Investigating Castillo-Grone’s Mimetic Difference Operators in Development of Geophysical Fluid Dynamics Models Implemented on GPGPUs

by

Mohammad Abouali

A Dissertation
Submitted to the Faculty of
San Diego State University
and
Claremont Graduate University

In Partial Fulfillment of
the Requirements for the Degree of
Doctor of Philosophy
in
Computational Science

San Diego, California

Approved by:

Jose E. Castillo

© 2014 Mohammad Abouali
All Rights Reserved
We, the undersigned, certify that we have read this dissertation by Mohammad Abouali and approve it as adequate in scope and quality for the degree of Doctor of Philosophy.

Dissertation Committee:

[Signature]
Dr. Jose E. Castillo, Chair

[Signature]
Dr. Gustaaf Jacobs Member

[Signature]
Dr. Peter Blomgren, Member

[Signature]
Dr. Ali Nadim, Member

[Signature]
Dr. Allon Percus, Member
Dedication

Dedicated to Gohnaz, Parvin, Azadeh, and Hossein
Abstract of the Dissertation

Investigating Castillo-Grone’s Mimetic Difference Operators in Development of Geophysical Fluid Dynamics Models Implemented on GPGPUs

by

Mohammad Abouali

San Diego State University and Claremont Graduate University

This research investigates the performance of Castillo-Grone’s Mimetic (CGM) difference operators in 2D and 3D fully curvilinear grids. A software package is provided that assists the user in generating Laplacian operator in fully curvilinear grids using higher order CGM operators. Furthermore, a stability analysis is performed using von Neumann’s analysis to understand the behavior of these operators in solving advection equation. A shallow water equation solver is provided that harnesses the power of the general purpose graphics processing units (GPGPUs) and some timing are provided.
Acknowledgements

First and foremost I would like to thank my advisor, Prof. Jose E. Castillo, for his support, constructive comments and encouragement. I like to thank him for our discussions during coffee and lunch breaks and all the time he had offered me. I am also deeply grateful to my graduate committee members, Prof. Peter Blomgren, Prof. Gustaaf (Guss) Jacobs, Prof. Ali Nadim, and Prof. Allon Percus; this research was not possible without their helpful comments and suggestions. I would also like to thank all my committee members for sharing their time and evaluating this dissertation.

I would like to thank all my colleagues and classmates, especially those that we were sharing the same office, and yes, we spent many hours, days and nights. Discussions we had were very insightful and made my stay at San Diego State University (SDSU) more memorable.

I also like to thank Parisa, and Charlotte at SDSU and Claremont Graduate University (CGU). I want to thank them for their patience, helpful comments, and assistance to maneuver through the complicated procedure of applying for admission, graduation, and all the paperwork that involved within.

There are many people that I would like to thank, as each person contributed to this dissertation by their own unique way. I want to thank David, Heidi, Joris, Joris (not a mistake, there were two Jorises), Christiaan, Jan-Jaap, Joshua, Eduardo, Dany, Micah, Omar, Angie, Mary, Behrouz, Reza, Marjan, Saeedeh; and I do apologize if I forget to mention some others, but I want you to know that I cherish all the good moments and I will always keep you dear. You guys are awesome.

I would also like to thank Ali, Sara, Leila, and Rojin. We spent numerous hours at each other accompany. I enjoyed our scientific discussions, hours that we spent
at the library, movie nights, cooking, partying. I am very happy to have friends like them.

I also like to thank Prof. Bernard Geurts and Dr. David Dietrich for their suggestions and sharing their insights in fluid mechanic and numerical modeling.

The last, I want to thank my family: my parents, Parvin and Hossein, and my sister, Azadeh, for their supports and encouragement throughout my whole life; and my beautiful wife, Golnaz Badr, for her endless and unconditional love; without their encouragement I was never able to accomplish this task. I want to thank them for standing by me and believing in me even when I didn’t. I am blessed to have you all in my life.

Mohammad Abouali

April 2014
## Contents

### I  Premier

#### 1 Introduction

1.1 Motivation and Objectives ........................................... 9  
1.2 Outline ........................................................................... 10

#### 2 Unified Curvilinear Ocean Atmosphere Model (UCOAM)  

2.1 Introduction ................................................................. 12

2.2 UCOAM Model Details .................................................... 17

2.2.1 Governing Equations ............................................... 17

2.2.2 Transforming Into Curvilinear Coordinates ....................... 19

2.2.3 Numerics ................................................................. 20

2.2.4 Pressure ................................................................. 21

2.3 Test Cases: Neutral Conditions ....................................... 24

2.3.1 Grid Generation ......................................................... 24

2.3.2 Boundary and Initial Conditions ................................. 26

2.3.3 Results and Discussion ............................................. 26

2.4 Another Test Case: Temperature Profile Interacting with The Seamount  

2.5 Conclusions ............................................................... 32
3 Regridding Data: A Package to Interpolate, Extrapolate, and Fit a Curve

3.1 Introduction
3.2 Governing Equations
3.3 MATLAB Command
3.4 Results
3.5 Software Availability

II Investigating CGM Operators

4 Accuracy of CGM Operators in 2D and 3D Curvilinear Domains

4.1 Introduction
4.2 Poisson’s Equation in Fully Curvilinear Grids
4.3 Castillo-Grone’s Mimetic (CGM) Difference Operators
4.4 Discretization
  4.4.1 Using Cell Centered Finite Difference
  4.4.2 Using the CGM Difference Operator
  4.4.3 A 2D Example
  4.4.4 A 3D Example
4.5 Conclusions and Recommendations

