) and fluxes (Equation (2.32)) involve all a loop over edges like the one sketched in Algorithm 6.1, and they are by far the most time consuming parts of the numerical scheme. When the code is executed on a CPU and the time per Runge-Kutta iteration is measured, the computation of the block-Jacobi preconditioning takes about 10% of the total time, the gradient 5% and the fluxes 70%. All together, they represent 85% of the total execution time. Therefore an efficient implementation of these loops is crucial in order to obtain an efficient solver, and the optimal approach strongly depends on the underlying hardware where the code is executed.

Algorithm 6.1 Generic edge loop for an edge-based solver. `ReadPointData` represents the point data needed to perform the inner loop computations, `F(data1,data2)` the inner loop operations and `WritePointResult` the writing of the resulting data. `Nedges` is the number of grid edges and `edgeNodes` is the edge-node connectivity.

```
void edgeComputations(...)
{
    for(edge=0;edge<Nedges;edge++)
    {
        point1 = edgeNode(1,edge);
        point2 = edgeNode(2,edge);
        data1  = ReadPointData(point1);
        data2  = ReadPointData(point2);
        term   = F(data1,data2);
        WritePointResult(point1,term);
        WritePointResult(point2,term);
    }
}
```

Figure 6.1: Reverse Cuthill-McKee ordering to minimize cache-misses.

In cached-processors like standard modern CPUs, when the executing process requires data from the processor memory, the operating system temporarily places these data in the cache memory. Not only the required data, but also a block of contiguous data is gathered. While these data remain in the cache, they can be re-used at no cost. If data outside of this block are required, then they must be re-taken from the main memory again, slowing the whole process. This action is called a cache miss. The need to optimize indirect addressing in edge-based solvers to avoid cache misses was acknowledged from the conception of this type of solvers ([59]). The extra cost of searching data in the main memory is usually between 2 and 30 times that required to access the same datum in the cache memory. To minimise the number of cache misses during the execution of the edge loop (Algorithm 6.1) the grid nodes are renumbered and the edges reordered to make nearby edges point to nearby points. This can be accomplished using the reverse Cuthill-McKee (RCM) ([30]) ordering technique. The resulting edge-node connectivity is presented in Figure 6.1, where a mesh with 25 nodes and 40 edges has been ordered using this technique.

When the problem size grows beyond the capacity of a single CPU or the execution time is too large, the problem has to be split in multiple parts to solve it in parallel. Two types of parallelisation have been implemented, depending on whether the data is shared among parallel processes or not.

6.3 Distributed memory parallelisation

When the parallel processes do not share the data residing in memory, the information between them is exchanged using the Message Passing Interface (MPI). This parallelism has been implemented to make use of all CPU cores (even though they might share the same physical memory) or to use more than one CPU. Ideally, if the time of computing a serial edge loop is t_s the time of the parallel algorithm should be t_s/P , where P is the number of processors. However, there is always an intrinsic overhead due to the parallelisation, usually associated with four facts:

- Varying processor efficiency with the problem size, i.e., $C_{eff}(P)$. Some processors perform faster when the problem size is small, such as cache-based CPUs, then $C_{eff} > 1$, while others perform slower, like GPUs, yielding a $C_{eff} < 1$. The time to execute the edge loop in parallel is then $t_s / (P \cdot C_{eff})$.
- Load imbalance of the different processes, i.e., not all processes compute the same number of edges in the edge loop. The one with the largest imbalance will need an extra time Δt_i to complete its part of the edge loop, while the rest of the processes are inactive while waiting to synchronise.
- Computation overhead, i.e., the extra-cost associated with the fact of splitting the edge loop computation across several processors. The edges belonging to the parallel domain frontier are computed twice, once for each parallel domain. Therefore, if E_i is the number of edges for sub-domain i , the computation overhead is expressed as

$$C_{oh} = \frac{\sum_{i=1}^N E_i}{E_s} = 1 + \frac{E_f}{E_s} \quad (6.1)$$

being N the number of parallel sub-domains, E_f the number of parallel frontier edges and E_s the number of edges of the entire fluid domain.

- Communication overhead, i.e., the time spent exchanging data among parallel processes, t_c . The larger t_c , the worse the parallel performance.

Considering these factors, the actual parallel speed-up, S_a , defined as the relation between the serial and the parallel execution of the edge loop, is

$$S_a = \frac{t_s}{\frac{t_s}{P} \frac{C_{oh}}{C_{eff}} + \Delta t_i + t_c} \quad (6.2)$$

The ratio between the actual and the ideal speed-up, P , is the parallel efficiency

$$\xi = \frac{1}{\frac{C_{oh}}{C_{eff}} + \left(\frac{\Delta t_i}{t_s} + \frac{t_c}{t_s} \right) P} \quad (6.3)$$

This expression highlights various mechanisms to increase parallel efficiency:

- Increase t_s by solving larger problems (the so called weak parallel scaling) and ensuring that $\Delta t_i/t_s$ and t_c/t_s decrease with the problem size by choosing a proper

partitioning algorithm that minimizes the load imbalance and the size of the parallel sub-domain frontiers. In this work, an efficient load balancing is achieved by using the METIS library ([53]), which provides methods to split a given edge graph in properly balanced sub-graphs minimizing the size of the domain frontiers (see figure 6.2). This in turn reduces the computation overhead C_{oh} which is proportional to the number of frontier edges E_f .

- Use processors whose $C_{eff} > 1$ when the problem memory size decreases, like cache-based CPUs. When the problem size is small enough to fit inside the cache the number of cache misses is zero and memory access much faster than for the larger serial problem. However, this effect is barely noticeable for most CPUs.
- Limit the communication overhead t_c . The communication time is the sum of the network latency time t_l , i.e., the time needed to establish the communication, and the time effectively spent sending the message, which is proportional to the message size M and inversely proportional to the network bandwidth BW :

$$t_c = t_l + \frac{M}{BW} \quad (6.4)$$

From the hardware point of view a low-latency, high-bandwidth network is desired. In this work we use an Infiniband network with $BW = 100Gb/s$ and $t_l = 1\mu s$; therefore, for large enough message sizes, $t_c \simeq M/BW$. Once BW is fixed, there is still room for improvement, by either reducing the message size by minimizing the frontier size, or with an appropriate design of the communication strategy. Some MPI implementations enable the possibility of overlapping communication and computation during the execution, by using the so-called non-blocking or asynchronous communications. The parallel sub-domain edge loop is then split into two parts to obtain correct results. The first part contains all those inner domain edges, i.e., edges that do not need to exchange information with the neighboring domains to produce correct results. The second part of the edge loop contains only frontier edges. The communication is started before the execution of the first edge loop and must be completed before the second part of the edge loop is executed. In

Figure 6.2: Mesh split in 16 sub-domains using the ParMETIS library routines ([53]).

that case the effective communication time is

$$t_{eff} = t_l + \max(0, t_c - t_{loop}) \quad (6.5)$$

As long as the cost of computing the contributions of the inner domain edges is greater than the cost of communicating the parallel frontier data, the effective communication time will almost vanish, which in turn increases the parallel efficiency.

6.4 Shared memory parallelism

In the shared memory parallelism, all the processors share the same memory space. That is certainly the case of the GPUs, where literally thousands of threads can be executed simultaneously, and all access the same physical memory at a very fast rate. According to Equation (6.3), for very large values number of parallel processes P , an efficient parallel algorithm should have the same work load (i.e., $\Delta t_i = 0$) and the data exchange between them kept to a minimum ($t_c \simeq 0$). Since the edge is the minimal entity of an edge loop, a good choice to ensure a balanced load when the number of parallel processes is very high is assigning each thread of the computation a single edge.

Algorithm 6.2 OpenCL kernel version of Algorithm (6.1).

```
__kernel void edgeComputations(...)
{
    edge    = get_global_id(0);
    point1  = edgeNode(1,edge);
    point2  = edgeNode(2,edge);
    data1   = ReadPointData(point1);
    data2   = ReadPointData(point2);
    term    = F(data1,data2);
    WritePointResult(point1,term);
    WritePointResult(point2,term);
}
```

OpenCL nomenclature Before moving on to present the parallelisation of an edge loop according to the one-thread-one-edge criterion, a brief clarification of the OpenCL nomenclature is needed to follow the next discussion. The functions in OpenCL that are executed on a multi-processor device (a GPU being a particular case of multi-processor) are called kernels. A call to an OpenCL kernel creates a number of processes called threads (in CUDA nomenclature), or work items, and distributes their execution across each of the device multi-processors. Each work item executes the compiled kernel source code. Thousands of work items are typically created on a GPU. The total number of work items of a given kernel is called the kernel global size.

The OpenCL kernel for an edge loop equivalent to that of a CPU (see Algorithm 6.1) is presented in Algorithm 6.2, which is very similar to Algorithm 6.1 except that there is no loop, since each work item runs independently of the others, computing just an edge whose index is given by the `get_global_id(0)` function that returns, for each item, which is its rank within the total number of processes. When this kernel is executed on the GPU without any condition, the result will most likely be wrong because two different work items can access the same memory position at the same time.

When two threads simultaneously read data from the same memory position, one of the work items must wait for the other to finish, the kernel execution time is increased since the data transfer rate is smaller, but the overall result is correct. However the

Figure 6.3: Reverse Cuthill-McKee followed by an ordering by groups to avoid simultaneous memory access within a group.

simultaneous write operation is not properly handled by the processor, the data stored in the write location are corrupted, and the final result is randomly wrong. For these reasons, memory contention, i.e., the simultaneous access to the same memory position, must be avoided. That requires reordering the edge loop to prevent a node from appearing twice in the same work item group. Thus the edges are grouped, and the size of these groups depends on GPU features such as the number of processors and the maximum number of simultaneous work items per processor. An example of edge grouping is depicted in Figure 6.3, where the edge graph of figure 6.1 has been split in groups of up to 10 edges each. Three groups have 10 edges each, and three other groups have 7, 2 and 1 edge each.

When running on NVIDIA GPUs, all the edge groups that contain a number of edges equal to the maximum number of work items that can be executed simultaneously are placed in a unique edge group, since the process by which the GPU places the threads in the running line ensures that no overlapping will be produced between different groups. Following the grouping example of figure 6.3, if we presume that the GPU can process 10 threads simultaneously, then the first three edge groups could be packed in a single-edge group of 30 edges, while the small-edge groups must remain ungrouped. That strategy allows us to place roughly 90% of the edges in a single group, improving the GPU parallel efficiency substantially. This behaviour is believed to be hardware dependent and should be thoroughly checked for each hardware.

As a result of the grouping process, an additional array, called `groupEdges`, has been

Algorithm 6.3 Sequence of OpenCL kernel calls when the edge loop is split in several edge groups. `groupEdges` points, for each group, to the first edge index of the group.

```
...
for(group=0;group<nGroups;group++)
{
    edgeComputations.globalSize =
    groupEdges[group+1] - groupEdges[group];
    edgeComputations(...,group,groupEdges);
}
...
```

Algorithm 6.4 Modified OpenCL kernel version of Algorithm 6.2 used when the edge loop has been split into a number of edge groups. Only the lines that are modified are shown. `groupEdges` points, for each group, to the first edge index of the group.

```
__kernel void edgeComputations(...,
                                group,
                                groupEdges)
{
    edge = get_global_id(0)+groupEdges[group];
    ...
}
```

created. This array indicates, for each group, which is its first edge. Thus, the number of edges of a given group is `groupEdges[group + 1] - groupEdges[group]`, and the index of the first edge of the group is `groupEdges[group]`. After the grouping has been performed, we execute as many calls to the OpenCL kernel as edge groups have been found, as specified in algorithm 6.3. For each kernel, the total number of work items is the number of edges of the group. The OpenCL kernel of algorithm 6.2 is also slightly modified to make the edges within the group point to the correct global index. The resulting kernel is presented in algorithm 6.4, where only the lines that change with respect to algorithm 6.2 are written. Executing algorithm 6.3 on the GPU produces correct results.

The next question that needs to be raised is whether the kernel implementation is optimal for whatever computing platform. The limiting factor for all cases is the data transfer rate between memory and processors. The faster the data transfer, the faster the code will perform, since the speed at which the processors can process the data is actually much higher than the speed at which the data enters the processing units. The access pattern to the grid data is regular when the grid is structured, and the compiler knows in advance where to find them. This allows a fast data transfer between memory and

processor. For unstructured grids, however, the memory location of the edge nodes is not directly known by the compiler since the access to the data is controlled through an array of pointers, in our case the `edgeNode` array. This is referred to as indirect addressing in the literature. The access to the memory in these situations is much less efficient and hence some improvements must be introduced to avoid excessive performance degradation. The strategy may be different depending on whether the processor has cache memory or not. In processors without cache memory, or with a small-sized one, such as GPUs, the reordering technique previously presented is of little help. As stated in the introduction, GPUs base their superior performance in high data transfer rate between memory and processors. But the conditions for achieving such rate are usually very stringent, and certainly hard to meet if indirect memory addressing is used. In that case, it is crucial to minimize as much as possible the amount of data transferred from the global GPU memory to the local on-chip memory, performing as many operations as possible with variables that physically reside on the local memory.

Therefore, the parameter that influences performances the most is the ratio between the number of floating point operations (FLOPs) and the number of indirect reads or writes. Roughly speaking, the larger the number of FLOPs per indirect addressing, the greater the benefit expected when porting the code from CPUs to GPUs. That is why CFD codes that employ high-order discontinuous Galerkin discretisations ([18]), which require performing many FLOPs per grid node, have reported the largest speed-up when comparing GPU and CPU execution times. However, in codes where the amount of computation per cell is not as large, like the example we are presenting here, the speed-up can be seriously compromised if not enough attention is paid to this issue. An excellent review of the techniques employed to minimise the number of indirect addressings in edge-based solvers can be found in [58]. In order to better understand the importance of this optimisation, we present here two limit cases: the gradient loop and the flux loop.

6.4.1 Gradient evaluation

When the gradient evaluation of equation (2.33) is programmed according to the pseudo-code presented in algorithm 6.2, the number of indirect addressings per edge is 14, two for

Algorithm 6.5 OpenCL kernel for computing the fluxes looping over each node's neighbors.

```
__kernel void nodeComputations(...)
{
    node = get_global_id(0);
    data1 = ReadPointData(node);
    totalTerm = 0;
    for(neigh=0;neigh<neighbors;neigh++)
    {
        point2 = neighbor(neigh);
        data2 = ReadPointData(point2);
        term = F(data1,data2);
        totalTerm = totalTerm + term;
    }
    WritePointResult(node,totalTerm)
}
```

reading the variables, six for reading the three gradient components and six for updating them. However, the number of operations performed inside the loop is very small, hence the performance of the loop is completely controlled by the memory access. One simple way of reducing the number of indirect memory accesses is presented in algorithm 6.5. In this case, the loop is performed over grid nodes, and for each node, an inner loop over all the edges that surround it is executed. The number of indirect addressings per edge in this case has been reduced to one, for reading the variables of the neighbor node. Since the total number of edges is now doubled (each edge is processed twice, once per conforming node), the total number of indirect addressings has been divided by seven. When the loop is executed as an OpenCL kernel, the number of work items is the number of grid nodes. For each node we compute the contributions of the edges that share it, storing just the final result and not the intermediate ones as it was done in algorithm 6.2. The performance of algorithms 6.2 and 6.5 has been measured by implementing and executing both kernels in a NVIDIA GeForce GTX780Ti[71]. The kernel of algorithm 6.2 is about seven times slower than that of algorithm 6.5, which compares well with the reduction in the number of indirect addressings.

6.4.2 Convective and viscous fluxes evaluation

Although the evaluation procedure of the convective and viscous fluxes is conceptually analogous to that of the gradient, the situation is somehow different because the number of FLOPs per indirect addressing is much higher. Even though it has been shown that the number of indirect addressings can be reduced using the modified loop, it must be noted that each edge is processed twice; hence the number of FLOPs of the modified loop is twice as large as that of the original edge loop. This fact may counter-balance the positive effect of reducing the number of indirect addressings. Thus, in the case of the flux evaluation, the algorithm 6.2 kernel is now 1.25 times faster than the modified algorithm 6.5 kernel.

If the kernel total execution time is split in the time spent accessing and writing the data (t_{mem}) and the time spent doing operations (t_{op}), the total execution time for those kernels using algorithm 6.2 can be written as

$$t_E = t_{mem} + t_{op} \quad (6.6)$$

The memory access time can be further subdivided into direct and indirect access time, but since the indirect access takes much longer than the direct access, it is reasonable to assume that $t_{mem} \simeq t_{IA}$. Following that assumption, the execution time for those kernels written according to algorithm 6.5 is approximately:

$$t_N = \frac{2t_{mem}}{N_{IA}} + 2t_{op}$$

where N_{IA} is the number of indirect addressings per edge. These relations allow us to quantify both t_{mem} and t_{op} . It also shows that algorithm 6.2 kernels will perform better than algorithm 6.5 ones as long as $t_{op}/t_{mem} \geq 1 - 2/N_{IA}$.

6.4.3 Hybrid distributed-shared memory parallelism

When the solver is executed on multiple OpenCL devices the shared memory parallelism model is used within each device as explained in section 6.4, whereas the exchange of communication among GPUs is carried out using MPI. In that case, the data exchanged among processes must be first passed from the OpenCL device to its host CPU, and from

Figure 6.4: Sequence of executions and data communications to compute an edge loop.

the host CPU back to the OpenCL device once the communication has been completed. Since there is no possibility within OpenCL of a direct device to device communication, the minimisation of the time spent in the device-host communications becomes critical to obtain an acceptable parallel speed-up when the number of GPUs used to solve a given problem is increased. Two strategies are employed in order to reduce the time spent in transferring data between the GPU and the CPU. One is the use of the GPU pinned memory, which allows a faster data transfer rate between host and device by a priori telling the process where the portion of CPU and GPU memory to communicate physically resides within the CPU and GPU. The other is the use of non-blocking read and write operations in the CPU-to-GPU communications, an option that is allowed by either CUDA or OpenCL, hence overlapping the GPU computation and the CPU-GPU data transfer, analogously to the computation/communication overlapping between MPI processes.

The overlap of CPU-GPU communications and GPU computations, whose effect is smaller than the use of pinned memory, is analogous to the overlap explained in section 6.3 where the parallelisation of the edge loop computation for CPUs is described. The strategy was to group the edges of the mesh in two groups: one with the inner edges, that do not need data from neighboring domains to produce correct output, and the other with the frontier edges, that do need these data. Each of these groups must in turn be further subdivided

to produce correct output when running on GPUs, as explained in section 6.4. We take advantage of that second subdivision to insert the CPU to GPU communications between the end of the first group of inner edges, which accounts for 90% of the total number of edges, and the beginning of the rest of inner edge loops. The complete sequence of sub-group executions and data communications, schematically presented at Figure 6.4, is:

1. Data to be communicated via MPI are sent from the GPU to the CPU. This communication is non-blocking.
2. While the data are sent, initialize the edge loop variables.
3. Check that the data sending has been finished and start the non-blocking MPI communication to send the data to the neighbouring domains.
4. While the data are sent, compute the first inner edge group contributions.
5. Check that the MPI communication has finished and begin transferring the received data from the CPU to the GPU. This communication is non-blocking.
6. While the data are sent, compute the rest of the inner edge group contributions.
7. Check that the data have been received by the GPU and compute the parallel frontier edges contribution.