5 Stability Analysis

5.1 Introduction
5.2 Numerical Approach
  5.2.1 Governing Equation
  5.2.2 Initial and Boundary Conditions
  5.2.3 Spatial Discretization
5.2.4 RK3 Scheme ......................................................... 63
5.3 Von Neumann Stability Analysis ...................................... 63
5.4 Results and Discussions ............................................... 66
5.5 Using Other Time Schemes .............................................. 69
5.6 Conclusion ............................................................... 72

6 Solving Poisson’s Equation On A Curvilinear Grid Using High-Order Mimetic Methods

6.1 Introduction .............................................................. 74
6.2 Methods ................................................................. 76
   6.2.1 Poisson’s Equation in Curvilinear Grids ......................... 76
   6.2.2 Castillo-Grone’s Mimetic (CGM) Difference Operators ........ 77
   6.2.3 Computational Grids in 2D Transformed Poisson’s Equations . 79
   6.2.4 MATLAB® Command ............................................... 81
6.3 Results and Discussion in 2D Cases ................................. 85
6.4 Conclusions in 2D ..................................................... 100
6.5 3D Curvilinear Grids ................................................... 101
6.6 Results in 3D ............................................................ 102
6.7 Software Availability .................................................. 109

III Applications in Geophysical fluid models 111

7 Shallow Water Equations Implemented on GPUs  

7.1 Introduction & Governing Equations ............................... 112
   7.1.1 Continuity Equation ............................................... 112
   7.1.2 Momentum Equations .............................................. 114
   7.1.3 De-singularizing .................................................. 114
## List of Figures