When the MPI distribution supports the overlap of communication and computation[34], this sequence minimises the extra cost of communicating data, either from the GPU to the CPU or between CPUs, by hiding the communication time behind the computation time.

6.5 Implicit Solver Parallelisation

The parallelisation of the implicit method is based on the parallelisation of the algorithms to solve the linear system. Firstly, the right-hand side, which is the residual, is computed in parallel by means of the procedures described in the previous sections. Next, the linear solver has to be parallelised. Depending on the algorithm, different techniques

Algorithm 6.6 Parallel Jacobi algorithm.

-
1. *Initial guess : $\Delta U = 0$*
 2. *for $k = 0, \dots, \text{linear iterations}$*
 3. *Compute $O = (L + U) \cdot \Delta U$ in parallel:*
 4. *Send ΔU of the frontier nodes from the GPU to the CPU*
 5. *Initialise O array to zero while ΔU data is sent.*
 6. *Start the MPI communication to send the ΔU to the neighbour domains.*
 7. *While ΔU at the frontiers is sent, compute the first inner edge group.*
 8. *Begin transferring data from the CPU to the GPU.*
 9. *While the data is transferred, compute the rest of the inner edge group.*
 10. *Compute the parallel frontier edges contribution.*
 11. *Update ΔU for every node:*
 12. $\Delta U_i = D_i^{-1} [R_i^n - O_i]$
 13. *end*
-

are required, but generally speaking, all the edge-loops that appear in the method are parallelised in the same way as the residual fluxes.

6.5.1 Jacobi method parallelisation

The Jacobi method depicted in Algorithm 4.1 provides wrong data of the variable ΔU in the frontier nodes, when it is executed in parallel using a shared memory paradigm. In these nodes, the correct values of ΔU must be communicated from the neighbouring domains.

Algorithm 6.6 shows the non-blocking parallel procedures explained in Section 6.4.3 applied to the Jacobi iteration. The computation of the product between the off-diagonal matrix and the unknown vector ΔU is performed through a loop over edges (line 3 in Algorithm 6.6) and the implementation in the GPU architecture is similar to that explained in section 6.4, obtaining a high degree of parallelism. Moreover, the line 12 in Algorithm 6.6 is executed through a loop of nodes with a high degree of parallelism as well. Overall, the implementation of the Jacobi solver both in GPUs and CPUs is extremely efficient. When the code is executed in multiple CPUs, steps 4 and 8 are skipped in Algorithm 6.6.

6.5.2 LU-SGS method parallelisation

The parallelisation of the either forward or backward sweeps of LU-SGS algorithm (Equations 4.2 and 4.3) is not as straightforward as the Jacobi solver parallelisation. When the upper triangular system is solved, all the nodes with lower index must have been solved before. There are several solutions to carry out the parallelisation of the LU-SGS algorithm, depending on whether a shared or distributed memory paradigm is used.

6.5.2.1 Shared memory LU-SGS parallelisation

When the memory is shared by all the processors, as it is the case for the GPUs, and many threads can be executed at the same time, then edge loops can be accelerated. As it has been mentioned, the parallel inversion of a generic sparse lower or upper triangular matrix is not straightforward. On the contrary than the Jacobi method, which is a pointwise single stage method independent of the order of execution, Gauss-Seidel type methods use the most recent available information during each sweep, and are not directly suitable for parallelisation. We will see that by means of permutations of the matrix elements, it is possible to create a block-decomposed matrix with only diagonal elements in the diagonal sub-matrices. The LU-SGS sweeps can be split into a block matrix-vector product and the inversion of large block-diagonal matrices. Each block operation can be parallelised, and the larger the size of the sub-block matrices, the higher the degree of parallelism.

One technique used for a long time to derive efficient algorithms is the multi-colouring, or graph colouring. The idea of the algorithm is to assign colours to the nodes such that any pair of adjacent nodes has different colours. The pseudo-code is depicted in Algorithm 6.7.

Reordering the nodes by groups of colours, the graph of the matrix is formed by blocks of nodes with the same colour, whereas the diagonal blocks are diagonal matrices, whose inversion is direct to parallelise, since every node is solved independently of the rest. An example of the original and multi-colouring matrix patterns is illustrated in Figure 6.5. The mesh with the original node numbering is depicted in Figure 6.5 (a). The non-zero

Algorithm 6.7 Multi-colouring.

```

for (i = 0; i < nnode; i++)
{
    color[i] = 0;
}
for (i = 0; i < nnode; i++)
{
    color[i] = min(k > 0 | k ≠ color[j]. ∀ j ∈ neighbors[i]);
}

```

Figure 6.5: (a) Two-dimensional mesh with periodic upper and lower boundaries, with the original node numbering and the colour code after applying the multi-colouring algorithm. Non-zero pattern of the original Jacobian matrix (b) and after applying the multi-coloring algorithm (c).

pattern of the matrix is shown in Figure 6.5 (b). Applying Algorithm 6.7, the resulting colour code is painted in Figure 6.5 (a). Finally, the nodes have been reordered by colours, resulting in the matrix pattern shown in Figure 6.5 (c). This mesh is formed by 13 nodes and the original numbering is the result of applying the reverse Cuthill-McKee ordering [30]. 4 colour levels are required in this particular example by the multi-colouring algorithm, the first two groups of colours have 5 nodes, the third group has 2 nodes, and the last only one node. If a hardware capable of managing 5 threads at once were used, the first block matrix could be inverted at once by executing the algorithm prescribed to update every node in separate thread. The performances obtained for the second block would be the same as for the first. A penalty in the speed-up appears when executing the two last blocks inversions, since all the available resources can not be used simultaneously. Nevertheless, if the time to execute a loop in serial is 13 units, the time to invert the block diagonal matrices, using 5 threads in a shared memory will be 4 units, yielding a speed-up factor of 3.25. As the problem size and the number of threads increase, this factor can be greatly improved.

When the linear system is reordered by colours, the implementation in a shared memory

Algorithm 6.8 Forward sweep for coloured blocks of the LU-SGS algorithm using an edge-based data structure.

1. *Solve* $(\mathbf{D} + \mathbf{L}) \cdot \Delta U^{1/2} = R_i^n$:
 2. *for* $i = 0, \dots, \#ncolors$
 3. *for* $j = 0, \dots, \#nodes[i]$
 4. *for* $k = 0, \dots, \#edges[j] : node_2 < node_1, node_1 = j$
 5. $L_j = L_j - \frac{1}{2} \left[\frac{\partial F_c(U_{node2}) n_{edgek}}{\partial U_{node2}} - |A_{edgek}| \right] \cdot \Delta U_{node2}^{1/2} \cdot \sigma_{edgek}$
 6. *end*
 7. $\Delta U_j^{1/2} = D_j^{-1} [R_j^n - L_j]$
 8. *end*
 9. *end*
-

Algorithm 6.9 Backward sweep for coloured blocks of the LU-SGS algorithm using an edge-based data structure.

1. *Solve* $(\mathbf{D} + \mathbf{U}) \cdot \Delta U = \mathbf{D} \cdot \Delta U_i^{1/2}$:
 2. *for* $i = \#ncolors, \dots, 0$
 3. *for* $j = \#nodes[i], \dots, 0$
 4. *for* $k = 0, \dots, \#edges[j] : node_2 > node_1, node_1 = j$
 5. $\mathbf{U}_j = \mathbf{U}_j - \frac{1}{2} \left[\frac{\partial F_c(U_{node2}) n_{edgek}}{\partial U_{node2}} - |A_{edgek}| \right] \cdot \Delta U_{node2} \cdot \sigma_{edgek}$
 6. *end*
 7. $\Delta U_j = \Delta U_j^{1/2} - D_j^{-1} \mathbf{U}_j$
 8. *end*
 9. *end*
-

architecture must be revisited in order to optimize the performances of the algorithm. Indeed, the linear system to be solved is of the form sketched in Figure 6.6, and then Algorithms 4.2 and 4.3 can be rewritten as Algorithms 6.8 and 6.9 respectively. Each coloured block color is solved by performing a node loop over the nodes with the same colour, which can be parallelised because of the data updated with this loop is $\Delta U_j^{1/2}$ and all the nodes with the index j are not coincident with the index $node_2$ used within the loop. The efficiency of this algorithm is deteriorated in the GPU since the memory access to the Roe's matrix $|A_{edgek}|_{5 \times 5}$ is very expensive. Indeed, it consists in an indirect reading because the position of the $node_2$ is not contained in the memory processor. As the lower and upper sweeps are performed through block coloured matrices, the product between either the lower block matrices or the upper block matrices, and the increment of the conservative variables can be done in a single step, storing the result in a global array, followed by the straightforward inversion of the block-colour diagonal matrix times the residual minus the block-color off-diagonal matrices times the increment of conservative variables previously computed. The result is sketched in Algorithms 6.10 and 6.11.

Algorithm 6.10 Forward sweep for coloured blocks LU-SGS algorithm using an edge-based data structure with block inversion in a different loop.

1. *Solve* $(\mathbf{D} + \mathbf{L}) \cdot \Delta U^{1/2} = R_i^n$:
 2. *for* $i = 0, \dots, \#ncolors$
 3. *for* $j = 0, \dots, \#edges[i]$ $node_2 < node_1$
 4. $L_{node1} = L_i - \frac{1}{2} \left[\frac{\partial F_c(U_{node2}) n_{edgej}}{\partial U_{node2}} - |A_{edgej}| \right] \cdot \Delta U_{node2}^{1/2} \cdot \sigma_{edgej}$
 5. *end*
 6. *for* $j = 0, \dots, \#nnodes[i]$
 7. $\Delta U_j^{1/2} = D_j^{-1} [R_j^n - L_j]$
 8. *end*
 9. *end*
-

Algorithm 6.11 Backward sweep for coloured blocks LU-SGS algorithm using an edge-based data structure with block inversion in a different loop.

1. *Solve* $(\mathbf{D} + \mathbf{U}) \cdot \Delta U = \mathbf{D} \cdot \Delta U_i^{1/2}$:
 2. *for* $i = \#ncolors, \dots, 0$
 3. *for* $j = 0, \dots, \#edges[i] : node_2 > node_1$
 4. $\mathbf{U}_{node1} = \mathbf{U}_{node1} - \frac{1}{2} \left[\frac{\partial F_c(U_{node2}) n_{edgej}}{\partial U_{node2}} - |A_{edgej}| \right] \cdot \Delta U_{node2} \cdot \sigma_{edgej}$
 5. *end*
 6. *for* $j = 0, \dots, \#nnodes[i]$
 7. $\Delta U_j = \Delta U_j^{1/2} - D_j^{-1} \mathbf{U}_j$
 8. *end*
 9. *end*
-

Firstly, within either the forward or backward sweeps, the lower sub-block triangular matrices times the vector of unknowns can be performed using an edge loop with a high degree of parallelism (lines 3-5 of Algorithms 6.10 and 6.11). Later on, the inversion of the sub-block diagonal can be done into a separated loop over nodes with a high degree of parallelism as well (lines 6-8 of Algorithms 6.10 and 6.11). Separating the loops in this manner, the memory access to the matrix $|A_{edgek}|_{5 \times 5}$ is inexpensive because it is contained in the processor. It is taken for granted that there is an indirect reading for the variable U_{node2} , but anyway is much less expensive than reading 25 elements. The penalty of this algorithm version is that the memory requirements are higher than that of the original one. The over-storage of the algorithm is justified as long as the global convergence time is improved, and the memory of the hardware is not a constraint.

Similar to the edge loop implementation in a share memory platform, the edges of the same colour must be grouped in such a way that two edges of the same group are not connected with the same node of the diagonal node. As a result of the grouping process, an additional

Figure 6.6: Block-decomposed matrix for LU-SGS solver.

array of dimension 2, called `groupEdgesColor`, has been created. This array yields, for each group of each color, which is its first edge. Thus, the number of edges of a given group of the color " c " is `groupEdgesColor[c][group + 1] - groupEdgesColor[c][group]`, and the index of the first edge of the group is `groupEdgesColor[c][group]`. After the grouping has been performed, we execute as many calls to the OpenCL kernel as edge groups have been found, as specified in algorithms 6.10 and 6.11. The main drawback of this grouping is that usually there are many edge sub-groups of small size for the smallest colour groups, leading to a penalty in the scalability. Nevertheless, when the size of the mesh is large enough, a significant acceleration is achieved, although not as large as for Jacobi solver, which is better suited for parallelisation.

6.5.2.2 LU-SGS distributed memory parallelisation

The parallelisation of the LU-SGS when using distributed memory is simplified by performing a mixture between the Jacobi and LU-SGS solver in the frontier nodes. That is, the nodes that has not been updated by the forward or the backward sweeps, are used with the value of the current status. This formulation has been developed in [62].

6.5.3 GMRES method parallelisation

The parallelisation of GMRES solver is based on the computation of the residual and matrix-vector products in parallel. This has been covered in the previous sections.

Additionally, the vector dot product needs to be parallelised as well, which is used to compute the norm of the residual and to generate the elements of the Hessenberg matrix.

In shared memory platforms, the dot product is parallelised in four steps:

1. Each thread sums two elements of the global array and puts the result in a local array `wklocal` which is only visible by the threads of each group.
2. The `wklocal` elements are further reduced by summing up pairs, halving the number of threads each time.
3. The final reduced result for each thread group, `wklocal[0]`, is kept in a global memory array, that has as many elements as thread groups.
4. This array is extracted to the cpu and the final reduction is done there, serially.

When multiple processors are used, the dot product is completed calling to the `MPI_Allreduce` function.

It is important to highlight that for distributed memory parallelism, all the processes have the Hessenberg matrix stored and the solution of the least square problem (line 31 of 4.4) is performed by all of them in the CPU, since typically this matrix has a small size, of the order of 20×20 . This way, the master process avoids broadcasting the solution of the least square problem to the rest of the processes.

Chapter 7

Numerical Results

7.1	Introduction	109
7.2	Linear solvers results	110
7.3	Multi-core and many-core performances of the baseline solver	117
7.4	Multi-core and many-core performances of the implicit solver	125
7.5	Test Cavity Flows	129
7.6	ADTURB Vane	134

7.1 Introduction

In this chapter the performances of the new implicit methods implemented in the baseline code, Mu^2s^2T , are presented. The reference code is an explicit five-stage Runge-Kutta scheme [56], where the artificial viscosity and the viscous terms are evaluated only in three stages. The JST and MUSCL schemes are used to discretise in space the equations. Either the scalar or matricial Roe's fluxes are implemented for the artificial viscosity terms. Local time step, block-Jacobi and multi-grid techniques are used to accelerate the convergence ([40]). Therefore, most of the implicit solver results are bench-marked against its explicit counterpart in order to assess the improvement in terms of algorithm implementation. Moreover, the numerical efficiency of the different linear solvers are

studied. A parametric study of the CFL number, the number of linear iterations or the convergence level of the linear system has also been conducted.

The code has been systematically executed both in CPUs and GPUs, comparing the performances of the different linear solvers in these two different platforms. The porting from the CPU to the GPU does not deliver consistent results for all the algorithms studied, and hence an in-depth assessment of the implicit method in GPUs is carried out.

Additionally, the solver is validated with a wide range of test cases available in the CFD community. First, simple inviscid cases are used to validate the implicit algorithms. Next, several viscous test cases are used to validate the numerical fluxes and the turbulence model. Finally, the validation of the code in turbomachinery applications is completed with the experimental data from a low-speed turbine rig.

7.2 Linear solvers results

The baseline case consists of an inviscid subsonic flow in a channel with a circular bump on the lower wall, which is known as the Ni's test case. The dimensions and a sketch of the channel are shown in Figure 7.1. Since the flow is inviscid, there is not cell clustering near the walls, and the mesh has been generated by means of the Constrained Delaunay Triangulation technique ([89]). The inlet Mach number is set to 0.5. This baseline case has been selected because there has been widely used in the literature (see for instance Löhner et al. [60]) to describe the performances of the GMRES+LU-SGS algorithm. In this way, the numerical algorithms implemented here can be bench-marked against another well-reputed CFD code with the same algorithm.

The convergence history for the GMRES+LU-SGS scheme presented by Löhner et al. [60] for this particular case of the subsonic channel bump is compared with the implicit methods implemented in Mu^2s^2T code. The convergence history is shown in Figure 7.2. The stopping criterion for the linear system is a norm reduction of 0.1. The Krylov subspace dimension for GMRES is set to 10. The CFL number used in Mu^2s^2T simulations are the optimum ones, whereas Löhner et al. [60] fix the CFL number to 10,000.

Figure 7.1: Surface mesh and dimensions of the circular bump channel test case (a) and Mach number colour contours (b) (13630 nodes and 68132 cells).

Figure 7.2: Convergence history for inviscid subsonic channel flow. Comparison between Mu^2s^2T code and Löhner et al. [60].

Although non-identical convergence histories are obtained between Mu^2s^2T and Löhner et al. [60], the number of iterations to reach the steady state is very similar. The differences likely come from several factors: the mesh is not identical since Löhner et al. use an advancing front technique whereas the mesh used for Mu^2s^2T is generated by the Constrained Delaunay Triangulation technique, and implementation issues related with the boundary conditions which are not easily found in the literature. Nevertheless, the speedup due to the algorithm is the same for both codes. For instance, to reduce the norm five orders of magnitude, LU-SGS solver converges in 4000 iterations whereas GMRES+LU-SGS converges in 150 iterations.

7.2.1 Effect of the Linear Solver Parameters

So far, the benchmarking of the implicit algorithms has been done using a well-documented test case in the literature. Before moving on, it is worthy to assess the impact on the convergence of the different linear solver parameters. This effect has been theoretically studied in subsection 5.3.4 for the 2D linear advection equation, where the impact of the CFL and the linear solver convergence level on the global convergence in terms of iterations and computational time have been assessed. Here the same study is carried out for the 3D Euler equations. The effect of the CFL on the convergence of different linear solvers is studied. Linear systems resulting from large CFL numbers are stiffer than those coming out from small CFL numbers, since they are less diagonal dominant, and therefore the number of inner iterations to reach a certain level of convergence is larger. On the other hand the number of time steps is reduced, and therefore an optimum value for the CFL number exists.

Moreover, the convergence rate of the linear solvers varies depending on the iteration phase. Indeed, if the initial solution is far from the exact solution, during the first steps there are more reflexions and is more difficult to converge the linear system. Nevertheless after a few time steps the number of iterations required to solve the linear system of equations is nearly constant.

Figure 7.3: Convergence history of the linear system for two CFL numbers and different linear solvers. (Inviscid circular bump channel).

Figure shows 7.3 the effect of the CFL number on the convergence history of the linear system for different linear solvers. The larger the CFL number, the more iterations are needed to converge. This is a general trend for all the linear solvers. Clearly, the Jacobi solver spends much more iterations to converge than the preconditioned GMRES, needing roughly 5 times more iterations. The GMRES solver without preconditioning stagnates, and it is not able to converge the linear system. Comparing the two GMRES preconditioners, the LU-SGS needs less number of iterations than the Jacobi preconditioning, of the order 1.5 for low CFL numbers, and 1.8 for large CFL numbers. GMRES+Jacobi solver is formed by running two iterations of the preconditioner. As the number of the Jacobi iterations to precondition GMRES is increased, the number of iterations to converge the linear system is reduced, as it is shown in Figure 7.4 (a). The preconditioning becomes very expensive when many iterations of the Jacobi solver are performed, therefore the reduction in iterations to converge the linear system is not made up by the extra cost of the preconditioning and then the global time is not improved. It is concluded that the most efficient preconditioning is obtained with a single iteration of the preconditioner solver, as can be seen in Figure 7.4 (b). Furthermore, GMRES+LU-SGS

Figure 7.4: Convergence history of the linear system in terms of iterations (a) and CPU time (b) for different linear solvers, preconditioners and number of iterations of the GMRES+Jacobi solver for the inviscid circular bump channel test case.

and Jacobi linear solvers have been included in Figures 7.4 (a) and (b), where it can be seen that the most efficient solver is the former.

The LU-SGS preconditioning effectiveness can be modified due to the multi-colouring reordering algorithm, in order to allow the parallelisation of the matrix inversion in many-core platforms. Here only the effect of the nodes reordering on the performance of the GMRES+LU-SGS scheme is investigated. The matrix patterns before and after applying the multi-colouring algorithm to this case are shown in Figure 7.5. The coloured sub-block matrices are clearly seen, as well as the diagonal sub-blocks, which are diagonal matrices. Ten groups of nodes are obtained in which the number of nodes varies from 2440 in the largest group to 5 in the smallest group. The resulting linear system needs more iterations to converge than the original linear system, as it is shown in Figure 7.6. The LU-SGS-GMRES scheme using multi-colouring ordering is 20% and 40% slower for CFL=10 and CFL=100, respectively, than the original ordering.

Regarding the GMRES solver, there is an additional parameter to be set, the Krylov subspace dimension. The convergence histories of GMRES+LU-SGS solver for different Krylov subspace dimensions is shown in Figure 7.7 (a). Many iterations are needed to converge the linear system for low Krylov subspace dimensions. For large Krylov subspace

Figure 7.5: Original matrix pattern of the linear system with the Cuthill-McKee ordering (a) and after applying multi-colouring reordering (b).

Figure 7.6: Effect of the multi-colouring reordering on the convergence history of GMRES+LU-SGS solver. Dimension of the Krylov subspace equals to 10.

Figure 7.7: Effect of the Krylov subspace dimension on the convergence history of GMRES+LU-SGS solver (a) and CPU time (b) for the inviscid circular bump channel test case.

dimensions, a stagnation is presented in the middle of the convergence. Therefore, there is an optimum number of dimensions to optimise the residual for GMRES solver, which is between 10 and 20 directions. In terms of CPU time to converge the linear system, the optimum oscillates as well between 10 and 20 directions. This conclusion is quite general and it is illustrated in Figure 7.7 (b) for the circular bump test case.

The complete convergence of the linearised equations is not necessary to converge the CFD simulation. Therefore, the number of the inner iterations can be prescribed, or alternatively, the convergence tolerance of the linear system fixed to a certain level. In Figure 5.6, it was concluded that the computational cost is reduced as either the linear iterations are lowered for a constant CFL number or the CFL is increased for a constant number of the inner iterations. This effectively means that the CFL should be as large as possible while the number of iterations of the linear system should be as low as possible. Different solvers behave in different ways depending on the convergence level or the number of inner iterations. Figure 5.6 displays a similar behaviour for a 3D simulation. Note that, in Figure 7.4 (b), there is a clear speed-up using the GMRES+LU-SGS solver if the linear solver is completely converged, but if the convergence level is set to 0.1, for instance, this speed-up is not so significant.

Running a full factorial matrix of cases varying the CFL number and the linear iterations of the linear system, the total computational time to converge the CFD simulation

Figure 7.8: Effect of the CFL number and inner iterations of the linear solver on the computational time to converge the simulation. Comparison between the implicit schemes (Jacobi linear solver (a) and GMRES+LU-SGS (b)) and explicit RK5 scheme for the circular bump channel test.

is represented in Figure 7.8, separating a couple of linear solvers: Jacobi (a) and GMRES+LU-SGS (b). The explicit integration method is also included in both plots in order to compare the performances of the implicit methods against their reference explicit counterpart. There is a range of CFLs and inner iterations where an optimum computational time is seen. The best performances of the Jacobi solver are obtained for CFL numbers between 20 and 50 using between 10 and 50 inner iterations. For the optimum range of parameters the simulation converges 6 times faster than the explicit reference solver. On the other hand, the GMRES+LU-SGS solver presents the best performances for a CFL number about 50 and 10 inner iterations, showing a speedup of 10 with respect the fully explicit scheme. Note that in this comparison, multi-grid preconditioning has not been considered.

7.3 Multi-core and many-core performances of the baseline solver

The code efficiency in multi-core and many-core devices is characterised by running two different benchmarks. The first one aims at characterising its performances when running on a single GPU. In that case, the influence of the problem size will be presented, as well

Figure 7.9: Influence of the number of grid points in the performance of the Mu^2s^2T solver for the NVIDIA GeForce GTX780Ti GPU and a single CPU core Intel Xeon E5-1620 @ 3.6 GHz.

as the influence of the GPU model. For this benchmark a series of grids to discretise the bump channel (see Figure 7.1) has been used. The second benchmark measures the code performance when solving the T106C airfoil[97] meshed with one million points on a cluster of GPUs. For both scenarios the GPU model that will be analysed is the NVIDIA GeForce GTX780Ti[71], with 3GB memory and 2880 GPU cores at 875 MHz. The cluster has 16 nodes networked using a 100Gb/s Infiniband network. Each node has two Intel Xeon E5-2650 @ 2.60GHz CPUs, with 8 cores each [45], and hosts 8 NVIDIA GeForce GTX780Ti GPUs. The CPUs are not intended for simulation purposes, since they are quite slow compared with the high-end Xeon CPUs, but serve well to control the GPUs used to perform the simulation and to drive the data exchange between neighbouring parallel frontiers during the simulation.

7.3.1 Single GPU benchmark

The first test compares the performance of the solver when simulating the flow in the bump channel with grids of varying density, from a coarse mesh of 720 points to a fine mesh of 1.6 million points. The purpose of this test is to quantify the performance degradation of the GPU as the size of the problem decreases. Even though the GPU is treated as a black box, it must be recalled that it consists of thousands of processors executing code in parallel. And as any other parallel machine its parallel scalability depends on a number of parameters, the main one being the problem size. By quantifying that parallel behaviour

we can determine the processor efficiency, C_{eff} , of equation (6.2), which will in turn be useful to understand the parallel scalability of the solver when executed on multiple GPUs. To determine C_{eff} , the number of mesh points is divided by the time needed to execute a fixed number of solver iterations, t_N . The resulting number is non-dimensionalised with the simulation time in the finest grid.

$$C_{eff}(N) = \frac{N}{N_{finest}} \frac{t_{finest}}{t_N} \quad (7.1)$$

If the parallel scalability of the GPU were perfect, $t_N \propto N$, and C_{eff} would be equal to 1, no matter the size of the case. However, the actual behaviour, presented in Figure 7.9, differs substantially from the ideal case, as C_{eff} strongly depends on the size of the case for $N_p < 10^6$. Thus, for $N_p = 10^5$, the efficiency is $C_{eff} \simeq 0.86$, whilst for the coarsest case of $N_p = 720$, the solver is 40 times less efficient than the baseline case of $N_p = 1.6 \times 10^6$. Therefore, for an efficient use of the GPU, the largest possible mesh should be used. Additionally, the efficiency C_{eff} for a single CPU core is plotted in Figure 7.9, where the opposite effect with respect the GPU is presented, i.e., the efficiency increases when the problem size decreases due to the reduction of the cache misses as explained in section 6.3.

It is also useful to compare the chosen GPU with other GPU models to rank their relative computing capability, and to understand which GPU characteristics affect solver performances the most. Figure 7.10 (left) shows the influence of the GPU model on the asymptotic execution time of the solver. The GPU models are ordered according to their release date, being the Quadro FX3800 released in 2009 and the GeForce GTX1080 in 2016. During these seven years the execution time has been divided by a factor of 11. That reduction is roughly proportional to the increase in GPU memory bandwidth (see Fig. 7.10 right). Indeed, it has been previously shown that the performance of some of the most time-consuming parts of the solver is often controlled by the access to the GPU memory. The faster the memory access is, the better the performances are. However, a complex Navier-Stokes solver like ours contains loops with a high variation of Flops per accessed byte and therefore a balanced increase of the bandwidth and the processor speed is needed to speed-up the solver.

Figure 7.10: Solver performance in different graphics cards.

Figure 7.10 (left) contains performance data obtained from commodity cards (Quadro and GeForce product lines) and cards specifically designed for scientific computing (Tesla product line). It can be seen that the performance of both are comparable, at least for single precision arithmetic, since in many occasions the chip-sets mounted in video-game and scientific computing cards are the same. The main difference between both is the reliability and the selling cost. In our GPU clusters, we have performed endurance tests to the graphics cards to discard the most faulty hardware and implemented automatic mechanisms to relaunch the jobs if a hardware error causes a job failure.

7.3.2 Network characterization

According to Equation (6.2), the network is characterised by its bandwidth, BW , which is determined by measuring the time, t_c , at which a given message of size M is transferred from one MPI process to another. If the network latency is neglected, $t_c \simeq M/BW$. This test measures the bandwidth of bidirectional data transfers of varying sizes between two MPI processes. The test has been taken from the MVAPICH2-1.8 implementation micro-benchmarks ([74]) and adapted to pass data from one GPU to another using OpenCL. This MPI implementation supports direct GPU-to-GPU communication if the CUDA language is used, i.e., the communication of arrays physically residing in the GPU memory can be performed without having to explicitly communicate them to the CPU before sending

Figure 7.11: Influence of the message size on the MPI communication data transfer rate. Communications between CPUs and between GPUs are presented.

them via MPI. That behavior is not supported in OpenCL, therefore the GPU-to-CPU and CPU-to-GPU communications are written explicitly using pinned memory to maximize the GPU-to-CPU and CPU-to-GPU bandwidths, as stated above.

Figure 7.11 shows a number of interesting results. First, the resulting bandwidth of the CUDA direct GPU-to-GPU communication is almost the same as that implemented in OpenCL using pinned memory; therefore, the penalty of using OpenCL instead of CUDA in terms of effective communication efficiency is very small. Second, for messages smaller than 10^5 bytes the GPU-to-GPU bandwidth is an order of magnitude lower than the CPU-to-CPU bandwidth. However, for large enough messages the bandwidth loss associated with the additional GPU-to-CPU and CPU-to-GPU data transfer is not so important: $BW_{CPU \leftrightarrow CPU} \simeq 5000 \text{ MB/s}$ whereas $BW_{GPU \leftrightarrow GPU} \simeq 3000 \text{ MB/s}$. Finally, for the 3D T106C mesh, decomposed in a number of parallel sub-domains ranging from 2 to 64. The size of the parallel frontiers data messages is large enough to be on the region of asymptotic behaviour of the bandwidth curve; hence, the overhead due to the use of the GPUs is minimum,

7.3.3 Multiple GPU Benchmark

Once the processor efficiency, C_{eff} , the message size, M , and the network bandwidth, BW , have been characterised, two terms of Equation (6.2) remain to be determined to fully understand the parallel speed-up behaviour. The first one is the computation

Figure 7.12: Speed-up curves of the Mu^2s^2T solver for the T106C test case.

overhead, C_{oh} , which is usually small compared to C_{eff} , hence a value $C_{oh} = 1$ can be assumed with little error. The second term is the load unbalance penalty, Δt_u , that can only be determined once the discrete domain has been sub-divided. However, as it will be shown below, its effect is small, although not as small as C_{oh} , and can also be neglected without compromising the understanding of the speed-up curve.

The speed-up curve for the T106C benchmark case is presented in figure 7.12. It has been obtained running a fixed number of Runge-Kutta iterations, without multigrid, splitting the initial one million point mesh into up to 64 sub-domains. It can be observed that the actual speed-up quickly differs from the ideal even for very low number of MPI processes. For $P = 16$, the actual speed-up is 7.3, and for $P = 64$ it is only 11.7. Three additional curves have been presented in order to better understand the sources of the speed-up degradation as the number of sub-domains is increased. The dotted curve with triangle symbols represents P/C_{oh} , i.e., how the ideal speed-up degrades due to the computational overhead associated with computing the frontier edge contributions twice. This proves that for this range of P , the computation overhead has little influence in the speed-up degradation (for the $P = 64$ case, $C_{oh} = 1.041$). If we add the effect of the GPU efficiency C_{eff} (shown in Fig. 7.9) we obtain the dashed curve with diamonds. $P C_{eff}/C_{oh}$ is the maximum achievable parallel speed-up assuming perfect load balancing and zero communication and computational overheads, taking into account the computational overhead and the GPU efficiency degradation as the size of the sub-domain decreases. By adding the load imbalance of the current case to the $P C_{eff}/C_{oh}$ curve we obtain

Figure 7.13: Speed-up curves of the Mu^2s^2T solver for problems of different sizes.

the curve with the square symbols, which is the maximum achievable speed-up given the actual sizes of the sub-domains. The penalty due to that imbalance is small for all P . The communication time t_c is accountable for the difference between the previous curve (diamonds) and the actual speed-up (circles). The loss of parallel speed-up due to the communications is noticeable, but not as large as the loss induced by the GPU efficiency C_{eff} , which is the main driver for the speed-up degradation.

Figure 7.13 compares the speed-up curve obtained with the one million point mesh of Figure 7.12 with another curve obtained by solving a nine million point mesh. As the size of the case increases, we move to higher values of C_{eff} and therefore the speed-up is significantly improved.

7.3.4 Comparison against a CPU cluster

Finally, Figure 7.14 compares the speed-up of the benchmark case run on the CPUs of the GPU cluster with the one obtained when running on GPUs. The curves has been shifted to reflect the speed-up obtained when running with a single GPU instead of a single CPU core. The available CPU clusters do not feature especially powerful processors, since they are just meant to control the GPUs, but in any case the speed-up when running in a single GPU instead of a single CPU core is 52. However, a fair comparison is to consider the CPU as a whole, just as we did for the GPU and its 2304 cores. Therefore, the GPU speed-up against the $P = 8$ time of the CPU curve is 7.9. This result is hard to match,

Figure 7.14: Execution time curves of the Mu^2s^2T solver for the T106C benchmark case.

even using more powerful CPUs and proves the fact that the use of GPUs is an excellent mechanism of speeding up the resolution of the RANS equations. In fact, a single GPU is slightly faster than 64 CPU cores, therefore enabling substantial savings both in terms of hardware acquisition cost and energy consumption. For a rough estimation of the savings, let us assume that the computational power of one single GPU is equal to that of 64 CPU cores. Since each node of the cluster has 8 GPUs, each node of the cluster would have the equivalent computational power of 512 CPU cores, or 64 CPUs. On the one hand, the price of the Intel Xeon in 2013 was 816 Euros and its maximum power consumption is 95W, then 64 CPUs cost 52,224 (neglecting the cost of the network for the sake of clarity) and consume 6080W. On the other hand, the price of the GeForce GTX 780Ti, also in 2013, was 512 Euros and its maximum power consumption is 250W. Therefore, for a roughly equivalent computational power, the cost of eight GPUs plus one CPU that hosts them is 4,912 Euros and the maximum power consumption 4095W. For approximately the same computational power, the GPU cluster is 10.6 times cheaper and consumes 33% less energy.

It is important to highlight that if the same economic study had been conducted using GPUs designed for scientific computing, such as the Tesla product line, the attractiveness of the solution is much less than in the current approach, becoming the comparison between CPU-based and GPU-based clusters much more balanced.

7.4 Multi-core and many-core performances of the implicit solver

The implementation and performances on GPUs devices of the baseline solver have been presented in the previous section. Analogously, the code performance on multi-core and many-core devices for the new implicit algorithms is presented in the next sub-sections. Particularly, the efficiency of the Jacobi and GMRES+LU-SGS linear solvers are detailed and compared against the baseline explicit solver.

7.4.1 Single GPU benchmark

As it has been described in the previous section, when a simulation is run in a many-core device, the size of the problem has a significant impact on the performances of the solver. Indeed, for small size problems the ratio between computation time and thread managing decreases strongly. Whereas running on a single CPU, small size problems present better efficiency because of the cache misses are minimised. For this study, a faster CPU than for the baseline solver has been used: Intel Xeon E5-1620 @ 3.6 GHz, which is approximately twice faster than Intel Xeon E5-2650 @ 2.60GHz.

The effect of the number of grid points on the computational efficiency, C_{eff} , for both, CPUs and GPUs devices is shown in Figure 7.15(a). The reference point to non-dimensionalise the time is $N_P \simeq 1.6 \times 10^6$. Moreover the data for three different solvers, without using multi-grid preconditioning, are shown: explicit RK5, implicit Jacobi and implicit GMRES+LU-SGS. Jacobi linear solver presents basically the same deterioration in the efficiency with the problem size as the explicit baseline solver, whereas this deterioration starts for GMRES+LU-SGS solver at larger number of grid points. Thus, for $N_p = 10^5$, the efficiency for Jacobi and explicit solvers is about 0.86, whilst for GMRES+LU-SGS solver is 0.69. For the coarsest case of $N_p = 720$, the Jacobi and explicit solvers are 40 times less efficient than the baseline case of $N_p = 1.6 \times 10^6$, whereas GMRES+LU-SGS is 109 times less efficient. Considering the CPU hardware, GMRES+LU-SGS solver presents much better efficiency than the other solvers for coarse grid cases, up to $N_p = 7 \times 10^3$.

Figure 7.15: Influence of the number of grid points in the performance for two processors (CPU: Intel Xeon E5-1620 @ 3.6 GHz, GPU: NVIDIA GeForce GTX780Ti) and different schemes: Explicit RK5, Jacobi with 10 linear iterations, and GMRES+LU-SGS with Krylov subspace dimension equal to 10.

Comparing the speed-up between a single CPU core and a single GPU (see Figure 7.15(b)), the implicit Jacobi solver presents the best acceleration, being 41 times faster in a GPU than in CPU. This result improves substantially the speed-up of the baseline explicit scheme, where the acceleration factor of the GPU with respect the GPU is approximately 26. Finally, the GMRES+LU-SGS scheme is not as scalable as the Jacobi solver, due to the fact that after the multi-colouring block decomposition, there are a few blocks with small size which are easy to parallelise efficiently. Nevertheless better speed-up than the baseline explicit scheme is achieved with the GMRES+LU-SGS scheme. However, the speed-up of the GMRES+LU-SGS linear solver, for small problem sizes, is decreasing at higher rate than the other schemes. Indeed, in Figure 7.16, the number of colour blocks and the nodes per block are shown after applying multi-colouring algorithm. Ten groups of colours are typically obtained. The first six groups represent the 90% of the total size, and every group is approximately one order of magnitude smaller than the total size (between 10% and 20% of the total number of nodes), and then the deterioration of the efficiency with the problem size, C_{eff} , can be detected for a number of points 20% larger than in the other solvers. The rest of the groups, which represents about 10% of the total size, do not scale properly, but their relative weight, in terms of computational time, is small compared with the total time to solve the linear system.