1.1 Energy Spectrum. ................................................. 5  
1.2 Physical domain transformed into computational domain. ....... 7  
2.1 Arakawa A, B, C, D, and E grids. ................................ 13  
2.2 Energy of the System using method I for pressure. ............... 23  
2.3 Barotropic over Kinetic Energy using method I. .................. 24  
2.4 Barotropic over Kinetic Energy using method II. ................ 24  
2.5 Final $C_1$ Grid. .................................................. 25  
2.6 Grid affecting the number of iterations required to solve for pressure. 27  
2.7 Contour lines for velocity along the $x$-axis passing through the zenith of the seamount using $C_1$. Slide 100 is at $t = 20000$; distances are in kilometers. .................................................. 27  
2.8 Contour lines for velocity along the $x$-axis passing through the zenith of the seamount using $S_1$. Slide 100 is at $t = 20000$; distances are in kilometers. .................................................. 28  
2.9 Changes of vertical velocity share of the total energy. .......... 28  
2.10 Difference of vertical velocity share of energy between $S_1$ and $C_1$ on plane passing through the crest of the seamount; distances are in kilometers. .................................................. 29  
2.11 Difference of vertical velocity share of energy between $S_1$ and $C_1$ on plane perpendicular to $y$-axis at $y = -150m$; distances are in kilometers. .................................................. 29  
2.12 Difference of vertical velocity share of energy between $S_1$ and $C_1$ on plane perpendicular to $y$-axis at $y = +150m$; distances are in kilometers. .................................................. 30  
2.13 Horizontally-averaged temperature profile. ...................... 32  
2.14 Horizontally averaged density profile. .......................... 32  
2.15 Changes in $T - \bar{T}$. ............................................. 32  
3.1 Interpolating from cell corners to cell centers. .................... 42  
3.2 A sample source and destination grid with scattered points. ..... 42  
3.3 Interpolation results for scattered points. ........................ 43  
3.4 Effect of nInterp on interpolation accuracy. ........................ 44  
4.1 Staggered computation of the CGM gradient operator in 1D. ....... 50  
4.2 Staggered computation of the CGM divergence operator in 1D. .... 50  
4.3 subindex numbering in 2D curvilinear grid. ........................ 53  
4.4 error comparison in 2D curvilinear grid. .......................... 55  
4.5 3D Curvilinear Grid. .............................................. 57  
4.6 Error comparison in 3D grid. ..................................... 57
5.1 Amplification factor using the fourth-order accurate CGM divergence operator with linear interpolation and the RK3 as the time discretization scheme. .................................................. 66
5.2 Sample solution using Kawamura scheme with $Cr = 0.6$. ................. 67
5.3 Sample solution using second-order accurate CGM divergence operator with fourth-order interpolation scheme, $Cr = 1.2$ (left), and sixth-order interpolation scheme, $Cr = 1.1$ (right). ............................................. 68
5.4 CGM gradient, where $f$ denotes the locations that the data must be held, and $Gf$ shows the location where the gradient is calculated using the CGM gradient operator. .................................................. 69
5.5 V1 Approach (left), data is located at the cell centers only. Using the boundary condition, data is interpolated at the boundary nodes to satisfy the grid requirements of the CGM gradient operator. The gradient is then calculated at the nodes and linearly interpolated to the cell centers. V2 Approach (right), data is interpolated to all the interior nodes. The CGM gradient operator is used to calculate the gradient at the cell centers directly. Hence, there is no need to interpolate the calculated gradient. .................................................. 70
5.6 Second-order accurate CGM gradient operator with fourth-order interpolation scheme using the V1 approach, $Cr = 1.65$ (left); using the V2 approach, $Cr = 1.2$ (middle); using the V3 approach, $Cr = 1.2$ (right). 70
5.7 All methods compared together. .................................................. 71
6.1 Staggered computation of the CGM gradient operator in 1D. ................. 78
6.2 Staggered computation of the CGM divergence operator in 1D. ............ 78
6.3 Staggered computation of the CGM gradient operator in 2D. ............... 78
6.4 Staggered computation of the CGM divergence operator in 2D. ............ 79
6.5 2D CGM grid in computational space and numbering. ....................... 82
6.6 2D U grid in computational space and numbering. .......................... 82
6.7 2D V grid in computational space and numbering. ........................... 82
6.8 2D $\eta$ grid in computational space and numbering. ....................... 83
6.9 2D $\xi$ grid in computational space and numbering. ....................... 83
6.10 2D Nodal grid in computational space and numbering. .................... 85
6.11 Boundary’s normal vectors pointing outward. .................................. 86
6.12 Sample numerical and analytic solution using fourth order operators. .... 88
6.13 Changes of error by increasing operator order. .............................. 89
6.14 Global Operator Order .................................................. 90
6.15 Error comparison between Neumann and Dirichlet boundary condition. 91
6.16 Sample numerical and analytic solution using fourth order operators for equation (6.25). .................................................. 92
6.17 The accuracy improves by increasing the operator order when Robin boundary condition is used. .................................................. 93
6.18 Improvement in the accuracy as the grid is refined once Robin boundary condition is used. .................................................. 94
6.19 A hard sample problem with lots of oscillations. ............................ 96
6.20 Diagonal profile of the hard sample sample case and the error. .......... 96
6.21 A Coarse Curvilinear Grid. .................................................. 97
6.22 Error variation by operator orders in curvilinear grid. ....................... 97
6.23 Sample Noisy Grid ($SNR=25$). .................................................. 98
6.24 Numerical and analytic solutions on the sample noisy grid. ............... 98
6.25 CGMGrid and node numbering. .................................................. 102
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.26</td>
<td>$U_{\text{Grid}}$, $U_{\text{Grid}}$, and $U_{\text{Grid}}$ grids and node numbering.</td>
<td>103</td>
</tr>
<tr>
<td>6.27</td>
<td>$\xi\eta$ and $\xi\zeta$ grids and node numbering.</td>
<td>104</td>
</tr>
<tr>
<td>6.28</td>
<td>$\eta\xi$ and $\eta\zeta$ grids and node numbering.</td>
<td>105</td>
</tr>
<tr>
<td>6.29</td>
<td>$\zeta\xi$ and $\zeta\eta$ grids and node numbering.</td>
<td>106</td>
</tr>
<tr>
<td>6.30</td>
<td>An example of 3D curvilinear grid and the solution.</td>
<td>108</td>
</tr>
<tr>
<td>7.1</td>
<td>Shallow Water Equation (SWE) variables in vertical profile.</td>
<td>113</td>
</tr>
<tr>
<td>7.2</td>
<td>Changes of error by increasing operator order.</td>
<td>125</td>
</tr>
<tr>
<td>7.3</td>
<td>Contour lines for the water table in Uniform flow test.</td>
<td>126</td>
</tr>
<tr>
<td>7.4</td>
<td>Orthogonal Rotated Domain.</td>
<td>127</td>
</tr>
<tr>
<td>7.5</td>
<td>Changes of error by increasing operator order.</td>
<td>130</td>
</tr>
<tr>
<td>7.6</td>
<td>Simulating a channel with a dry zone in the middle, $C_z = 35$, $t = 86400$ [s] $= 24$ [hr], $dx = dy = 50$ [m].</td>
<td>131</td>
</tr>
<tr>
<td>7.7</td>
<td>Simulating a channel with a dry zone in the middle, $C_z = 20$, $t = 7200$ [s] $= 2$ [hr], $dx = dy = 10$ [m].</td>
<td>132</td>
</tr>
<tr>
<td>7.8</td>
<td>Monterey Bay Bathymetry.</td>
<td>133</td>
</tr>
<tr>
<td>7.9</td>
<td>Water table in the Monterey bay with flat bathymetry.</td>
<td>133</td>
</tr>
<tr>
<td>7.10</td>
<td>Changes in Water table at three points during 24 hours simulation.</td>
<td>134</td>
</tr>
<tr>
<td>8.1</td>
<td>Time evolution of a Gaussian function advected through the domain.</td>
<td>137</td>
</tr>
<tr>
<td>8.2</td>
<td>Time evolution of a Gaussian function advected through the domain.</td>
<td>138</td>
</tr>
<tr>
<td>8.3</td>
<td>Node numbering.</td>
<td>139</td>
</tr>
<tr>
<td>8.4</td>
<td>Amplification Factor.</td>
<td>141</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Summary of the Grids .................................................. 26
4.1 Comparison between number of required iterations and the existing code. .................................................. 55
5.1 Stability using other time discretization schemes in solving conservative and non-conservative advective equation. .................................................. 72
5.2 Stability in solving acoustic equations. ........................................ 72
6.1 Required time in seconds and number of non-zero operator. ........... 86
6.2 RMSE on noisy grid using the minimum required number of nodes for interpolation. .................................................. 99
6.3 RMSE on Noisy Grid using more nodes than the minimum required for interpolation. .................................................. 100
6.4 Effect of operator order in 3D regularly spaced cubes in Example 1. 104
6.5 Effect of operator order in 3D regularly spaced cubes in Example 2. 106
6.6 Effect of operator order in 3D regularly spaced cubes in Example 2 using a higher resolution grid. .................................................. 107
6.7 Effect of operator order in 3D curvilinear grid. .......................... 108
6.8 Effect of operator order in 3D curvilinear grid with higher resolution. 108
6.9 Effect of interpolation order on the operator in 3D curvilinear grids. 110
7.1 Timing the interpolation kernel on GPU (20 interpolation before data transform). .................................................. 122
7.2 Timing the interpolation kernel on GPU (40 interpolation before data transform). .................................................. 123
7.3 Timing SWE solver. .................................................. 123
7.4 Timing SWE Solver on GTX 480. .................................................. 123
Part I
Premier
Chapter 1
Introduction

The curious nature of human being is always an incentive to investigate the flow patterns in the Atmosphere and in the water bodies [121]. For a long time, our understanding of the nature was based only on the field measurements. These measurements were usually sporadic and spatially scattered. Using Satellite Remote Sensing (RS) techniques helped us to overcome the temporal and spatial issues in ground base measurements; but only to some extends, as they deliver either high spatial resolution or high temporal resolution.

It is not far from the truth to say that nowadays, our understanding of the world around us highly depends on the outputs and the results of the numerical models [85]. The result of the numerical models, combined with the observations, are interpreted to obtain information about the nature, decide which path to take, what project to fund, how to embrace against certain events or a natural disaster, and if not being able to completely alleviate the damage, how to minimize it. Therefore, the accuracy of these numerical models is of utmost importance as they play a vital role in every single steps and decisions [47, 48].

Navier-Stokes’ (NS) equations form the basis in any numerical model that deals with flow pattern in fluids, or in this case geo-physical fluids [47, 48, 122]. Claude-Louis Navier and George Gabriel Stokes established the NS equations in 18th century
to describe the flow pattern of fluid substances by applying the second law of Newton
to fluids. NS equations can be used in the study of atmospheric and oceanic flows,
pipe flows, river flows, designing aircraft, cars, wind turbines, jet engines, studying the
dispersion of the pollutants, distribution of the nutritious substances in the oceans,
cloud formations, and many more field. NS equations are also very important from
simply the mathematical viewpoint. Despite its high importance, virtually in any
field, mathematicians have not yet provided any proof of the existence and smoothness
of the solution in three dimensions. In fact, Clay Math Institute recognizes this as one
of the most important problem and offers one million dollar for a proof or a counter
proof.

There are several methods to solve NS equations, the most accurate one being
Direct Numerical Simulation (DNS) [45, 48, 59]. In DNS, the space is discretized
and the NS equation is solved directly. However, DNS requires very fine grid, down
to dissipation scales, in order to provide valid results. It has been shown that the
number of required grid points is proportional to $R_e^{9/4}$, where $R_e$ is the Reynolds
number [5, 106]. This grid requirement of DNS, makes it impractical and unfeasible
for geophysical fluid [45, 47, 48]; since, both the area is large and the Reynolds number
is way higher than $10^6$. Even these days with the most advanced supercomputers,
DNS simulations are restricted to purely research studies in moderate or less complex
geometries. It is not just the lack of enough memory to address such a fine resolution.
The problem also arises from the fact that when the grid spacing goes down, a smaller
time step is required to keep the numerical model stable. Hence, more number of
iterations is required to cover the same time-span.

In summary, DNS is the most accurate method and it requires the least amount of
computation per iteration per grid nodes. However, the number of required iterations
and nodes are so high that at the end it becomes the most expensive method and in
many cases even unfeasible.

Reynolds Averaged NS (RANS) or better to say Unsteady RANS (URANS) addressed the high computational demand of the DNS method by decomposing the velocities and other variables into their mean part and their fluctuation in time [48]. In simple words, velocities and other fields are averaged over time.

This approach makes the application of NS simulation in geo-physical fluids feasible; however, due to the non-linear terms in the NS equation, i.e. the advection terms, it is not possible to completely alleviate the needs of fluctuation part [48, 122]. As a result, there would be still some cross products of different fluctuation terms. It can be shown mathematically that these cross products show how the two variables are correlated [9, 10, 48, 122].

There are different approaches to evaluate these cross products based on the simulated mean value. These methods, generally known as Turbulence Model, are required to close the system of the equations and they play the role of the closure in the mathematical terminology [122]. Mellor-Yamada [86], k- [15, 64, 75, 135], and k-[87, 88, 134, 135] are a few examples.

Although the RANS approach is computationally the least demanding; however, it is highly dependent on the turbulence model. These turbulence models are not universal; neither they are scale dependent, meaning that they model all the scales of the flows and they need tuning for each different application [9, 10, 47].

The middle approach is the use of the Large Eddy Simulation (LES) technique to solve NS equations. In LES, the variables are decomposed into their mean part and fluctuation part based on a general spatial filter [5, 48, 106, 112]. In simple words, the velocity and other fields are averaged based on a chosen length scale. Any structure larger than this length scale is simulated directly using the NS equation and those structures smaller than the length scale are modeled using a so-called Sub-Grid Scale
The logic behind this filtering becomes evident, once we look at the energy spectrum of the fluid [48, 112]. The energy spectrum curve can be divided in three sections, Figure 1.1. First section on the left represents large-scale features that are problem dependent and changes based on the forcing, geometry and other factors specific to the problem. The third section, on the right part of the spectrum, represents the viscous range and very small structures. The middle part of the spectrum is known as the inertial range. Regardless of the problem conditions, it quite as nearly follows the five third power law. This implies that the smaller structures of the flow field can be modeled based on the resolved larger structures [48, 108, 112]. It has to be noted that it is the chosen length scale, which defines what structure is large, i.e. what is going to be resolved and what is going to be modeled using SGS, and the term large eddies here has nothing to do with the real large structures that exist in geo-physical fluids. Indeed, this is the length scale, which determines both the cost and the accuracy of LES simulations. The cost and the accuracy of LES vary between those of the RANS and DNS based on the selected length scale.