Figure 7.16: Colour blocks groups after multi-colouring reordering for different problem sizes of the inviscid bump channel.

Figure 7.17: Speed-up curves of the Mu^2s^2T code for the bump test case on different platforms.

7.4.2 Multiple GPU Benchmark

Analogously to the results presented in sub-section 7.3.3 for the baseline explicit solver, the performances on multiple GPUs of the implicit solver developed in this work are described here. The speed-up curves for the bump channel case of 1 million points are represented in Figure 7.17. They have been obtained running without multigrid a fixed number of iterations: 5 for the explicit Runge-Kutta, 10 for the implicit Jacobi solver and 5 for GMRES+LU-SGS solver. Additionally, the latter solver has been run with a dimension of the Krylov subspace equal to 10. Both implicit solvers, Jacobi and GMRES+LU-SGS, present lower speed-ups than their explicit counterpart. Up to 16 MPI processes, the

Jacobi solver presents similar performances than the explicit baseline solver; however, for $P = 64$, the Jacobi solver speed-up is 15.5, whereas the explicit speed-up is 18.6. The penalty in terms of speed-up for the Jacobi solver is due to the communications, because 10 inner linear iterations are performed per global iteration, and hence more communications per iteration are taking place, adding a penalty in the efficiency. If only five inner iterations for the Jacobi solver are performed, the explicit speed-up curve is recovered. Moreover, the GMRES+LU-SGS solver speed-up is deteriorated significantly for the entire range of MPI processes, because the efficiency, C_{eff} , is lower than in the other solvers, as it has been presented in Figure 7.15(a). For instance, when $P = 64$, the speed-up is only 10.6. Additionally, the amount of communications per iteration is much higher than in the inner iterations of the other solvers. Indeed, if the Krylov subspace dimension is fixed to 10, and the number of inner iterations to five, 50 evaluations of a fluxes edge-loop kernel evaluations are performed, plus its corresponding communications, which means that the term Δt_u is higher than in the other solvers.

7.4.3 Comparison against a CPU cluster

Finally, Figure 7.18 compares the speed-up of the benchmark case run on the CPUs of the GPU cluster with the one obtained when running on GPUs. The effect on the speed-up of the implicit solvers against their explicit counterpart is presented as well. In order to reflect the speed-up obtained when running with a single GPU instead of using a single CPU core, the corresponding curves of speed-up running on GPUs are scaled with the factors between 1-GPU-to-1-CPU described in Figure 7.15(b). Note that Figure 7.18 does not include the speed-ups due to the algorithms, only compares the relative performances of GPUs and CPUs of the different solvers. Without considering algorithm speed-ups, the Jacobi solver presents the largest speed-up when moving the simulation from the CPU cluster to the GPU cluster. For example, running the case in 16 GPUs with the Jacobi solver, would take the same time than running with 64 GPUs with the baseline explicit solver. Conversely, the GMRES+LU-SGS solver does not improve the performances of the baseline explicit solver. For instance, running the case in 64 GPUs with the GMRES+LU-SGS solver, would take the same time than running with 32 GPUs

Figure 7.18: Speed-up curves of the Mu^2s^2T code for the bump benchmark case on different platforms.

the explicit solver. This situation is not presented in practice, because one million point case would never be partitioned in 64 domains in a highly demanded and busy cluster.

7.5 Test Cavity Flows

In this section, simplified cavities that are well known in the literature are simulated in order to validate the numerical implementation of the solver as well as to assess the performances of the new implicit solver in cavity flows, where typically their explicit counterpart presents some stiffness. The motivation for this is the application of the implicit solve to cavity flows, that somehow triggered all the work presented here. The following chapter will be actually focused in the application of this solver to realistic complex problems, including under-platform cavities.

7.5.1 Lid-Driven Cavity

The lid-driven cavity has been widely used in the past as a test for incompressible flows by many authors. The results given by [91] are taken as reference to compare with Mu^2s^2T code. The Lid-driven cavity problem consists of a closed-square cavity of side L , where the top surface moves at constant velocity U . Four different Reynolds numbers are simulated: 400, 1000, 3200 and 5000, where the Reynolds number is defined as:

$$Re = \frac{\rho UL}{\mu}$$

where ρ is the density of the fluid, U the velocity of the lid, L the length of cavity side and μ the dynamic viscosity. To cover all the Reynolds numbers, the velocity of the lid and density are kept constant and the dynamic viscosity is scaled by a factor to match the Reynolds number.

A three-dimensional structured mesh of $150 \times 150 \times 3$ points with clustering near the walls has been used. For low Reynolds cases the mesh is too fine and such resolution is not necessary to obtain a good result, however these mesh has been selected to guarantee a proper boundary layer resolution for high Reynolds numbers. Moreover to match the incompressible solution, low-Mach number preconditioning has been used, otherwise the excess of artificial viscosity over-damps the solution, especially for high Reynolds numbers. The initial conditions within the cavity are: density of 1.2 Kg/m^3 , null velocity and a pressure of 101325 Pa . The mean Mach number is 0.005.

The most relevant results are displayed in Figure 7.19. The streamlines are depicted at the top row, where there is a main center vortex and counter-rotating vortices at the corners, that increase with the Reynolds number. The velocity profiles are plotted in the middle rows of the same figure. A good agreement in terms of velocity profiles between the incompressible solution and the Mu^2s^T solver is found for low Reynolds numbers, whereas for high Reynolds numbers, small discrepancies appear. Moreover, for high Reynolds numbers, some differences can be distinguished between explicit and implicit schemes, being the incompressible solution in the middle of both solutions. Finally, the convergence histories are showed in the bottom row of Figure 7.19. Lower convergence rate is observed as the Reynolds number is increased, what can explain the discrepancies observed in the velocity profiles. Nevertheless, for low Reynolds number, the Jacobi solver converges approximately 4 times faster than its explicit counterpart. The problem size is so small that the GMRES+LU-SGS scheme is not able to run faster than the Jacobi scheme.

Figure 7.19: Solution of the lid-driven cavity for different Reynolds numbers. Streamlines (top), velocity profiles (Middle) and convergence histories (Bottom) for a mesh of 150×150 nodes.

7.5.2 Two coaxial rotating disks

The flow between two rotating coaxial disks, also known as the Von Kármán flow, has been extensively studied. There is a similarity solution for infinite disks and hence it is a good example to validate the solver for cavity flows. Originally, a similarity solution of a single rotating disk was found by Von Kármán [92]. The proof of the existence of a solution for a flow between two rotating coaxial disks has been studied by many authors and rigorously determined in [4]. The boundary value problem is derived by applying the similarity transform in cylindrical coordinates (x, θ, r) :

$$v_x = \frac{\Omega r}{Re_d} f_\zeta(\zeta), \quad v_\theta = \Omega r g(\zeta), \quad v_r = -2 \frac{\Omega d}{Re_d} f(\zeta)$$

where

$$\zeta = \frac{x}{d}, \quad Re_d = \frac{2\Omega d^2}{\nu}$$

being d the gap between the disks, Re_d the rotational Reynolds number based on the gap d , Ω the rotational speed of the rotor, and ν the kinematic viscosity of the fluid.

The incompressible Navier-Stokes equations becomes a system of ordinary differential equations:

$$\begin{aligned} f_{\zeta\zeta\zeta\zeta} + f \cdot f_{\zeta\zeta\zeta} + Re_d^2 \cdot g \cdot g_\zeta &= 0 \\ g_{\zeta\zeta} + f \cdot g_\zeta - f_\zeta \cdot g &= 0 \end{aligned}$$

with the boundary conditions:

$$\begin{aligned} f(0) = f_\zeta(0) = f(1) = f_\zeta(1) = g(0) &= 0 \\ g(1) &= 1 \end{aligned}$$

This similar solution has been solved by finite differences for several parameters of Re_d . The domain for the CFD simulation consists of a cavity aspect ratio $d/b = 0.0637$, where b is the cavity outer radius. This aspect ratio is low enough to fulfill the infinite disks hypothesis. A structured grid consisting of 200×70 nodes has been used and the 2D axisymmetric Navier-Stokes equations solved in this grid. The velocity profiles for three different Reynolds numbers are shown in Figure 7.20(a). Good agreement between the similarity solution and the CFD simulations for all the Reynolds numbers can be seen. Regarding the convergence histories, which are shown in Figure 7.20(b), the implicit solver

Figure 7.20: Similar solution and $Mu^2 s^2 T$ simulation results of the non-dimensional velocity profiles for a rotor-stator system of two coaxial disks (a) and convergence histories for different schemes (b).

converges twice faster than the explicit solver.

7.5.3 Sealed Rotor-Stator Cavity

Sealed rotor-stator cavity consists of a rectangular cross-sectional cavity in which one disk is rotating while the other disk and the tip are stationary. This case has been studied experimentally by Daily and Nece ([51]) and offers a suitable test case to validate the code for this type of configurations at turbulent regimes. The sealed cavity is characterized by the aspect ratio d/b , being d the axial gap between both disks and b the outer radius. For this analysis the aspect ratio used is 0.0637. The operating condition chosen is a rotational Reynolds number equals to $4.4 \cdot 10^6$, where the rotational Reynolds number is defined as:

$$Re_\phi = \frac{\rho \Omega b^2}{\mu}$$

being, ρ the density, Ω the rotational speed and μ the dynamic viscosity. The non-dimensional velocity profiles are shown in Figure 7.21(a), where the experiments and $Mu^2 s^2 T$ solution are compared. Although the velocities are underpredicted, there is a good agreement between simulations and experiments, with a maximum error of 1.6%.

Figure 7.21: Comparison between experiments and Mu^2s^2T solver of the non-dimensional velocity profiles (a) and moment coefficient on the rotor (b) of a sealed rotor-stator cavity.

Additionally, experimental data for the moment coefficient are available from [51]. The moment coefficient is defined as:

$$c_m = \frac{4\pi}{\Omega^2 b^5} \int_{r_{inner}}^{r_{outer}} \mu r^2 \left. \frac{dV_\theta}{dx} \right|_{rotor} dr$$

A c_m comparison of experimental data and Mu^2s^2T solution is displayed in Figure 7.21(b). The momentum is overpredicted by the solver, but even though there is a relative error of 60%, the trend of the curves is similar.

7.6 ADTURB Vane

The ADTurb vane has been chosen as a representative three-dimensional turbine vane because this test case has been used historically to assess Mu^2s^2T performances. This corresponds to a high speed turbine rig designed to measure forced response at DLR. the domain, which is depicted in Figure 7.22, is meshed by a hybrid semi-structured grid by extruding through the radial direction a 2D blade-to-blade mesh built by quads and triangles, leading finally to a mesh formed by prisms of 191000 nodes. The maximum cell aspect ratio is 20. $k - \omega$ turbulence model is used. As the y^+ of the mesh is 50, an enhance wall function model for the boundary layer is also used. The exit Mach number

Figure 7.22: Mach number contours at the mid-span section and mesh detail for the ADTurB vane model. The mesh consists of 191000 nodes.

is 0.86.

The convergence of the implicit solvers, Jacobi and GMRES+LU-SGS, is compared with the explicit baseline solver, both in CPU (Figure 7.23 (a)) and GPU (Figure 7.23 (b)). Without multi-grid preconditioning, GMRES+LU-SGS solver is the fastest in CPU platform, a factor of 10 over their explicit baseline solver, whereas the Jacobi implicit solver is 4. When multi-grid preconditioning is used, the explicit solver converges in the same time as the Jacobi solver without multi-grid. The acceleration of the multi-grid preconditioning for the Jacobi solver is not as strong as the explicit case. For the explicit scheme the low frequency modes are neither convected nor damped in the fine grid, but are transformed in high frequency modes in the coarse grid, where are quickly smoothed. On the contrary, the implicit solver is able to transport quickly low frequency errors in the fine mesh, due to the larger CFL numbers, as well as the high frequency modes are properly annihilated. Some acceleration with multi-grid preconditioning is achieved when Jacobi solver is selected, but not as high as for the explicit solver. However, Jacobi with multi-grid preconditioning converges at the same time than GMRES+LU-SGS without multi-grid. When moving to GPU platform, GMRES+LU-SGS is not superior to Jacobi+multi-grid method, due to its lower efficiency in GPUs. Note that the size of this case is about 200K nodes, and the GPU efficiency coefficient, c_{eff} , is 0.82 for GMRES+LU-SGS solver whereas is 0.95 for Jacobi and explicit solvers.

Figure 7.23: Convergence histories for a ADTurB vane model (191000 nodes). The CFL numbers for the different solvers: RK5=3.5, Jacobi=15 and GMRES+LU-SGS=60. Linear iterations for the implicit solvers: Jacobi=5 and GMRES+LU-SGS=1. Krylov subspace dimension for GMRES solver=5.

Finally, the main conclusion is that a speed-up of two is achieved for the implicit solver when multi-grid preconditioning is available. When multi-grid fails, due to errors in the grid agglomeration, which is the case of the complex domains such as turbine blades that includes cavities and inter-stage seals, the implicit method is able to provide a solution in the same time that the baseline explicit solver with multi-grid.

Chapter 8

CFD Analysis in a Low-Speed Turbine Rig

8.1	Introduction	138
8.2	Model Description	139
8.3	Multistage results	144
8.4	Unsteady Flow Solution	153
8.5	Enhanced model to postdict experimental data	154
8.6	Conclusions	157

The influence of the sealing flows on the secondary flows of a low-pressure turbine has been assessed numerically using multi-row steady and unsteady simulations. The experimental data obtained at the Large Scale Turbine Rig (LSTR) at Technische Universität Darmstadt have been used to validate the numerical method and complement the simulations.

Steady and unsteady state solutions and experiments are compared to understand the importance of the unsteadiness in the accuracy of numerical simulations. It is concluded that unsteady rotor/stator simulations enhance the prediction of the stator secondary flows, especially in the tip region.

The effect of the sealing air is analysed, varying the cooling mass flow for two operating

conditions. The penetration of the sealing flow in the main stream increases with the cooling flow, displacing the horseshoe and passage vortices towards the mid-span.

8.1 Introduction

Modern turbine design demands complex numerical simulations in order to obtain more accurate results to reduce experimental analyses and costs. Therefore the effect of interactions between rotor blades, stator vanes and cavity leakages should be taken into account. Additionally, experimental techniques are not able to measure certain regions of cavities, seals and passages, leading to CFD as an essential tool to describe flow structures. This is feasible within the design loop thanks to recent computer developments, as it is described in [43], which allows to include in the simulations more geometric details and unsteady calculations.

In axial turbines there are rotating blades and stationary vanes which are not in contact, separated by spaces which allow the ingestion of hot air from the main annulus into the cavities, reducing the life of the discs. Cooling air is bled from the compressor and injected into the inner turbine cavities in order to create a positive net flow through the rim seals between the rotor and stator rows. This procedure leads to a penalty in the turbine efficiency and is of broad interest.

Many authors have researched on rim seal flows, deriving correlations for estimating ingestion due to the disc pumping [8, 22]. The aerodynamic and performance effects associated with rotor-stator gap coolant injections were investigated [64, 65], concluding that small quantities of cooling air can have significant impact on the performance and exit conditions of the turbine stage.

Recent research in linear cascades has been done [39, 86], testing different rim seal configurations. Additional experiments have been carried out in turbine rigs [11, 25, 85], studying the interaction between the main annulus and the cavities. The effect of the hot air ingestion and the cooling effectiveness is well detailed in [38].

Apart from the experimental research, cavity flows and its interaction with the main stream have been the subject of many CFD studies [46, 12, 41, 5, 3]. Steady and unsteady

state effects are assessed and the flow pattern within the inner cavities is described, understanding the interactions between cavities and main annulus.

The purpose of the present work is to compare CFD results with available experimental data, obtained by Schrewe et al. [85] at the Large Scale Turbine Rig at Technische Universität Darmstadt, which represents an upscaled low speed two-stage LPT. This is part of the EU FP6 MAGPI (Main Annulus Gas Path Interactions) programme, which has the objective of improving the understanding of interactions between cooling and main stream flows. Complete multistage simulations of the whole test matrix performed in the rig and an additional unsteady simulation of the whole turbine has been done to assess unsteadiness effects. Different cooling mass flow configurations have been studied, covering ingestion and leakage sealing flows through the upstream NGV rim seal. The flow structure within the cavity is analyzed in detail. Moreover, the egress flow from the cavity is studied, which disturbs the end-wall boundary layer and reinforces the horseshoe vortex, increasing the losses.

8.2 Model Description

8.2.1 Computational Domain and Mesh

The CFD simulations presented in this paper are based on an existing rig [84]. It consists of a two-stage low Mach number turbine. Each stage comprises 48 stator vanes and 72 rotor blades. The main interest cavity is located between the first rotor and the second stator and the cooling is blown by 24 equispaced cooling holes, which are connected to an inner sub-cavity as shown in Figure 8.1. Two different cooling flows can be supplied independently. The mass flow through the rim seal between the first stator and the first rotor is kept constant whereas the mass flow through the cooling holes can be varied for different cooling conditions. Therefore the Cooling Flow 2 is increased to obtain higher coolant flow in R1-S2 cavity but the Cooling Flow 1 must be reduced to balance the flow in S1-R1 cavity since the flow through the labyrinth seal between the sub-cavity and the S1-R1 cavity is also increased.

Figure 8.1: Sketch of the model geometry, SAS flows and instrumentation position (taken from Schrewe et al. [85]).

Figure 8.2: Model and mesh for steady state simulations (Pink: static frame, cyan: rotating frame).

Figure 8.3: Model and mesh for unsteady state simulations (Pink: static frame, cyan: rotating frame).

Several simplifications have been considered in the simulations. The lower half cavity in front of the first rotor is cut out in order to reduce the mesh size, trying not to perturb the flow structure through the rim seal, since earlier simulations proved that its impact downstream of the first rotor is negligible as long as the mass flow is not changed. The cooling holes are modelled as an axi-symmetric slot with the equivalent area for steady state simulations, whereas the real geometry is meshed for unsteady state calculations as shown in Figures 8.2 and 8.3 respectively.

The domains are meshed using an ITP in-house mesher [14]. In Figure 8.4 a sketch of the mesh generation process of the cooling hole, cavities and airfoils is shown. Two-

Figure 8.4: Mesh generation detail of the cooling hole, cavities and airfoils.

Stator 1	Rotor 1	Stator 2	Rotor 2
1.6	1.9	2.2	1.5

Table 8.1: Mesh size of every passage row (million of nodes).

dimensional unstructured hybrid meshes with quad layers to resolve boundary layers are extruded along the span-wise and pitch-wise directions to mesh the airfoils and the cavities respectively, generating semi-unstructured blocks with triangular prisms and hexahedra, which are finally merged into a conformal grid. In Table 8.1 the mesh size is described. The Reynolds Average Navier-Stokes equations are integrated up to the wall and the y^+ of the first boundary layer point is between 1-2 for the airfoil surfaces and 3-6 for the cavity walls.

Rotor-stator interfaces, which are located at the same position as the measurement planes, are modeled as mixing planes for steady state simulations (so called multistage simulations) and sliding planes for unsteady state simulations. The relative distances between these planes and the airfoils are given in Table 8.2.

Finally, only one passage is simulated for steady state cases, obtaining a mesh of 7.2 million mesh points, whereas $1/24^{th}$ periodic sector model is used for unsteady calculations,

	$x/S1_c$	$x/R1_c$	$x/S2_c$	$x/R2_c$
MP01	3.23 ←			
MP02	0.30 →	0.83 ←		
MP03		0.46 →	0.35 ←	
MP04			0.69 →	0.24 ←
MP05				1.47 →

Table 8.2: Relative distances of the Measurement Planes. (←: upstream the leading edge, →: downstream the trailing edge).

	OP1	OP2
Inlet Total Temperature (K)	313.15	308.15
Reduced Inlet Mass Flow (-)	0.099	0.060
Spool Speed (r.p.m.)	766	366
S1-R1 Total Temperature (K)	310	305
Cooling Hole Total Temperature (K)	310	305

Table 8.3: Rig operating conditions.

resulting a 16 million mesh points.

8.2.2 Simulation Matrix

Two operating points have been simulated. The first corresponds to the design working point and the second to an off-design working point. Table 8.3 presents the operating conditions for both points. The reduced mass flow is defined as:

$$\tilde{m}_i = \frac{\dot{m}_{inlet} \sqrt{R \cdot T_{t,inlet}}}{A_{inlet} \cdot P_{t,inlet}} \quad (8.1)$$

The rim seal mass flow upstream the rotor 1 is kept constant. On the other hand, the mass flow through the cooling holes is varied in order to obtain either ingested or egress flow through rim seal upstream the second stator. The SAS flows were estimated by the MAGPI partners, using a 1D flow network solver called SPAN. Table 8.4 summarizes the design mass flow rates through the rim seals obtained by SPAN.

8.2.3 Boundary Conditions

A uniform flow field is imposed at the inlets of the main annulus, rotor 1 front cavity and cooling holes. Total temperature and angles are prescribed. The mass flow is imposed varying the inlet total pressure. Linear radial distribution profile of static pressure is

Cooling Condition	OP	$\dot{m}_{S1-R1}$	$\dot{m}_{R1-S2}$	$\dot{m}_{S2-R2}$
MIP	OP1	0.5	-0.3	0.31
IP			-0.1	0.30
HL			0.3	0.30
HL-WP3			1.0	0.31
VHL			1.5	0.32
HL	OP2	0.5	0.3	0.28
HL-WP3			1.0	0.29
VHL 1			2.0	0.31
VHL 2			3.0	0.35

Table 8.4: Mass flow rates through the rim seals (% Main Annulus Flow).

imposed in the main annulus exit, taken from static pressure tapping measurements at the hub and casing.

8.3 Multistage results

Overall, these complex configurations present difficulties to converge with multigrid preconditioning and for this simulations is not used. Steady state mixing-plane and unsteady sliding-plane boundary conditions are only available for the Jacobi solver. For this reason, all the cases are run with the implicit Jacobi solver. A comparison of the convergence history of the implicit solver developed in this thesis and the baseline explicit solver are represented in Figure 8.5. Explicit solver tends to oscillate and is not able to reach the steady state solution. However, if the explicit convergence history for the explicit solver is extrapolated, it would converge approximately four times slower than the Jacobi solver, which is a similar result obtained for single blades. Looking at the mass flow balance (Figure 8.5 (right)), the explicit solution oscillates the normalised total mass flow a 6 %, whereas this error is 2 order of magnitude lower for the implicit solver. This is an example, not only the accelerate the convergence for this type of configurations, but also to guarantee the convergence.

8.3.1 Seal Air Flows

he only available mass-flow measurements are: main annulus inlet, Cooling Flow 1 and Cooling Flow 2 (see Figure 8.1). The flow split into the different cavities is estimated

Figure 8.5: Convergence history for implicit Jacobi and RK5 solvers of the density L2 norm (left) and the total mass flow balance (right).

		$\tilde{m}_i[-]$			$\% \frac{\dot{m}_{C12}}{\dot{m}_i}$		
		SAS	EXP	CFD	SAS	EXP	CFD
OP1	MIP	0.104	0.099	0.099	0.53	0.53	0.49
	IP	0.104	0.099	0.099	0.70	0.70	0.70
	HL	0.104	0.099	0.099	1.09	1.11	1.12
	HL-WP3	0.104	0.099	0.099	1.81	1.81	1.79
	VHL	0.104	0.099	0.099	2.32	2.32	2.41
OP2	HL	0.061	0.059	0.059	1.09	1.08	1.1
	HL-WP3	0.061	0.060	0.059	1.80	1.78	1.8
	VHL 1	0.061	0.060	0.059	2.85	2.81	2.54
	VHL 2	0.061	0.060	0.059	3.85	3.83	3.51

Table 8.5: Estimated, measured and computed mass flows comparisons.

by analysis using the so-called SPAN fluid network solver. Therefore, the only possible comparison is the sum of the Cooling Flow 1 and the Cooling Flow 2, which is equal to the sum of the S1-R1 rim seal and the cooling hole flows. These amounts are shown in Table 8.5. There is an acceptable agreement between CFD results and experiments, in terms of mass flows. CFD overpredicts the rim seal flow upstream the Stator 2 approx. 0.18% of the main annulus flow compared with SPAN predictions for all the operating points. It means that for the IP of OP1 the net upstream Stator 2 seal flow egresses from the cavity instead to be ingested. This over-prediction in the rim sealing flow is associated to a mass flow under-prediction through the labyrinth seal.

As similar effects for both operating points are obtained as the cooling mass flow is varied, for the sake of brevity solely OP1 results are shown from now on.

Figure 8.6: Effect of cooling mass flow on static pressure in the Stator 2 front cavity (a) and rear cavity (b) for OP1.

8.3.2 Cavity Static Pressure

The static pressure has been measured in the R1-S2 and S2-R2 cavities below the Stator 2. In Figure 8.6, the effect of the cooling flow on the static pressure is represented. Although the CFD underpredicts the pressure for low coolant rates and overpredicts it for high coolant rates, the trend is very similar both for the experiments and the numerical calculations. Moreover, CFD differs from experiments for high coolant ratios due to the cooling slot model approximation. It is worth mentioning that the static pressure increases as the cooling mass flow increases for low coolant ratios, there is a maximum around 0.6% and then a pressure drop appears as the cooling mass flow increases. The explanation of this feature is that the vortex structure within the cavity varies drastically and hence the velocity flow field, as it is observed in Figure 8.7. A large clock-wise recirculation vortex dominates the flow field for MIP. The motion of this vortex is controlled by the pumping effect of the rotor wall. For HL case, the coolant mass ratio is low enough not to disturb the main vortex, but the velocity near the stator foot is lower since the flow egress to the main annulus instead to stay rotating within the cavity. For high coolant ratios, the main central vortex is split into two counter rotating vortices and the jet impinges the rim seal platform. The velocity intensity increases near the stator foot, decreasing the pressure.

In order to assess the effect of the slot model for high coolant ratios, unsteady simulations of the Stator 2 single row with the rotating cooling hole have been performed. The time-averaged well flow structures are included in Figure 8.7. Although the main central vortex

Figure 8.7: Streamlines within the Stator 2 front cavity colored with $\|\vec{v}\|/\omega r$ for OP1.

structure is well modelled by the slot, for high coolant ratios the intensity of the vortices is underpredicted by the slot model.

8.3.3 Stator 2 hub end-wall pressure distribution

The static pressure has been measured on the hub end-wall at 4% and 10% upstream of the leading edge and 5% downstream of the trailing edge of Stator 2. In Figure 8.8 the three representative cooling conditions for OP1 are represented. The results are discussed using a non-dimensional pressure coefficient k_p [49], which is defined as:

$$k_p = \frac{P_s - P_{s,average}}{\frac{\rho\omega^2 r_{hub}^2}{2}} \quad (8.2)$$

The curves exhibit a sinusoidal shape which correspond with the potential effect of the Stator 2. At 10% in front of the Stator 2, significant discrepancies appear between CFD and experiments, particularly for MIP case and it could be due to the overprediction of the rim seal flow. For CFD results, there is a clear tendency to increase the pressure peaks as the coolant ratio increases. This effect is appreciated more clearly at 4% in front of the Stator 2, where better agreement between CFD and experiments is achieved. Moreover,

Figure 8.8: Stator 2 end-wall pressure distribution.

the pressure peaks are shifted as the coolant ratio varies because the saddle point moves from the pressure side for ingestion cases to the suction side for leakage cases.

5% downstream the Stator 2, there is a better agreement than upstream results between the CFD calculations and the experiments. The lack of experimental points do not allow to compare properly the CFD results since high gradients of static pressure are present, but there is a good agreement comparing qualitatively the shape of the curves with the results reported in [49].

8.3.4 Impact of the egress flow on the secondary structures

Flow structure at the rim seal is very complicated and determinant in the development of the secondary vortices. Typically a recirculation bubble appears in the rim seal when the flow separates from the rotor platform. The egress flow from the well cavity tends to be sucked by the rotor wall and is ejected with the opposite swirl velocity of the rim seal recirculation bubble. Therefore, the effect of the egress flow is to modify the local angle of the end-wall stream, displacing the saddle point from the pressure side to the suction side of the vane leading edge as shown in Figure 8.9 (bottom), where the end-wall streamlines are represented. This displacement of the saddle point increases the deflection of the boundary layer migration from the pressure side and the impingement against the suction side appears earlier, increasing the intensity of the passage and horseshoe vortices (red streamlines), where the egress flow is ejected.

This effect may be seen in Figure 8.9 (top). Associated with the passage vortex, a trailing vortex is formed due to different radial velocities between the stream from the pressure

Figure 8.9: Effect of the egression and ingestion upstream seal flow on the hub secondary flows. Streamlines and total pressure contours downstream the Stator 2 (top). Swirl velocity on end-wall streamlines (bottom).

and the suction side downstream the airfoil (blue streamlines). The larger losses are located at the trailing vortex since the end-wall boundary layer migration ends up at this vortex.

Figure 8.10: Profile pressure distribution at 5%, 20% and 50% span for MIP, HL and VHL cooling conditions and OP1.

8.3.5 Stator 2 pressure distribution profiles

Static pressure profiles for Stator 2 are shown in Figure 8.10 in terms of pressure coefficient defined as:

$$c_p = \frac{P_s - P_{s,MP03:50\%}}{P_{t,MP03:50\%} - P_{s,MP03:50\%}} \quad (8.3)$$

Three representative cooling conditions are compared for the design operating point. Overall, there is a perfect agreement between CFD and experiments on the pressure side, but appreciable discrepancies appear on the suction side, in particular, the differences are present along all the chord at 5% span, whereas at 20% and 50% span the differences are noticeable in the rear part of the blade. Nevertheless, the same tendency is reproduced by the simulations when the coolant ratio is varied. At mid-span the pressure profiles are not affected by the cooling mass flow (Fig. 8.10 (c)). At 20% span the c_p distributions show variations from 80% of the chord on (Fig. 8.10 (b)). Finally, at 5% span the cooling mass flow affects all the suction side and part of the pressure side close to the leading edge (Fig. 8.10 (a)).

The higher errors between CFD and experiments are located at 5% span and the vicinity of the trailing edge at 20% span are consistent with the differences in the seal air flow

upstream the Stator 2. It seems that the secondary flows are not properly predicted by the CFD calculations due to the overprediction of the sealing flow.

8.3.6 Overall Main Annulus Effects

In Figure 8.11 (a), the experimental contours of slope swirl angle downstream the second stator for the design operating point and the whole range of cooling conditions are compared with multistage simulations. Qualitative good agreement is obtained at mid-span and tip regions, but significant discrepancies can be appreciated at the hub. The underturning at 25% span for high coolant ratios is more intense in the simulations than in the experiments. For low coolant ratios, this underturning decreases for the multistage simulations and is shifted towards the hub for the experiments because the secondary vortices are systematically located at lower radius in the experiments than in the simulations. It is consistent with the earlier results since the ingestion is lower for CFD.

In Figure 8.11 (b) the circumferentially area averaged slope swirl angle is shown for three representative cooling mass flow conditions and for three measurement planes: MP02 (downstream the first stator), MP03 (downstream the first rotor) and MP04 (downstream the second stator). The cooling mass flow does not make a significant effect at MP02 and MP03, therefore only the HL condition is shown in Figure 8.11 (b). Similar effects can be seen downstream the Stator 2 compared with the contours explained before. The flow is perfectly predicted by CFD calculations downstream the first stator. However, downstream the first rotor there is a shift of the curve around 3° . The explanation is associated to the fact that angle errors in relative frame are amplified when transforming to absolute frame, a 1° error in relative frame corresponds to 3° of error in absolute frame.

Larger discrepancies are obtained in terms of circumferentially area averaged radial angle, which are not well understood, as shown in Figure 8.12 (b), being the better agreement between CFD and experiments the VHL case. Overall, CFD overpredicts the radial angle. However, there is a good qualitative agreement in the radial angle contours at MP04 in Figure 8.12 (a). A similar trend than in the slope swirl angle is observed when the cooling

Figure 8.11: Comparison of contours of slope swirl angle at MP04 between experiments (a) top and multistage CFD simulations (a) top. Circumferentially averaged incidence angle at MP02, MP03 and MP04 (b).

mass flow is varied in terms of the penetration of the secondary structures.

Total pressure loss coefficient downstream of the Stator 2 is shown in Figure 8.13, which is defined as:

$$\xi = \frac{P_{t,MP03:50\%} - P_{t,MP04}}{P_{t,MP03:50\%} - P_{MP04:50\%}} \quad (8.4)$$

Multistage simulations generate more losses than experiments (8.13 (b)). Wakes are deeper and narrower in the CFD simulations than in the experiments, which is associated to a deficiency in the turbulence modelling. The peaks of losses at 70% and 90% span are well predicted by CFD, although an additional peak appears at 80% span in the simulations. This additional peak can be understood looking at the total pressure contours in Figure 8.13 (a), where two cores of losses are perfectly distinguished, meanwhile in experiments both cores are merged. This is due to the mixing caused by the unsteadiness as it will be seen afterwards. The hub losses increase as the cooling mass flow increases because the passage vortex becomes more intense and penetrates more in the main annulus as it has been seen in previous sections. Moreover the passage vortex for low coolant ratios is for the experiments closer to the hub than for the CFD calculations. This result is consistent with the angle distributions previously presented. It is worth mentioning that there is a strengthening of the wake at 40% span because the wake is distorted by

Figure 8.12: Comparison of contours of radial angle at MP04 between experiments (a) top and multistage CFD simulations (a) top. Circumferentially averaged radial angle at MP02, MP03 and MP04 (b).

the counter rotating hub and tip passages vortices. This effect is represented in Figure 8.9 (b) with green streamlines.

8.4 Unsteady Flow Solution

Unsteady state of the whole rig has been simulated for OP1 and the HL cooling condition case in order to assess the unsteadiness effect.

The simulation includes sliding planes between the rotor blades and the stator vanes to transfer the information between different domains. Figure 8.14 shows the unsteady nature of the flow, with the rotor blade wakes passing through the stator, and the main flow and cavity flows interacting and modifying the secondary flow pattern. The whole domain has been simulated using 16 GPUs. Two and a half days were needed to complete 50 blade passing periods.

Overall, there is an acceptable agreement between multistage and time-averaged unsteady simulations at mid-span and similar results are obtained for both time-averaged unsteady and multistage simulations, in terms of pressure distributions on the Stator 2. Noticeable discrepancies with respect the steady solution are seen in the tip while the hub is fairly

Figure 8.13: Comparison of contours of total pressure loss coefficient at MP04 between experiments (a) top and multistage CFD simulations (a) top. Circumferentially averaged total pressure loss coefficient at MP04 (b).

unaffected. Unsteadiness intensifies secondary flow mixing and smooths stream-wise vortices. An improvement of the tip secondary flow prediction is shown in Figure 8.15 (d), where the second peak of losses that may be seen in the multistage simulation which is not present in the unsteady simulation at 80% span. A second improvement that has been identified is that the tangential lean of the wake is closer to the experiments in the unsteady simulation than in the multistage analysis.

8.5 Enhanced model to postdict experimental data

Since for low coolant ratios significant discrepancies between CFD and experiments appear at the stator hub, a detailed analysis has been carried out in order to understand from where these differences are coming from. As the mass flow split within the inner cavities is not measured, there is some uncertainty about the real split of the sealing flows. The multistage and unsteady simulations show more penetration of the secondary flows in the main flow path than the experiments, therefore it seems that there is an overprediction on the mass flow at R1-S2 rim seal in the numerical analyses. Many authors (e.g.: [46, 50, 21])

Figure 8.14: Snapshot of the entropy field in a time instant for a multi-row unsteady analysis. (a) 3D view of the meridional planes; (b) blade-to-blade plane view.

Figure 8.15: Contours of total pressure loss coefficient downstream the Stator 2 for experiments (a), multistage (b) and unsteady simulations (c). Circumferentially averaged total pressure loss coefficient downstream the Stator 2 (d).

state that CFD predictions usually underpredict the ingestion of hot gasses. 360-degree sector simulations show large structures within the cavities resulted from blade and vane flow interactions and rim seal ingress and egress flows that are not obtainable from sector model simulations.

A modified Stator 2 model has been used, removing the inter-stage seal domain in order to force the mass flow through the labyrinth seal. Two simulations have been performed, the first imposing the inter-stage seal mass flow expected for the experiments (0.3%) and the second increasing this mass flow up to 1.4%. The boundary conditions for the main annulus are taken from the radial distribution profiles of the experiments corresponding to the HL condition, while the cooling mass flow is kept constant (0.6%) and therefore an ingestion of 0.8% is imposed on the upstream rim seal.

Figure 8.16: Contours of total pressure loss coefficient downstream the Stator 2 for experiments (a), 0.3% ISS flow (b) and 1.4% ISS flow (c). Circumferentially averaged total pressure loss coefficient downstream the Stator 2 (d).

Figure 8.17: Effect of ISS mass flow and static pressure reference on the Stator 2 pressure distribution profiles.

Figure 8.16 shows the contours of total pressure loss coefficient and its radial distribution profile. The secondary structures have less penetration as the seal air ingestion is increased, obtaining better qualitative agreements with the experiments. Indeed, when the passage vortex goes down, a peak of losses appears around the 35% span, although it is located at a lower radius in the simulations. The passage vortex for the CFD calculation is not as close to the hub as the experimental results and its peak of losses is thinner and deeper. The inability of the $k - \omega$ model to properly mix secondary flow structures is the origin of the existence in the simulations of these thin a deep cores of losses.