![Energy Spectrum](image)

Figure 1.1: Energy Spectrum.

As it was said, DNS approach is too expensive and even unfeasible for geo-physical
fluids and it is believed to be the same case for the next couple of decades [47]. RANS methods are still dominant, particularly in the ocean models. However, LES approach is already being used widely in the atmospheric studies [50, 126] and it is believed to be the approach for the next generations of the coastal ocean models, where very high resolution is required [47, 110, 129].

There is an ever-growing demand for higher resolution models in environmental studies. Particularly with the advances in the available computational power and memory, the majority of the research efforts are focused on the development of the models that are capable of handling the requirements needed for this resolution. The model requirement changes based on its resolution. As an example, pressure is commonly assumed to be hydrostatic, i.e. it is assumed that the pressure at each point is equal to the weight of the fluid above it [38, 43, 44]. This assumption has been used successfully for quite a while in many ocean and atmosphere models [52]. There are studies showing that for horizontal grid resolution of 1km by 1km, the hydrostatic assumption still holds and the deviation is negligible [43]. However, the finer the resolution it gets, the more pronounced these deviations are. There are some quasi-hydrostatic models for intermediate resolutions, and for very fine resolutions, such as few meter in horizontal, the fluid must be assumed to be fully non-hydrostatic [79, 81]. One of the requirements for any LES model is to solve for pressure using a non-hydrostatic approach; therefore, this was another motivation to use LES approach.

It is not just the pressure that is affected by the resolution. Even stratification in the oceans is a scale dependent assumption. While in a 1km wide grid cell it can be assumed that the ocean is stratified, in a grid cell with few meter of horizontal resolution this assumption does not hold any more. Particularly, at the interface of the rivers and the coastal regions, extra care must be taken into account [110].
In coarse resolution modeling everything is smooth and the high-frequency features, or the small structures are removed. However, these small structures have a profound effect on the accuracy and the quality of the coarse resolution models. These small-scale effects are incorporated into the coarse models using a sort of parameterizations [52, 119, 136]. Hence, it is crucial to tune these parameters depending on the region. When performing high resolution modeling, the small features are preserved, particularly the changes in the topography. In coarse resolution models the topography is smooth, while in finer resolution models, more small-scale features on the topography are preserved. Therefore, selecting the right coordinate system in vertical direction becomes important.

The curvilinear grids allow more flexibility in moving the grid nodes in order to achieve better grid metrics [30, 59]; hence, the better numerical results, Figure 1.2. Curvilinear grids are commonly used in the horizontal directions in many models. However, for ease of the calculation and storage, they are not used in the vertical direction. To address high-resolution studies of geo-physical fluids, and assuring the quality even over the steep slopes, a fully curvilinear coordinate system is suggested to be used in all three directions.

![Physical domain transformed into computational domain.](image)

Figure 1.2: Physical domain transformed into computational domain.

Without any doubt a numerical model must incorporate proper physics in order
to deliver acceptable numerical results. However, that is not all. It was shortly described how the grid is also affecting the results. Another major factor is the numerical scheme in use to solve the PDEs [59]. Different schemes have different requirements. Both the accuracy and the performance of a model will vary based on the discretization scheme in use.

The majority of the equations, describing physical phenomena, are written using the gradient, the divergence, and the curl operator. These operators, depending on the field that they are applied, have a physical meaning. Mimetic discretization method, as their name implies, mimics the physical property of their operator and satisfies them exactly in the discrete environment. Castillo and Grone have developed a set of mimetic operators known as Castillo–Grone’s mimetic (CGM) operators. CGM operators have been successfully used in many fields [24, 25, 26].

There are numerous studies comparing CGM operators in the regularly spaced Cartesian grids. There are few papers investigating the performance of the CGM operators in irregularly spaced Cartesian grids [16]. However, there is no study inspecting the performance of CGM operators in the fully curvilinear grids.

Another approach to improve the performance of a model and decrease the total computational time is to use several processing unit (PU) in parallel to reduce the time needed to solve for Poisson equation. There are numerous studies and packages, which use multiple Central PUs (CPUs) in parallel. When all the CPUs are on a single node (machine) openMP standard is commonly used for the programming model and when the CPUs are distributed across different nodes MPI (Message Passing Interface) programming model is generally used. It is also possible to mix openmp and MPI together.

CPUs are not the only computing nodes available on a machine. There are also many computing cores available on the graphic cards, called Graphics PU (GPU).
Although each graphics computing core has less computing power; however, there are many computing cores available on a single graphic card, 2600+ for some models. Therefore, on a single machine, it is possible to harness the power of these many graphics computing cores to achieve a certain computing tasks in much shorter time. Recently, there is a general trend towards using these GPU for general-purpose computing. Many companies, including NVIDIA, provide hardwares, known as General Purpose GPU (GPGPU), for scientific computing.

Another focus of this thesis is to harness the power of GPGPUs. GPUs have been already used successfully, in scientific computing and environmental studies. In some cases, it was possible to achieve up to 540+ times speedup by using GPUs. This motivates us to use the power of GPUs.

1.1 Motivation and Objectives

The main objective of this thesis is to investigate the Castillo–Grone’s Mimetic (CGM) operators in fully curvilinear 2D and 3D grids and how they behave. To do so, the following objective must be satisfied.