Figure 8.17 displays c_p distribution for the modified model without the labyrinth seal. For the first case (0.3% of mass flow through the labyrinth seal), similar results as in the previous multistage simulations are obtained. As there is an underprediction of the c_p on the suction side, the reference static pressure upstream the Stator 2 has been reduced by 150 Pa, obtaining a much better approximation of the c_p distributions. This is because the c_p value is very sensitive to small variations of the reference static pressure when the static

pressure on the airfoil is low enough compared with the reference total pressure while the c_p remains constant when the static pressure on the airfoil is close to the reference total pressure. For instance, in the present work, variations of 0.1% of the reference static pressure induces c_p variations of 3% for the pressure side and 20% on the suction side. Upon this correction it may be observed that the matching in the pressure distribution along the airfoils is good and that the incidence of the flow in the airfoil is correct as well. Moreover, the corrected c_p for the case with a labyrinth mass flow of 1.4% is compared in Figure 8.17. The upstream rim seal ingestion does not affect the c_p at 50% and 20% span. At 5% span, there is an improvement of c_p , consistently with the results shown in Figure 8.16.

8.6 Conclusions

This section presents a numerical investigation of the spoiling effects between the main annulus of a low speed turbine rig and a stator well, in comparison with available experimental data. CFD code Mu^2s^2T and $k - \omega$ turbulence model have been used.

Steady state simulations of the whole rig using a mixing plane approximation have been performed for all the operating conditions tested in the rig, modeling the cooling holes as a slot. The cooling slot approximation has been assessed with single row models, comparing the slot and the rotating cooling hole models. The main conclusion is that higher cooling mass flow leads to higher differences in the flow field within the stator well, although the impact on the main annulus is similar for both models.

The multistage simulations overpredict the flow through the rim seal upstream the stator, producing more penetration of the secondary flows than in the experiments. This effect is more noticeable for low coolant ratios where the sensitivity of the egress flow is higher. Thinner and deeper wakes and secondary vortices result from CFD, generating more losses this deficiency, being associated to the $k - \omega$ turbulence model. Nevertheless, CFD captures perfectly the trends of the experiments as the coolant ratio is increased.

An unsteady state simulation of the whole rig has been performed for an intermediate cooling condition. The main differences between steady and unsteady models are present in the

secondary flow structures, being the latter more mixed, although the overall performance of the turbine just changes slightly.

Finally, a simplified single stage model have been studied by removing the labyrinth seal and forcing the ingestion through the stator upstream rim seal. Better results are obtained in terms of the structure of the secondary vortices downstream the stator when the ingestion is increased with respect the design intent of the experiment.

Chapter 9

Conclusions and Further Work

In this thesis, we have presented a strategy to accelerate the convergence to the steady state of the numerical solution of the Navier-Stokes equations. The starting point is an existing explicit unstructured code accelerated by means of multigrid and Block-Jacobi preconditioning. Using the same spatial discretisation as the baseline solver an implicit method capable of executing both in CPUs and GPUs has been developed.

Firstly, an investigation of the most efficient implicit methods for unstructured grids, using a linearisation based on the first-order Jacobian was conducted. The most efficient solver found during this search was the GMRES+LU-SGS, although the Jacobi solver, due to its simplicity, is worth considering.

A two-dimensional Von Neumann analysis has been performed to assess the stability and convergence analysis and determine the optimum parameter setting of the implicit method. Then, the damping and propagation of the different modes are analysed for the implicit solver and compared with the explicit baseline scheme. Also with this analysis, the asymptotic convergence rates can be estimated, and the optimum parameters of the Jacobi implicit method are derived from this analysis, which have been corroborated by means of 3D simulations of the RANS solver. It is concluded that the 2D Von Neumann analysis of the numerical method is a very valuable tool to understand the convergence and properties of the scheme.

During the development of this thesis the baseline code was ported to GPUs, and then, the implicit was optimised to be executed as well in GPUs. The solver is mainly written

in C++ and it is intended to be executed either on CPUs or GPUs. The OpenCL programming language has been used to write those portions of the code that must be executed on a multi-processor environment, as it is the case of the GPUs. This approach allows the resulting code to be executed on a wide range of architectures with a minimal implementation penalty and ensures the uniqueness of the solution for all platforms. Implementation details of the most time-consuming parts of the code have been presented. The access to the unstructured grid data needed to evaluate gradients, fluxes, etc. is not direct and presents limitations which are more severe when the edge loops are parallelised in shared memory multi-processors. Simultaneous access to the data must be avoided in order to obtain correct results. The factor that limits computing performance the most is the access to the multi-processor memory. Therefore, the larger the number of floating point operations per memory access, the more optimal the code execution is on these architectures. The MPI parallelisation has also been carefully designed to avoid communication bottlenecks that could prevent a good parallel scalability when the number of grid points per MPI process becomes very small. Moreover, when OpenCL devices such as GPUs are used, data communication between the OpenCL device and the CPU must be carefully designed. It is shown that once the OpenCL GPU-to-GPU MPI communication is optimised, it is comparable to that of other languages that allow direct GPU-to-GPU communication.

For the baseline explicit solver and a standard design case of one million point , a single process executes up to 26 times faster on a GPU NVIDIA GeForce GTX780Ti than on a CPU core Intel Xeon E5-2650 @ 2.60GHz, whereas this speed-up is 41 for the implicit Jacobi solver. If the multi-core capability of CPUs is exploited, GPUs are still up to eight times faster. However, the GPU performance is severely reduced when the problem size decreases below 10^6 mesh points. When multiple GPUs are used in parallel, the speed-up is very high as long as the number of points of each partition is large enough. When the effect of the processor efficiency is disregarded, the communications become the next bottleneck but its influence is much lower. However, for large enough cases, the speed-up is still quite acceptable. For example, the speed-up of a nine million point mesh running in 32 GPUs is 25, whereas is 9 for a one million point mesh. The overall reductions in

Multigrid	RK5	Jacobi	GMRES+LU-SGS
No	1	4	8
Yes	4	8	-

Table 9.1: Summary of the speed-up for the methods studied in this work with respect to the baseline solver without multigrid. 1 million mesh point running on a single NVIDIA GeForce GTX780Ti.

execution time and acquisition and operation costs are remarkable. The viability of the solution for a production CFD code has been demonstrated in a cluster of commodity GPUs in an industrial environment. From an economic point of view, this means a $10\times$ reduction in the acquisition cost of the hardware for the same computational power, consuming a 33% less energy. The code develop during this work is currently used as well by the end-users locally in their desktops, making use of a second dedicated standard graphics card mounted on them for this purpose.

The implementation of the Jacobi solver for many-core architectures has been carried out using the same strategy as for the residual fluxes. However to parallelise the LU-SGS solver, multi-colouring reordering is needed to construct large block sub-matrices that can be highly parallelised. This reordering induces two main disadvantages: on the first hand, the reordering causes a penalty in terms of the convergence of the linear solver, and on the second hand, after the multi-colouring reordering, there are several low-size sub-block matrices which are not highly parallelise. As consequence, the efficiency of the GMRES+LU-SGS solver on GPUs is not as good as on CPUs. Nevertheless, GMRES+LU-SGS solver converges in the same time as the Jacobi solver with multigrid preconditioning, which is around 10 times faster than the explicit baseline solver without multigrid. In turn, the Jacobi solver without multigrid preconditioning converges in the same time as the explicit baseline solver with multigrid, and an additional factor of two is obtained when the Jacobi solver is preconditioned with multigrid. These speed-up results are summarised in Table 9.1.

The new implicit method developed has been validated with a several canonical test cases well known by the CFD community. Moreover, the experimental data of a low-speed turbine rig obtained during MAGPI European project have been compared against this new CFD solver, demonstrating its capability to cope with complex simulations, including

sliding planes, cavities, etc. and reproduce accurately simple experimental data (non-dimensional pressure distributions) and complex trends associated to the purge flows.

In the near future, the implicit algorithms will be extended to the linear and non-linear harmonic version of the code, and the adjoint version used for shape optimization. In the medium term it is planned to re-use the existing knowledge and software framework to implement an implicit versions of a high-order unstructured code.

Bibliography

- [1] Jameson A. Solution of the euler equations for two dimensional transonic flow by a multigrid method. *Applied Mathematics and Computations.*, 13:327–356, 1983.
- [2] Jameson A., Schmidt W., and Turkel E. Numerical solution of the euler equations by finite volume methods using runge-kutta time stepping schemes. In *AIAA Paper 81-1259. 14th Fluid and Plasma Dynamic Conference, Palo Alto*, June 1981.
- [3] A. Andreini, R. Da Soghe, B. Facchini, and S. Zecchi. Turbine stator well cfd studies: Effects of cavity cooling air flow. In *ASME Paper No. GT2008-51067*, 2008.
- [4] Elcrat A.R. On the swirling flow between rotating coaxial disks. *Journal of Differential Equations*, 18:423–430, 1975.
- [5] V.N.D. Autef, J.W. Chew, N.J. Hills, and I.L. Brunton. Turbine stator-well flow modeling. In *ISAIF8-008*, 2007.
- [6] B. Balwin and H. Lomax. Thin layer approximation and algebraic model for separated turbulent flows. In *AIAA Paper 78-257*, 1978.
- [7] T.J. Barth and S.W. Linton. An unstructured mesh newton solver for compressible fluid flow and its parallel implementation. In *AIAA paper, 95-0221*, 1995.
- [8] F. J. Bayley and J. M. Owen. The fluid dynamics of a shrouded disk system with a radial outflow of coolant. *Journal of Engineering for Power (transactions of the ASME)*, 92:335–341, 1970.
- [9] C. Bertolli, A. Betts, N. Lorient, G.R. Mudalige, D. Radford, D.A. Ham, M.B. Giles, and P.H.J. Kelly. Compiler optimizations for industrial unstructured mesh cfd

- applications on gpus. In Hironori Kasahara and Keiji Kimura, editors, *Languages and Compilers for Parallel Computing*, volume 7760 of *Lecture Notes in Computer Science*, pages 112–126. Springer Berlin Heidelberg, 2013.
- [10] M. Blanco and D.W. Zingg. Fast newton-krylov method for unstructured grids. *AIAA J.*, 36 (4):607–612, 1998.
- [11] D. E. Bohn, A. Decker, H. Ma, and M. Wolff. Influence of sealing air mass flow on the velocity distribution in and inside the rim seal of the upstream cavity of a 1.5-stage-turbine. In *ASME Paper No. GT2003-38459*, 2003.
- [12] J. Boudet, N.J. Hills, and J.W. Chew. Numerical simulation of the flow interaction between turbine main annulus and disc cavities. In *ASME Paper No. GT2006-90307*, 2006.
- [13] T. Brandvik and G. Pullan. An accelerated 3d navier-stokes solver for flows in turbomachines. *Journal of Turbomachinery*, 13(2):021025, Oct. 2010.
- [14] M.A. Burgos, J.M. Chia, R. Corral, and C. Lopez. Rapid meshing of turbomachinery rows using semi-unstructured multi-block conformal grids. *Engineering with Computers*, 26(4):351–362, 2009.
- [15] M.A. Burgos, J. Contreras, and R. Corral. Efficient edge-based rotor/stator interaction method. *AIAA Journal*, 41 No 1:19–31, 2011.
- [16] M.A. Burgos, J. Contreras, and R. Corral. Efficient edge-based rotor/stator interaction method. *AIAA Journal*, 49:19–31, 2011.
- [17] M.A. Burgos and R. Corral. Application of phase-lagged boundary conditions to rotor/stator interaction. In *ASME Paper 2001-GT-0586. 46th ASME Gas Turbine and Aeroengine Congress, New Orleans*, 2001.
- [18] P. Castonguay, D. Williams, P. Vincent, M. Lopez, and A. Jameson. On the development of a high-order, multi-gpu enabled, compressible viscous flow solver for mixed grids. In *20th AIAA Computational Fluid Dynamics Conference*, number AIAA 2011-3229, Honolulu, HI, 2011.

- [19] Sitaraman J. Chandar, D.J. and Mavriplis. D.J. A gpu-based incompressible navier-stokes solver on moving overset grids. *International Journal of Computational Fluid Dynamics*, 27(6):268–282, 2013.
- [20] R.F. Chen and Z.J. Wang. Fast, block lower-upper symmetric gauss-seidel scheme for arbitrary grids. *AIAA J.*, 38 (12):2238–2245, 2000.
- [21] W. Cheng-Zhang, B.V. Johnson, S.P.Mathiyalagan, J.A. Glahn, and D.F. Cloud. Rim seal ingestion in a turbine stage from 360-degree time-dependent numerical simulations. In *ASME Paper No. GT2012-68193*, 2012.
- [22] J. W. Chew. A theoretical study of ingress for shrouded rotating disc systems with radial outflow. *ASME Journal of Turbomachinery*, 113:91–97, 1989.
- [23] J. Contreras and R. Corral. Quantitative influence of the steady non-reflective boundary conditions on blade-to-blade computations. In *ASME Paper 2000-GT-0515. 45th ASME Gas Turbine and Aeroengine Congress, Munich*, May 2000.
- [24] J. Contreras, R. Corral, J. Fernández-Castañeda, G. Pastor, and C. Vasco. Semiunstructured grid methods for turbomachinery applications. In *ASME paper 2002-GT-30572*, 2002.
- [25] D. D. Coren, N. R. Atkins, J. R. Turner, D. E. Eastwood, S. Davies, P. R. N. Childs, J. Dixon, and T. S. Scanlon. An advanced multi-configuration stator well cooling test facility. In *ASME Paper No. GT2010-23450*, 2010.
- [26] R. Corral. *Resolución de las Ecuaciones de Euler en régimen transónico mediante métodos multimalla*. PhD thesis, Escuela Técnica Superior de Ingenieros Aeronáuticos, Universidad Politécnica de Madrid., June 1990.
- [27] R. Corral and J. Fernández-Castañeda. Surface mesh generation by means of steiner triangulations. *AIAA Journal*, 39(1):176–180, January 2001.
- [28] A. Corrigan, F. Camelli, R. Löhner, and F. Mut. Porting feflo to gpus. In J.C.F. Pereira and A. Sequeira, editors, *V European Conference on Computational Fluid Dynamics ECCOMAS*, Lisbon, Portugal, 14-17 June 2010.

- [29] A. Corrigan, F. Camelli, R. Löhner, and J. Wallin. Running unstructured grid based cfd solvers on modern graphics hardware. In *Proceeding of the 19th AIAA Computational Fluid Dynamics*, number AIAA 2009-4001, San Antonio, Texas, 22-25 June 2009.
- [30] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the ACM National Conference*, pages 157–192, 1969.
- [31] W.N. Dawes, P.C. Dhanasekaran, and W.P. Kellar. Acceleration of an unstructured hybrid mesh rans solver by porting to gpu architectures. In *Proceedings of the 49th AIAA Aerospace Sciences Meeting*, number AIAA Paper 2011-0944 2011-944, Orlando, Florida, January 2011.
- [32] D.C. *Turbulence Modeling for CFD*. DCW Industries, Inc., 2006.
- [33] L. Deng, F. Wang, H.L. Bai, and Q.X. Xu. Parallelizing a high-order weno scheme for complicated flow structures on gpu and mic. In *International Conference on Computational Science and Engineering*, 2015.
- [34] Alexandre Denis and François Trahay. MPI Overlap: Benchmark and Analysis. In *International Conference on Parallel Processing*, 45th International Conference on Parallel Processing, Philadelphia, United States, August 2016.
- [35] Mavriplis D.J. On convergence acceleration techniques for unstructured meshes. In *AIAA paper 98-2966*, 1998.
- [36] Richard P. Dwight. *Efficiency Improvements of RANS-Based Analysis and Optimization using Implicit Methods on Unstructured Grids*. PhD thesis, University of Manchester, 2006.
- [37] Turkel E. Preconditioned methods for solving the incompressible and low speed compressible equations. *Journal of Computational Physics*, 72:277–298, 1987.
- [38] D. Eastwood, D.D. Coren, C.A. Long, N.R. Atkins, P.R.N. Childs, T.J. Scalon, and A. Guijarro-Valencia. Experimental investigation of turbine stator well rim

- seal, reingestion and interstage seal flows using gas concentration techniques and displacement measurements. In *ASME Paper No. GT2011-45874*, 2011.
- [39] R.D. Erickson, T.W. Simon, l. Zhang, and H.-K. Moon. Experimental investigation of disc cavity leakage and hub endwall contouring in a linear rotor cascade. In *ASME Paper No. GT2011-46700*, 2011.
- [40] Gisbert F. *Resolution of the Adjoint Navier-Stokes Equations using a Preconditioned Multigrid Method*. PhD thesis, Escuela Técnica Superior de Ingenieros Aeronáuticos, Universidad Politécnica de Madrid, 2007.
- [41] C. Georgakis, C. Whitney, G. Woollatt, V. Stefanis, and P. Childs. Turbine stator well cfd studies: Effect of upstream egress ingestion. In *ASME Paper No. GT2007-27406*, 2007.
- [42] M.B. Giles. Non-reflecting boundary conditions for euler equations calculations. *AIAA Journal*, 28(12):2050–2057, 1990.
- [43] F. Gisbert, R. Corral, and G. Pastor. Implementation of an edge-based navier-stokes solver for unstructured grids in graphics processing units. In *ASME Paper No. GT2011-46224*, 2011.
- [44] F. Gisbert, R. Corral, and J. Pueblas. Computation of turbomachinery flows with a parallel unstructured mesh navier-stokes equations solver on gpus. In *AIAA Paper 2013-2864. 21th AIAA Computational Fluid Dynamics Conference, San Diego*, 2013.
- [45] Intel. Intel xeon e2650 v2 specifications. http://ark.intel.com/products/75269/Intel-Xeon-Processor-E5-2650-v2-20M-Cache-2_60-GHz, 2013.
- [46] R. Jakoby, L. De Vito, J. Larson, K. Lindblad, T. Zierrer, D. Bohn, A. Decker, and J. Funcke. Numerical simulation of the unsteady flow field in an axial gas turbine rim seal configuration. In *ASME Paper No. GT2004-53829*, 2004.
- [47] A. Jameson and E. Turkel. Implicit schemes and lu decompositions. *Mathematics of Computations*, 37:385–397, 1981.

- [48] A. Jameson and S. Yoon. Lower-upper implicit schemes with multiple grids for the euler equations. *AIAA J.*, 25(7):929–935, 1987.
- [49] B.V. Johnson, R. Jakoby, D.E. Bohn, and D. Cunat. A method for estimating the influence of time-dependent vane and blade pressure fields on turbine rim seal ingestion. *ASME Journal of Turbomachinery*, 131, 2009.
- [50] S. Julien, J. Lefrancois, G. Dumas, G. Boutet-Blais, S. Lapointe, J.-F. Caron, and R. Marini. Simulations of flow ingestion and related structures in a turbine disk cavity. In *GT2010-22729*, 2010.
- [51] Daily J.W. and Nece R.E. Chamber dimension effects on induced flow and frictional resistance of enclosed rotating disks. *ASME J. Basic Eng.*, 82:217–232, 1960.
- [52] A. Karakus, T. Warburton, M.H.M. Aksel, and C. Sert. A gpu-accelerated adaptive discontinuous galerkin method for level set equatio. *International Journal of Computational Fluid Dynamics*, 30(1):56–68, 2016.
- [53] G. Karypis, K. Schloegel, and V. Kumar. Parmetis, parallel graph partitioning and sparse matrix ordering library, version 4.0.3, August 15 2013.
- [54] A. Khajeh-Saeed and J.B. Perot. Direct numerical simulation of turbulence using gpu accelerated supercomputers. *Journal of Computational Physics*, 235:241–257, February 2013.
- [55] Khronos Group. Opencl 2.0 specification. <https://www.khronos.org/registry/cl/specs/openc1-2.0.pdf>, November 2013.
- [56] Martinelli L. *Calculations of viscous flow with a multigrid method*. PhD thesis, Princeton University, 1987.
- [57] R. Lohner. Matching semi-structured and structured grids for navier-stokes calculations. In *AIAA paper 93-3348*, 1993.
- [58] R. Löhner. *Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods, 2nd Edition*. Wiley, 2008.

- [59] R. Löhner and M. Galle. Minimisation of indirect addressing for edge-based field solvers. *Commun. Numer. Meth. Engng*, 18:335–343, March 2002.
- [60] H. Luo, J. D. Baum, and R. Löhner. A fast, matrix-free implicit method for compressible flows on unstructured grids. *Journal of Computational Physics.*, 146:664–690, 1998.
- [61] H. Luo, D. Sharov, J.D. Baum, and R. Lohner. On the computation of compressible turbulent flows on unstructured grids. *International Journal of Computational Fluid Dynamics*, 4, issue 4:253–270, 2001.
- [62] Hong Luoa, Dimitri Sharov, Joseph D. Baum, and Rainald Lohner. Parallel unstructured grid gmres+lu-sgs method for turbulent flows. In *AIAA paper 2003-273*, page 0d, 2003.
- [63] D. Mavriplis. *Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes*. PhD thesis, Mechanical and Aerospace Engineering Department, Princeton University, 1987.
- [64] C. McLean, C. Camci, and B. Glezer. Mainstream aerodynamic effects due to wheel-space coolant injection in a high-pressure turbine stage: Part i - aerodynamic measurements in the stationary frame. *ASME Journal of Turbomachinery*, 123, 2001.
- [65] C. McLean, C. Camci, and B. Glezer. Mainstream aerodynamic effects due to wheel-space coolant injection in a high-pressure turbine stage: Part ii - aerodynamic measurements in the stationary frame. *ASME Journal of Turbomachinery*, 123, 2001.
- [66] A. Meister. Comparison of different krylov subspace methods embedded in an implicit finite volume scheme for the computation of viscous and inviscid flow fields on unstructured grids. *J. Computational Physics*.140, 311-345, 140:311–345, 1998.
- [67] Y. Moryossef and Y. Levy. Unconditionally positive implicit procedure for two-equation turbulence models: Application to k-w turbulence models. *Journal of Computational Physics.*, 220:88–108, 2006.

- [68] Pierce N. and Giles M. Preconditioned multigrid methods for compressible flow calculations on stretched meshes. *Journal of Computational Physics*, 136:425–445, 1997.
- [69] R. Ni. A multiple grid scheme for solving the euler equations. *AIAA 5th Computational Fluid Dynamics Conference*, pages 257–264, 1981.
- [70] K. Niemeyer and C.J. Sung. Recent progress and challenges in exploiting graphics processors in computational fluid dynamics. *J. Supercomput.*, 67(2):528–564, 2014.
- [71] NVIDIA. Nvidia geforce gtx780ti specifications. <http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-780-ti/specifications>, January 2013.
- [72] NVIDIA. Cuda. http://www.nvidia.com/object/cuda_home_new.html, 2016.
- [73] IEEE Spectrum: Special Report: 50 Years of Moore’s Law (Interview). Gordon moore: The man whose name means progress, the visionary engineer reflects on 50 years of moore’s law, March 2015.
- [74] Ohio State University. Mvapich: Mpi over infiniband, 10gige/iwarp and roce, 2016.
- [75] OpenACC, 2016.
- [76] Moinier P. *Algorithm Developments for an Unstructured Viscous Flow Solver*. PhD thesis, University of Oxford, 1999.
- [77] A. Pueyo and D.W. Zingg. An efficient newton-gmres solver for aerodynamic computations. In *AIAA Paper, 97-1955*, 1997.
- [78] Gómez R. *Una Estructura de Datos basada en Aristas para la Resolución de las Ecuaciones de Navier-Stokes*. PhD thesis, Escuela Técnica Superior de Ingenieros Aeronáuticos, Universidad Politécnica de Madrid, Noviembre 2000.
- [79] P.L. Roe. Approximate riemann solvers, parameter vectors and difference schemes. *J. Comput. Phys*, 43-2:357–372, 1981.
- [80] G. Romanelli, L. Mangani, E. Casartelli, A. Gadd, and M. Favale. Implementation of explicit density-based unstructured cfd solver for turbomachinery applications

- on graphical processing units. In *Proceedings of ASME Turbo Expo 2015*, number GT2015-43396, Montréal, Canada, June 2015. International Gas Turbine Institute.
- [81] Allmaras S. Analysis of a local matrix preconditioner for the 2d navier-stokes equations. In *AIAA Paper 1993-3330. 11th AIAA Computational Fluid Dynamics Conference, Orlando*, July 1993.
- [82] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.
- [83] A.I. Sayma, M. Vahdati, L. Sbardella, and M. Imregun. Modeling of three-dimensional viscous compressible turbomachinery flows using unstructured hybrid grids. *AIAA J.*, 38. No 6:945–954, 2000.
- [84] S. Schrewe, C. Linker, A. Krichbaum, and H.-P. Schiffer. Measurements of rim seal mixing processes in an axial two stage turbine. In *20th International Society of Airbreathing Engines Conference*, 2011.
- [85] S. Schrewe, H. Werschnik, and H.-P. Schiffer. Experimental analysis of the interaction between rim seal and main annulus flow in a low pressure two stage axial turbine. In *ASME Paper No. GT2012-68192*, 2012.
- [86] P. Schuler, K. Dullenkopf, and H.-J. Bauer. Investigation of the influence of different rim seal geometries in a low pressure turbine. In *ASME Paper No. GT2011-45682*, 2011.
- [87] D. Sharov and K. Nakahashi. Reordering of 3-d hybrid unstructured grids for vectorized lu-sgs navier-stokes computations. In *AIAA Paper, 97-2102*, 1997.
- [88] W. Sutherland. The viscosity of gases and molecular force. *Philosophical Magazine*, 36(5):507–531, 1893.
- [89] Sloan S.W. A fast algorithm for generating constrained delaunay triangulations. *Computers and Structures*, 47, No. 3:441–450, 1993.
- [90] J. Tölke and M. Krafczyk. Teraflop computing on a desktop pc with gpus for 3d cfd. *Int. J. Comput. Fluid Dyn.*, 22(7):443–456, 2008.

- [91] Ghia U., Ghia K.N., and Shin C.T. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of Computational Physics*, 48:387–411, 1982.
- [92] Kàrmàn V. Über laminare und turbulente reibung. *Zeit Angew. Math. Mech.* 1, pages 232–252, 1921.
- [93] B. van Leer. Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov’s method. *Journal of Computational Physics*, 32:101–136, 1979.
- [94] V. Venkatakrishnan. Implicit schemes and parallel computing in unstructured grid cfd. Technical report, 26th Computational Fluid Dynamics Lecture Serires, VKI, 1995.
- [95] V. Venkatakrishnan and T.J. Barth. Application of direct solvers to unstructured meshes for the euler and navier-stokes equations using upwind schemes. In *AIAA paper 89-0364*, 1989.
- [96] V. Venkatakrishnan and D.J. Mavriplis. Implicit solvers for unstructured meshs. *J. Computational Physics*, 105(1):83–91, 1993.
- [97] J.R. Wood, T. Strasizar, and M. Hathaway. Test case e/ca-6, subsonic turbine cascade t106. In *Test Cases for Computation of Internal Flows*. AGARD-AR-275, July 1990.
- [98] H.C. Yee. Linearized form of implicit tvd schemes for the multidimensional euler and navier-stokes equations. *Computers and Mathematics with Applications*, 12 (4):413–432, 1986.
- [99] S. Yoon and A. Jameson. Lower-upper symmetric-gauss-seidel method for the euler and navier-stokes equations. *AIAA J.*, 26(9):1025–1026, 1988.

Appendix A

Boundary Conditions Derivatives

- **Inlet:**

For the 1D non-reflecting inlet boundary conditions, the invariant R^- is linearized, which derivatives with respect the conservative variables are:

$$\frac{\partial R^-}{\partial U} = \frac{1}{\rho a} (-v_n a + \gamma (E - u_k^2), n_x a + \gamma u, n_y a + \gamma v, n_z a + \gamma w, \gamma)$$

To derive the derivatives of the total pressure, first the total pressures is expressed as:

$$P_t = P \left(1 + \frac{\gamma - 1}{2} M^2 \right)^{\frac{\gamma}{\gamma - 1}}$$

The derivative of the total pressure with respect the conservative variables, after applying the chain rule, are:

$$\frac{\partial P_t}{\partial u_i} = \left(1 + \frac{\gamma - 1}{2} M^2 \right)^{\frac{1}{\gamma - 1}} \left[\frac{\partial P}{\partial u_i} \left(1 + \frac{\gamma - 1}{2} M^2 \right)^{\gamma} + \frac{\gamma P}{2} \frac{\partial (M^2)}{\partial u_i} \right]$$

The different derivatives are:

$$\frac{\partial P}{\partial U} = (\gamma - 1) \left(\frac{u_k^2 + \omega^2 r^2}{2}, -u, -v, -w, 1 \right)$$

$$\frac{\partial (M^2)}{\partial U} = \frac{1}{a^2} \left(\frac{\partial u_k^2}{\partial U} - M^2 \frac{\partial a^2}{\partial U} \right)$$

where:

$$\frac{\partial u_k^2}{\partial U} = \frac{2}{\rho} (-u_k^2, u, v, w, 0)$$

$$\frac{\partial a^2}{\partial U} = \frac{\gamma (\gamma - 1)}{\rho} (-E + u_k^2, -u, -v, -w, 1)$$

The derivative of the total temperature with respect the conservative variables are:

$$\frac{\partial T_t}{\partial u_i} = \frac{\partial T}{\partial u_i} \left(1 + \frac{\gamma - 1}{2} M^2 \right) + \frac{(\gamma - 1) P}{2\rho} \frac{\partial (M^2)}{\partial u_i}$$

$$\frac{\partial T}{\partial U} = \frac{\gamma - 1}{\rho} (-E + u_k^2, -u, -v, -w, 1)$$

The swirl angle is defined as:

$$\alpha_s = \text{atan} \left(\frac{v_\theta}{v_m} \right)$$

being:

$$v_\theta = v \cdot \sin\theta - w \cdot \cos\theta$$

$$v_m = \sqrt{u^2 + v_r^2}$$

and the radial velocity is:

$$v_r = v \cdot \cos\theta + w \cdot \sin\theta$$

and the angle θ is:

$$\tan\theta = \frac{z}{y}$$

when the case has not asi-symmetric periodicity, the angle θ is equal to $\pi/2$.

The derivatives of the swirl angle with respect the conservative variables is:

$$\frac{\partial \alpha_s}{\partial u_i} = \frac{1}{\rho u_k^2} \left(v_m \frac{\partial v_\theta}{\partial u_i} - v_\theta \frac{\partial v_m}{\partial u_i} \right)$$

the different derivatives that appear are:

$$\frac{\partial v_\theta}{\partial U} = (0, 0, \sin\theta, -\cos\theta, 0)$$

$$\frac{\partial v_m}{\partial U} = \frac{1}{v_m} (0, u, v_r \cos\theta, v_r \sin\theta, 0)$$

Finally, the radial angle is defined as:

$$\alpha_r = \text{atan} \left(\frac{v_r}{u} \right)$$

The derivatives of the radial angle with respect the conservative variables is:

$$\frac{\partial \alpha_r}{\partial u_i} = \frac{1}{\rho v_m^2} \left(u \frac{\partial v_r}{\partial u_i} - v_r \frac{\partial u}{\partial u_i} \right)$$

$$\frac{\partial v_r}{\partial U} = (0, 0, \cos\theta, \sin\theta, 0)$$

$$\frac{\partial u}{\partial u_i} = (0, 1, 0, 0, 0)$$

• **Outlet:**

For the 1D non-reflecting outlet boundary conditions, the entropy s , the invariant R^+ , and the flow angles α_s and α_r are linearized.

The derivatives with respect the conservative variables of the entropy are:

$$\frac{\partial s}{\partial U} = \frac{\gamma - 1}{\rho^\gamma} \left(\frac{u_k^2 + \omega^2 r^2}{2} - \frac{a^2}{\gamma - 1}, u, v, w, 1 \right)$$

The derivatives with respect the conservative variables of the invariant R^+ are:

$$\frac{\partial R^+}{\partial U} = \frac{1}{\rho} \left(-|\vec{v}| + \frac{\gamma}{a} (|\vec{v}|^2 - E) \right), u \left(\frac{1}{|\vec{v}| + \epsilon_v} - \frac{\gamma}{a} \right), v \left(\frac{1}{|\vec{v}| + \epsilon_v} - \frac{\gamma}{a} \right), w \left(\frac{1}{|\vec{v}| + \epsilon_v} - \frac{\gamma}{a} \right), \frac{\gamma}{a}$$

The derivatives with respect the conservative variables of the angle α_s are:

$$\frac{\partial \alpha_s}{\partial U} = \frac{1}{\rho |\vec{v}|^3} (0, -u \cdot v, u^2 + w^2, -v \cdot w, 0)$$

The derivatives with respect the conservative variables of the angle α_r are:

$$\frac{\partial \alpha_r}{\partial U} = \frac{1}{\rho |\vec{v}|^3} (0, -u \cdot w, -v \cdot w, u^2 + v^2, 0)$$

Appendix B

GMRES linear solver

GMRES is a projection method to solve linear systems 4.1, which belongs to the family of Krylov subspace methods. This is a generalization of the conjugate gradient method for solving a linear system where the coefficient matrix is not symmetric and/or positive definite. The idea of the algorithm is to minimise the residual in each iteration of the form:

$$r_{m,min} = \min_{x_m \in x_0 + \mathcal{K}_m} \|b - Ax_m\| \quad (\text{B.1})$$

To generate the orthogonal basis of the Krylov subspace there are several versions, such as the Householder Arnoldi or Arnoldi-Modified Gram-Schmidt. The latter is used in this work since it is more reliable in presence of round-off errors compared with the standard Arnoldi's procedure [82].

The first vector to generate the Krylov basis is the normalized initial residual:

$$v_1 = \frac{r_0}{\|r_0\|} \quad (\text{B.2})$$

the next vector of the basis is generated by projecting the vector Av_1 orthogonally onto the vector v_1 and then subtracting this projection from itself:

$$v_2 = \frac{Av_1 - \text{proj}_{v_1}(Av_1)}{\|Av_1 - \text{proj}_{v_1}(Av_1)\|} \quad (\text{B.3})$$

where $\text{proj}_{v_1}(Av_1)$ is the orthogonal projection of Av_1 onto the vector v_1 and can be calculated as: The operator $\langle , \rangle$ denotes the scalar product. The vector v_2 is normalised in order to obtain an orthonormal basis. The physical interpretation of the basis generation

Figure B.1: Interpretation of the Arnoldi process to create the Krylov subspace basis.

is depicted in Figure B.1.

The rest of the recurrence is constructed as follows: defining $h_{j,m-1} = \langle Av_{m-1}, v_j \rangle$ and $h_{m,m-1} = \left\| Av_{m-1} - \sum_{j=1}^{m-1} \text{proj}_{v_j} (Av_{m-1}) v_j \right\|$, we can write:

$$v_m h_{m,m-1} + \sum_{j=1}^{m-1} v_j h_{j,m-1} = Av_{m-1} \quad (\text{B.4})$$

And expressing the above relation in matrix form:

$$V_{m+1} \tilde{H}_m = AV_m \quad (\text{B.5})$$

where $\tilde{H}_k$ is the Hessenberg matrix defined as:

$$\tilde{H}_m = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \dots & \dots & h_{1,m} \\ h_{2,1} & h_{2,2} & h_{2,3} & \dots & \dots & h_{2,m} \\ 0 & h_{3,2} & h_{3,3} & \dots & \dots & h_{3,m} \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & h_{m,m-1} & h_{m,m} \\ 0 & \dots & \dots & 0 & 0 & h_{m+1,m} \end{bmatrix} \quad (\text{B.6})$$

and V_m is the matrix formed by the Arnoldi's vectors $v_1, v_2, \dots, v_m$ as columns:

$$V_m = \begin{bmatrix} | & | & & | & | & | \\ v_1 & | & v_2 & | & \dots & | & \dots & | & \dots & | & v_m \\ | & | & & | & | & | \end{bmatrix} \quad (\text{B.7})$$

The approximate solution x_m expressed into the Arnoldi basis is:

$$x_m = x_0 + V_m y_m \quad (\text{B.8})$$

Instead of searching using the coordinates of vector x_m , its components in the Arnoldi basis, denoted by y_m , are used as the degrees of freedom to minimise the residual $b - Ax_m$. After some algebra, the residual to minimise becomes:

$$\begin{aligned} \|b - Ax_m\| &= \|b - A(x_0 + V_m y_m)\| = \|r_0 - AV_m y_m\| = \left\| \|r_0\| v_1 - V_{m+1} \tilde{H}_m y_m \right\| = \\ &= \left\| V_{m+1} \left(\|r_0\| e_1 - \tilde{H}_m y_m \right) \right\| = \left\| \|r_0\| e_1 - \tilde{H}_m y_m \right\| \end{aligned}$$

Then Equation B.1) becomes:

$$\|b - Ax_m\|_{\min} = \left\| \|r_0\| e_1 - \tilde{H}_m y_m \right\|_{\min} \quad (\text{B.9})$$

being e_1 the unitary vector: $(1, 0, \dots, 0)^T$. The objective now is to solve the least square problem of the right side of Equation B.9. Since the Hessenberg matrix is almost upper triangular, differing only in the sub-diagonal as non-null elements, it is not difficult to transform the Hessenberg matrix into upper triangular matrix by means of Givens rotations. The first rotation is:

$$\Omega_1 = \begin{bmatrix} c_1 & s_1 & 0 & \dots & 0 \\ -s_1 & c_1 & 0 & & \vdots \\ 0 & 0 & 1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 1 \end{bmatrix} \quad (\text{B.10})$$

where:

$$c_1 = \frac{h_{1,1}}{\sqrt{h_{1,1}^2 + h_{2,1}^2}}, \text{ and } s_1 = \frac{h_{2,1}}{\sqrt{h_{1,1}^2 + h_{2,1}^2}} \quad (\text{B.11})$$

after the first rotation, the Hessenberg matrix becomes:

$$\tilde{H}_k^{(1)} = \begin{bmatrix} h_{1,1}^{(1)} & h_{1,2}^{(1)} & h_{1,3}^{(1)} & \dots & \dots & h_{1,m}^{(1)} \\ 0 & h_{2,2}^{(1)} & h_{2,3}^{(1)} & \dots & \dots & h_{2,m}^{(1)} \\ 0 & h_{3,2} & h_{3,3} & \dots & \dots & h_{3,m} \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & h_{m,m-1} & h_{m,m} \\ 0 & \dots & \dots & 0 & 0 & h_{m+1,m} \end{bmatrix} \quad (\text{B.12})$$

where the element $h_{2,1}^{(1)}$ is eliminated. Repeating the process, the generic rotation matrix can be written as:

$$\Omega_i = \begin{bmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & c_i & s_i & \ddots & \vdots \\ \vdots & \ddots & -s_i & c_i & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 0 & 1 \end{bmatrix} \quad (\text{B.13})$$

where:

$$c_i = \frac{h_{i,i}^{(i-1)}}{\sqrt{\left(h_{i,i}^{(i-1)}\right)^2 + h_{i+1,i}^2}}, \text{ and } s_i = \frac{h_{i+1,i}}{\sqrt{\left(h_{i,i}^{(i-1)}\right)^2 + h_{i+1,i}^2}}. \quad (\text{B.14})$$