1. Implementing LES technique to solve for NS equations describing the geophysical fluids.

2. Using fully curvilinear grids in all three dimensions, including the vertical directions, to describe the physical domain, and investigating its effect on the numerical results.

3. Making use of Castillo-Grone’s Mimetic (CGM) operators in solving NS equations, particularly investigating:

   (a) How CGM operators behave in curvilinear coordinates.
(b) How they can enhance both the accuracy and the performance of the environmental models.

(c) In particular, How CGM operators enhance both the accuracy and the performance of solving a Poisson Equation to obtain non-hydrostatic pressure.

4. Harnessing the power of many computing cores available on GPGPUs and using the state of the art CUDA programming model.

1.2 Outline

In Chapter 2 the Unified Curvilinear Ocean Atmosphere Model (UCOAM) is introduced and some of the grid dependent assumptions in ocean and atmosphere models are tested. UCOAM is an LES model that solves NS equations (Objective 1 and 2).

In Chapter 3 a new software package is introduced that simplifies the task of interpolation and extrapolation. This package simplifies the entire procedure to a sparse matrix multiplication that depending on the problem could be pre-computed (extra, not part of the objectives).

In Chapter 4 the accuracy of the CGM operators in solving Poisson’s equation in 2D and 3D is investigated and a baseline of performance is established (Objective 2 and 3).

In Chapter 5 the stability of CGM operators in one dimensional space is investigated (Objective 2 and 3).

In Chapter 6 a software package is introduced that can be used to solve Poisson’s equations in 2D and 3D domains using second, fourth, or sixth-order CGM operators. The performance of the CGM operators are further investigated (Objective 2 and 3, Extra).
In Chapter 7 a shallow water equation model is introduced that make use of GPUs and cubic interpolation (objective 4).

In Chapter 8 the stability analysis is extended to higher dimensions.
Chapter 2

Unified Curvilinear Ocean Atmosphere Model (UCOAM)


2.1 Introduction

The study of atmospheric and oceanic flow patterns has challenged scientists, researchers, and traders for centuries. Although scattered measurements taken around the globe have provided a general understanding of these patterns, it was not until the recent development of more accurate numerical models that scientists have finally been able to gain a better understanding of the eddy structures within flow fields. In addition, better numerical algorithms and an increase in computing power have now made it possible to obtain vital information from areas never before breached. See McWilliams [85] for a more thorough review of the history of model development.

Beyond an increase in computing power, the availability of memory modules and higher-capacity storage devices have now made it possible to perform simulations at a much finer spatial resolution. At coarser resolutions, most of the structures
existing on the terrain are smoothed; however, with finer resolution these undulations are preserved, and the effect of the topography on the flow field can be studied in greater detail. This is significant because, although these smaller structures do not appear in coarse-resolution models, long-term, coarse-resolution simulations are strongly affected by the parameterizations of such small-scale features and processes [52, 61, 119, 137].

To represent the physical space of a model, a coordinate system is first selected. There are several coordinate systems available in both the horizontal (Arakawa A, B, C, D, and E Grids) and the vertical. For description of each Arakawa grid refer to Figure 2.1.

Most ocean models use B and C grids [12, 13, 14, 52, 90], including CANDIE [77, 116], which is based on the DieCAST model [39] and uses an A grid, and the MITGCM model [79, 80, 81], which allows use of both the standard C and C/D grids [8]. Use of an A, B, C, D, or E grid will affect the stability or accuracy of a numerical model; the choice of vertical coordinate is also important [33, 34, 41, 52, 137], as it affects most parameterization and representation issues [52]. Though the potential
density or $\rho$ coordinate is used in some ocean models [19, 52, 54, 55, 104, 114], the $z$-coordinate and the sigma-coordinate are the two most commonly used coordinate systems in the vertical direction [52]. There are numerous studies comparing these two coordinates; while some favor use of the $z$-coordinate [38, 39, 40, 52, 77, 116, 120], others favor the sigma-coordinate [13, 20, 43, 44, 53, 86, 115]. However, all the studies have one conclusion in common: whichever coordinate system is utilized in the model affects the numerical results.

In the case of the sigma-coordinate, the vertical grid-lines are kept straight, which reduces the amount of required storage and also eases the integration in the vertical direction. However, the sigma-coordinate was originally designed only for slopes up to 45 degrees [70, 71], and it is commonly known that the sigma-coordinate affects the numerical results on steep slopes [18, 52, 56, 83]. One reason for this error is the non-orthogonality of the grid in the vertical direction. In curvilinear grids, the equations or the operators need to be transformed. For example some cross derivatives are generated as a result of the non-orthogonality of the grid. These derivatives are generally ignored, and the solution decoupled in both the horizontal and the vertical, which significantly simplifies the implementation. However, in cases of steep slopes, these cross-derivatives become significant, and must therefore be addressed (see Chapter 4).

Models with coarse spatial resolution, $5 \sim 25km$, smooth the bottom bathymetry or topography, resulting in the removal of most of the undulations in the terrain, and the production of low slopes everywhere. As this will remove most of the terrain’s structure, the bathymetry’s effect on the flow is either damped or ignored. The interaction of abyssal eddies with realistic bathymetry is crucial in the simulation of significant ocean currents, including those of the Gulf Stream [62], as well as the Black Sea Rim Current and eddies which result from its interaction with major
coastal bathymetry abutments [120]. Although finer resolution models are able to preserve more of the terrain’s structures, they also cause an increase in the slope; as a result, the distribution of grid points in the vertical direction becomes important as a means of controlling grid-induced numerical errors. Also the measurements support the fact that very small features on the terrain affect the roughness properties; and subsequently, the mixing levels [107]. Researchers and oceanographers have shown that small features in the topography can also affect the dissipation rate, general circulation, and internal waves [17, 37, 49, 76, 123, 138].