After this rotation, the Hessenberg matrix is transformed to:

$$\tilde{H}_m^{(i)} = \begin{bmatrix} h_{1,1}^{(1)} & h_{1,2}^{(1)} & h_{1,3}^{(1)} & \dots & \dots & \dots & \dots & \dots & h_{1,m}^{(1)} \\ 0 & h_{2,2}^{(2)} & h_{2,3}^{(2)} & \dots & \dots & \dots & \dots & \dots & h_{2,m}^{(2)} \\ 0 & 0 & h_{3,3}^{(3)} & \dots & \dots & \dots & \dots & \dots & h_{3,m}^{(3)} \\ \vdots & \ddots & \ddots & \ddots & \ddots & & & & \vdots \\ \vdots & & \ddots & \ddots & h_{i,i}^{(i)} & h_{i,i+1}^{(i)} & \dots & \dots & h_{i,m}^{(i)} \\ \vdots & & & \ddots & 0 & h_{i+1,i+1}^{(i)} & \dots & \dots & h_{i+1,m}^{(i)} \\ \vdots & & & & \ddots & h_{i+2,i+1} & \ddots & & \vdots \\ \vdots & & & & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & & & & h_{k,k-1} & h_{k,k} \\ 0 & \dots & \dots & \dots & \dots & \dots & \dots & 0 & h_{k+1,k} \end{bmatrix} \quad (\text{B.15})$$

Finally, this process leads to:

$$\tilde{R}_m = \tilde{H}_m^{(m)} = Q_m \tilde{H}_m = \begin{bmatrix} R_m \\ - \\ 0 \end{bmatrix} \quad (\text{B.16})$$

Being: $Q_m = \Omega_m \Omega_{m-1} \dots \Omega_1$, which is an orthogonal matrix that fulfills: $Q_m Q_m^T = I$.

Equation B.9 can be manipulated to obtain:

$$\|b - Ax_m\|_{\min} = \left\| Q_m^T \left(Q_m \|r_0\| e_1 - Q_m \tilde{H}_m y_m \right) \right\|_{\min} \quad (\text{B.17})$$

Since the matrix Q_m is unitary, Equation B.17 becomes:

$$\|b - Ax_m\|_{\min} = \left\| \tilde{g}_m - \tilde{R}_m y_m \right\|_{\min}, \quad (\text{B.18})$$

where the independent term has been rotated as:

$$\tilde{g}_m = Q_m \|r_0\| e_1 = \begin{bmatrix} g_m \\ - \\ \gamma_{m+1} \end{bmatrix} \quad (\text{B.19})$$

The residual can then be expressed as:

$$\left\| \tilde{g}_m - \tilde{R}_m y_m \right\| = \sqrt{\|g_m - R_m y_m\|^2 + \gamma_{m+1}^2} \quad (\text{B.20})$$

The minimum of the left-hand side is reached when the first term of the right-hand side

of Equation B.20 vanishes. If the coefficient matrix A is non-singular, then R_m is also non-singular, and therefore the minimum residual is achieved when

$$y_m = R_m^{-1} g_m \tag{B.21}$$

and is equal to γ_{m+1} . This property is very useful to set a breakdown for this algorithm, because the minimum residual of the next step can be known a priori during the Arnoldi basis generation. If the residual is below a certain level, the Arnoldi iteration can be stopped, and then the solution is updated with the current rotated Hessenberg matrix. To obtain the vector y_m from Equation B.21 is straightforward, because the matrix R_m is upper triangular.

Appendix C

Asymptotic analysis of the 2D Linear Advection amplification matrix

Considering the Fourier spatial operator corresponding to the bi-dimensional linear advection equation 5.26 and JST scheme, which is given by 5.29, the asymptotic analysis let us to assess the trend of the maximum eigenvalue of the amplification matrix of the numerical scheme. Here two methods are studied, the family of Runge-Kutta schemes and the implicit integration based on a first-order Jacobian matrix linearisation.

Moreover, there are two limits of the eigenvalues footprint, the first corresponds when the flow angle is much smaller than the cell aspect ratio ($\tan \alpha \ll \Lambda$) and the second limits appears when $\tan \alpha \sim \Lambda$. Without loss of generality, we assume that the cell aspect ratio is fixed to one and the first limit corresponds with $\alpha = 0^\circ$ and the second limit with $\alpha = 45^\circ$.

For the first limit, the Fourier spatial operators becomes:

$$\begin{aligned} \tilde{Z}_{\alpha=0} = & -\frac{CFL}{2} \{i [\sin(m\Delta x)] + \\ & + \mu_4 [(\cos(m\Delta x) - 1)] \cdot \\ & \cdot [(\cos(m\Delta x) + \cos(n\Delta y) - 2)]\} \end{aligned} \quad (C.1)$$

and for the second limit this operator is:

$$\tilde{Z}_{\alpha=45} = -\frac{CFL}{4} \{ \mu_4 [(\cos(m\Delta x) + \cos(n\Delta y) - 2)]^2 + i [\sin(m\Delta x) + \sin(n\Delta y)] \} \quad (C.2)$$

Runge-Kutta scheme:

For the explicit Runge-Kutta scheme family, the amplification matrix is given by Equation 5.14. The maximum eigenvalue appears for long wave numbers, that is, when $m\Delta x \sim n\Delta y \ll 1$. For simplicity, we rename $m\Delta x \sim n\Delta y = \hat{k}$ as the wave number. Neglecting lower order terms, for the maximum damping mode, the eigenvalue of the amplification matrix can be approximated as:

$$\lambda_{max} \simeq 1 + Z + \frac{Z^2}{2} + \dots \simeq 1 + Z_R + \frac{Z_R^2 - Z_I^2}{2} + i(Z_I + Z_R Z_I) + \dots$$

being:

$$Z_R \sim -\mu_4 CFL \hat{k}^4$$

$$Z_I \sim CFL \hat{k}$$

which modulus of the maximum eigenvalue of the amplification matrix is:

$$|\lambda_{max}| \simeq \sqrt{1 + 2Z_R} \simeq 1 + Z_R \simeq 1 - \mu_4 CFL \hat{k}^4$$

where lower order terms have been neglected. From this analysis can be derived that the explicit method damps the higher mode proportional to the CFL number and inversely proportional to the number of grid points to the power of four.

Implicit method based on a first-order Jacobian:

For the implicit time integration based on the first-order Jacobian matrix, the amplification matrix is given by Equation 5.22. On the contrary of the case of the explicit scheme, the implicit solver based on the first-order Jacobian presents two different performances for the both limits cases $\alpha = 0^\circ$ and $\alpha = 45^\circ$. In turn, there are other two limits when the CFL number is of the order of unity or tends to infinity. Therefore, the four combinations are studied separately.

Implicit method based on a first-order Jacobian, $\alpha = 0^\circ$ and $CFL \rightarrow \infty$:

For this analysis, without loss of generality, only the modes through the flow direction can be considered, i.e., $n\Delta y = 0$, reducing the analysis to a 1D problem. Then, the Fourier

spatial operators become:

$$Z_R = -\frac{\mu_4}{2} CFL (\cos(m\Delta x) - 1)^2$$

$$Z_R^{1st} = \frac{1}{2} CFL (\cos(m\Delta x) - 1)$$

$$Z_I = -\frac{1}{2} CFL \sin(m\Delta x)$$

The amplification matrix given by 5.22 can be approximated as:

$$G_{CFL \rightarrow \infty} \simeq -\frac{Z_R - Z_R^{1st}}{Z_R^{1st} + iZ_I}$$

which modulus is:

$$|\lambda_{max}| \simeq \frac{Z_R - Z_R^{1st}}{\sqrt{(Z_R^{1st})^2 + (Z_I)^2}}$$

and after some algebra becomes:

$$|\lambda_{max}| \simeq \frac{-\frac{\mu_4}{2} (\cos(m\Delta x) - 1)^2 - \frac{1}{2} (\cos(m\Delta x) - 1)}{\sqrt{\frac{1 - \cos(m\Delta x)}{2}}}$$

the maximum is found for high frequency modes, i.e., $m\Delta x = \pi$. Finally, the spectral radius of the amplification matrix for the implicit scheme based on the first-order Jacobian, a flow angle $\alpha = 0^\circ$ and large CFL numbers is:

$$|\lambda_{max}| \simeq 1 - 2\mu_4$$

which indicates that the spectral radius is constant and does not depend on the grid size.

Implicit method based on a first-order Jacobian, $\alpha = 0^\circ$ and $1 \ll CFL \ll \infty$:

For this case, the amplification matrix given by 5.22 can be approximated as:

$$G_{CFL \rightarrow \infty} \simeq \frac{I + Z_R - Z_R^{1st}}{I - Z_R^{1st} - iZ_I}$$

which their eigenvalues modulus are:

$$|\lambda_{max}| \simeq \frac{1 - Z_R - Z_R^{1st}}{\sqrt{(1 - Z_R^{1st})^2 + (Z_I)^2}}$$

and after some algebra becomes:

$$|\lambda_{max}| \simeq \frac{1 - \frac{\mu_4}{2} CFL (\cos(m\Delta x) - 1)^2 - \frac{1}{2} CFL (\cos(m\Delta x) - 1)}{\sqrt{1 + CFL (1 - \cos(m\Delta x)) (0.5 + CFL)}}$$

the maximum is found for low frequency modes, i.e., $m\Delta x \ll 1$, and assuming that the CFL numbers much higher than 0.5, the maximum eigenvalue is of the form:

$$|\lambda_{max}| \simeq \frac{1 - \frac{\mu_4}{8} CFL (m\Delta x)^4 + \frac{1}{4} CFL (m\Delta x)^2}{\sqrt{1 + 0.5 CFL^2 (m\Delta x)^2}} \simeq \left(1 + \frac{1}{4} CFL (m\Delta x)^2\right) \left(1 - \frac{1}{4} CFL^2 (m\Delta x)^2\right)$$

and finally, we the maximum eigenvalue is yield:

$$|\lambda_{max}| \simeq 1 - \frac{1}{4} CFL^2 (m\Delta x)^2$$

which indicates that the spectral radius is reduced proportionally with the square of the CFL number and increased proportionally with the square of the number of grid points.

Implicit method based on a first-order Jacobian, $\alpha = 45^\circ$ and $CFL \rightarrow \infty$:

For this analysis, we can not assume that $n\Delta y = 0$, because there is a coupling of cross modes in the imaginary part of the spatial operator and this effect can not be neglected. Then, the Fourier spatial operators become:

$$\begin{aligned} Z_R &= -\frac{CFL}{4} \mu_4 [(\cos(m\Delta x) + \cos(n\Delta y) - 2)]^2 \\ Z_R^{1st} &= \frac{CFL}{4} (\cos(m\Delta x) + \cos(n\Delta y) - 2) \\ Z_I &= -\frac{1}{2} CFL (\sin(m\Delta x) + \sin(n\Delta y)) \end{aligned}$$

The amplification matrix given by 5.22 can be approximated as:

$$G_{CFL \rightarrow \infty} \simeq -\frac{Z_R - Z_R^{1st}}{(Z_R^{1st})^2 + Z_I^2} (Z_R^{1st} - iZ_I)$$

By observation of the eigenvalues footprint for this case (see Figure 5.2(b)), the maximum eigenvalue lies on the real axis, i.e., the imaginary part of this maximum eigenvalue is cancelled, which means that $\sin(m\Delta x) = -\sin(n\Delta y)$, and then $m\Delta x = -n\Delta y$. The Fourier spatial operators remain:

$$\begin{aligned} Z_R &= -\mu_4 CFL (\cos(m\Delta x) - 1)^2 \\ Z_R^{1st} &= \frac{CFL}{2} (\cos(m\Delta x) - 1) \\ Z_I &= 0 \end{aligned}$$

	RK5	Implicit $\alpha = 0^\circ$	Implicit $\alpha = 45^\circ$
$CFL \sim 1$	$1 - \mu_4 CFL (m\Delta x)^4$	$1 - \frac{1}{4} CFL^2 (m\Delta x)^2$	$1 - \frac{\mu_4}{4} CFL (m\Delta x)^4$
$CFL \rightarrow \infty$		$1 - 2\mu_4$	$1 - \mu_4 (m\Delta x)^2$

Table C.1: Comparison of the asymptotic spectral radius of the amplification matrix for different solvers.

Under this assumption, the maximum eigenvalue becomes:

$$|\lambda_{max}| \simeq -\frac{Z_R - Z_R^{1st}}{Z_R^{1st}} \simeq 1 - \frac{Z_R}{Z_R^{1st}}$$

Also the maximum eigenvalue of the amplification matrix is presented for long-length wave, that is, $m\Delta x \ll 1$, and after some algebra becomes:

$$|\lambda_{max}| \simeq 1 - \mu_4 (m\Delta x)^2$$

which indicates that the spectral radius increases with the square of the number of grid points.

Implicit method based on a first-order Jacobian, $\alpha = 45^\circ$ and $1 \ll CFL \ll \infty$:

Finally, to study the last case, we can make use of the assumption of $m\Delta x = -n\Delta y$. Then, the amplification matrix given by 5.22 can be approximated as:

$$G \simeq 1 + \frac{Z_R}{1 - Z_R^{1st}}$$

as the maximum eigenvalue is presented for low frequency modes, $Z_R^{1st} \ll 1$, then the amplification matrix is:

$$G \simeq 1 + Z_R$$

and substituting the operator corresponding to this case, the maximum eigenvalue of the amplification matrix becomes:

$$|\lambda_{max}| \simeq 1 - \frac{\mu_4}{4} CFL (m\Delta x)^4$$

obtaining similar trends than for the explicit Runge-Kutta scheme.

All the asymptotic limits derived in this appendix is summarised in Table C.1.

Appendix D

The non-dimensional Jacobian matrices for the Euler equations

The Euler equations are derived from the RANS equations neglecting the viscous terms. The stability analysis of the linearised Euler equations is performed in three-dimensions assuming a periodic boundary conditions in the x , y and z directions. Without loss of generality we assume as well that the component of the velocity in z direction is null. The linearised Euler equations about a mean uniform flow can be written as:

$$\frac{\partial \mathbf{u}}{\partial t} + A_x \frac{\partial \mathbf{u}}{\partial x} + A_y \frac{\partial \mathbf{u}}{\partial y} = 0$$

where A_x and A_y are the Jacobian matrices. This set of equations can be expressed in dimensionless form, using the speed of sound.. In this case the non-dimensional Jacobian matrices become:

$$A_x = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ M^2 \left(\frac{\gamma-1}{2} - \cos^2 \alpha \right) & M(3-\gamma) \cos \alpha & -M(\gamma-1) \sin \alpha & 0 & \gamma-1 \\ -M^2 \cos \alpha \sin \alpha & M \sin \alpha & M \cos \alpha & 0 & 0 \\ 0 & 0 & 0 & M \cos \alpha & 0 \\ -M \cos \alpha \left(\frac{1}{\gamma-1} + \frac{2-\gamma}{2} M^2 \right) & \left(\frac{1}{\gamma-1} + \frac{M^2}{2} \right) - (\gamma-1) M^2 \cos^2 \alpha & -(\gamma-1) M^2 \cos \alpha \sin \alpha & 0 & \gamma M \cos \alpha \end{bmatrix} \quad (D.1)$$

$$A_y = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ -M^2 \cos \alpha \sin \alpha & M \sin \alpha & M \cos \alpha & 0 & 0 \\ M^2 \left(\frac{\gamma-1}{2} - \sin^2 \alpha \right) & -M(\gamma-1) \cos \alpha & M(3-\gamma) \sin \alpha & 0 & \gamma-1 \\ 0 & 0 & 0 & M \sin \alpha & 0 \\ -M \sin \alpha \left(\frac{1}{\gamma-1} + \frac{2-\gamma}{2} M^2 \right) & -(\gamma-1) M^2 \cos \alpha \sin \alpha & \left(\frac{1}{\gamma-1} + \frac{M^2}{2} \right) - (\gamma-1) M^2 \sin^2 \alpha & 0 & \gamma M \sin \alpha \end{bmatrix} \quad (D.2)$$

The matrices $|A_x|$ and $|A_y|$ used to construct the artificial viscosity terms are obtained departing from a diagonal matrix containing the absolute values of the eigenvalues of $|A_x|$

and $|A_y|$.

$$\begin{aligned} |\Lambda_x| &= \text{diag}(M \cos \alpha, M \cos \alpha, M \cos \alpha, M \cos \alpha + 1, M \cos \alpha - 1) \\ |\Lambda_y| &= \text{diag}(M \sin \alpha, M \sin \alpha, M \sin \alpha, M \sin \alpha + 1, M \sin \alpha - 1) \end{aligned} \quad (\text{D.3})$$

The matrices $|A_x|$ and $|A_y|$ are obtained using the transformation matrices, T_x and T_y

$$|A_x| = T_x |\Lambda_x| T_x^{-1} \text{ and } |A_y| = T_y |\Lambda_y| T_y^{-1} \quad (\text{D.4})$$

where

$$T_x = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ M \cos \alpha & 0 & 0 & M \cos \alpha + 1 & M \cos \alpha - 1 \\ M \sin \alpha & 0 & -1 & M \sin \alpha & M \sin \alpha \\ 0 & 1 & 0 & 0 & 0 \\ \frac{M^2}{2} & 0 & -M \sin \alpha & \frac{1}{\gamma-1} + \frac{M^2}{2} + M \cos \alpha & \frac{1}{\gamma-1} + \frac{M^2}{2} - M \cos \alpha \end{bmatrix} \quad (\text{D.5})$$

$$T_y = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & M \cos \alpha & 1 & M \cos \alpha & M \cos \alpha \\ 0 & M \sin \alpha & 0 & M \sin \alpha + 1 & M \sin \alpha - 1 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & \frac{M^2}{2} & M \cos \alpha & \frac{1}{\gamma-1} + \frac{M^2}{2} + M \sin \alpha & \frac{1}{\gamma-1} + \frac{M^2}{2} - M \sin \alpha \end{bmatrix} \quad (\text{D.6})$$