Despite the strong effect these small features have on the flow field, the momentum equation regarding the vertical component of the velocity in geophysical fluids is commonly disregarded. The horizontal scale is much larger than the vertical scale in oceanic and atmospheric flows; hence, the magnitude of the vertical component of the velocity is much smaller than the magnitude of the two horizontal components of the velocity. In order to free up storage space, the vertical velocity is not kept for further processing. Because of this, the majority of ocean and atmosphere models do not solve for the momentum equation for the vertical component of the velocity, but rather utilize a technique that uses changes in pressure and density to estimate the vertical velocity. In addition, removing the vertical momentum equation makes it possible to choose a larger time step, while maintaining a stable scheme. As mentioned above, in high-resolution modeling, the effect of the bottom bathymetry on the velocity field is significant and must be calculated as accurately as possible [124, 125].

Primitive Navier-Stokes equations under Boussinesq (NS) approximations are widely used in numerical studies of geophysical fluids [81, 86, 113, 122]. Computationally less demanding, the Reynolds-averaged Navier-Stokes (RANS) method has been used to solve NS equations for decades [5, 45, 48, 122, 135]. Conversely, large-eddy simulations (LES) provide more accurate results, but at a higher computational
cost [5, 48, 106, 112]. LES models are widely used in atmospheric boundary layer studies, although more so in studies of flat surfaces than in applications over topography [9, 10, 50, 126]. RANS formulation is still predominantly used for oceanic simulations; however, due to advancements in computational power and memory, LES modeling of oceanic flow is now possible [110, 129]. However, LES is still computationally too expensive to be used for regional and global ocean modeling, and is anticipated to remain so for the foreseeable future [47], limiting LES usage in ocean modeling to ultra-high-resolution studies with a horizontal resolution of just a few meters (such as oceanic flows near the coast or at the interface of estuaries connected to oceans and small bays, such as San Diego bay). Therefore, the term large in LES is scale-dependent, and is not related to large eddies within the ocean, which are studied by general circulation models.

In this chapter, the Unified Curvilinear Ocean Atmosphere Model (UCOAM) is introduced. UCOAM is a LES model capable of working with curvilinear grids in all three dimensions, with either sigma- or non-sigma coordinates in the vertical direction. In addition, the grid can be orthogonal or non-orthogonal in all directions, including the vertical. Moreover, neither the z-coordinate nor the sigma-coordinate is capable of handling geometries with non-convexity in the vertical direction, but because UCOAM uses general curvilinear coordinates in all three directions, it is able to handle such geometries and perform the simulations. UCOAM solves all three momentum equations, including that of the vertical momentum, and is fully non-hydrostatic. It is also designed to have a horizontal resolution of only a few meters, making it unsuitable for global ocean circulation modeling. It is thus best adapted for use in ultra-high-resolution modeling, such as the study of strong current interaction in small-scale coastal regions with steep features, the effects of curved boundaries on turbulence, or the effects of bottom roughness.
2.2 UCOAM Model Details

2.2.1 Governing Equations

UCOAM uses the Large-Eddy Simulation (LES) approach to solve the fully three-dimensional primitive Navier-Stokes (NS) equation with Boussinesq’s approximation. As in all LES models, it is assumed that the large structures of the flow are affected by both the geometry and the specific conditions of problem, while the small structures are universal and can be modeled globally. These variables are called filtered variables, and are decomposed into both their mean and fluctuation parts. After non-dimensionalizing and substituting these filtered variables in the NS equation [5, 6, 48, 112, 122], the final, non-dimensional momentum equation used in UCOAM is:

\[
\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} + \frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} - \frac{1}{Ro}\epsilon_{ij3} u_j + \beta \delta_{i3} = 0
\] (2.1)

In the above equation, \( u_i \) is the mean part of the velocity; \( Ro \) is the Rossby number, taking into account the rotation of the Earth, and defined as follows:

\[
Ro = \frac{U}{Lf},
\] (2.2)

and \( f \) is the Coriolis factor, defined as:

\[
f = 2\Omega \sin(\phi),
\] (2.3)

where \( \Omega \) is the angular speed of the Earth and \( \phi \) is the latitude. \( \beta \), in NS equations, is the buoyancy factor. In oceanic flows, the buoyancy factor is defined as [82, 113, 122]:

\[
\beta = \frac{\hat{\rho}}{\rho^*},
\] (2.4)
where \( \dot{\rho} = \rho - \rho_* \) is the deviation from the standard or mean density. In oceanic flows, density is a function of the pressure, salinity, and temperature, called the equation of state (EOS). UCOAM uses the UNESCO equation of state, which is widely accepted in the oceanography community and has an error of \( 3.5 \times 10^{-3} \text{Kg/m}^3 \) in typical oceanic conditions [46, 84, 91, 92, 128]. In the atmospheric flows, using the gas law [122] the buoyancy factor can be written as:

\[
\beta = \frac{\dot{\theta}}{\theta_*}.
\] (2.5)

Again \( \dot{\theta} = \theta - \theta_* \) is the deviation from the standard temperature, and \( \theta \) is the potential temperature. The sub-grid stress tensor, \( \tau_{ij} \), is calculated using the Smagorinsky model [118], i.e.:

\[
\tau_{ij} = -2\nu_T \bar{S}_{ij},
\] (2.6)

where the turbulent eddy viscosity is set to:

\[
\nu_T = (C_s \ell)^2 \sqrt{2\bar{S}_{ij}\bar{S}_{ij}}
\] (2.7)

\( C_s \) is the Smagorinsky constant and ranges from 0.08 to 0.22 [9, 10]. \( \ell \) is the length scale, set to \( \ell = (\Delta_x \Delta_y \Delta_z)^{\frac{1}{3}} \), and \( \bar{S}_{ij} \) is the strain-tensor, defined as:

\[
\bar{S}_{ij} = \frac{1}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}).
\] (2.8)

There are more advanced and dynamic sub-grid scale models (the Smagorinsky model is known for being too dissipative close to the solid boundaries). However, studies show that dynamic models do not perform well for large scale flows (e.g., oceanic or atmospheric flows), and the Smagorinsky model’s performance, with some
changes close to the boundary, is acceptable for use with such flows [10, 22, 110]. Consequently, UCOAM also uses the Smagorinsky model.

Changes in density will result in the generation of buoyancy forces, which affects the velocity field. Since UNESCO EOS determines density as a function of temperature, salinity, and pressure, changes in temperature and salinity affect the flow field. Hence, the temperature and salinity are considered active scalars in UCOAM, and are computed as follows:

\[
\frac{\partial(\theta, S)}{\partial t} + u_j \cdot \frac{\partial(\theta, S)}{\partial x_j} = -\frac{\partial \tau_j^{(\theta, S)}}{\partial x_j} + Q^{(\theta, S)} \tag{2.9}
\]

where \( Q^{(\theta, S)} \) is the source/sink term; and the sub-grid model is defined as:

\[
\tau_j^{(\theta, S)} = -\frac{\nu_T}{(Pr, Sc)} \frac{\partial(\theta, S)}{\partial x_j} \tag{2.10}
\]

where \( Pr \) is the Prandtl number and \( Sc \) is the Schmidt Number. The Prandtl number for 10°C seawater is approximately 10, and the Schmidt number is typically half the Prandtl number.

2.2.2 Transforming Into Curvilinear Coordinates

Unlike other ocean models, the vertical gridlines in UCOAM are allowed to curve like the horizontal gridlines, giving UCOAM more flexibility to distribute the grid nodes along the vertical line in order to reduce grid-induced errors. This means that in the vertical direction, UCOAM is capable of using a grid that is both terrain-following and curvilinear, allowing UCOAM to work with both orthogonal and non-orthogonal grids in both the vertical and horizontal directions. Problems encountered with use of the general curvilinear coordinate in UCOAM are: (1) larger amounts of storage are needed to store and represent the grid, and (2) a more difficult calculation of
derivatives is required. With sigma-coordinates one need only store the manner in which the nodes are distributed in the horizontal and the depth at each point, while UCOAM requires the \((x,y,z)\) coordinates of every node. To ease calculation, all equations (not the variables) are transformed into a uniform curvilinear grid; i.e., from \((x,y,z)\) to \((\xi,\eta,\zeta)\). (This transformation is explained in detail by Hoffmann [30, 59].)

2.2.3 Numerics

To solve the above equations, a finite-difference scheme is used. Second-order accurate central discretization is used for spatial discretization, except for non-linear terms, where a fourth-order accurate scheme based on Kawamura’s method is used [65]. In Kawamura’s method two one-sided schemes are combined together as follows:

\[
(f_\xi \frac{\partial u}{\partial \xi})_{i,j,k} = (f_\xi)_{i,j,k} \frac{-u_{i+2,j,k} + 8(u_{i+1,j,k} - u_{i-1,j,k}) + u_{i-2,j,k}}{12\delta\xi} + |(f_\xi)_{i,j,k}| \frac{u_{i+2,j,k} - 4u_{i+1,j,k} + 6u_{i,j,k} - 4u_{i-1,j,k} + u_{i-2,j,k}}{4\delta\xi},
\]

and \(f_\xi = u_{\xi x} + v_{\xi y} + w_{\xi z}\). Although equation (2.11) appears to be a central finite-difference scheme at first glance, it is actually a forward and a backward scheme combined into one equation. Depending on the sign of \((f_\xi)_{i,j,k}\) either the forward or the backward scheme can be used.

For time discretization, a third-order accurate Runge-Kutta method (RK3) [117, 130] is used. If \(\frac{\partial u}{\partial t} = f(u, \cdots)\), using RK3, this equation is discretized as:

\[
\begin{align*}
u' &= u^n + \frac{\delta t}{3}f(u^n, \cdots) \\
u'' &= u^n + \frac{\delta t}{2}f(u', \cdots) \\
u^{n+1} &= u^n + \delta tf(u'', \cdots)
\end{align*}
\]
2.2.4 Pressure

In equation (2.1), pressure is an a priori known field. It is common in atmospheric and oceanic models to assume that a hydrostatic condition holds for pressure; hence, in the majority of ocean models, the pressure at each point is considered to be the weight of the column of water above it. While the hydrostatic assumption holds well for coarse simulations, the finer the resolution, the more pronounced the deviation of the non-hydrostatic pressure becomes. This deviation from hydrostatic pressure must be addressed at the resolutions, which UCOAM is designed for. One disadvantage of using a non-hydrostatic model is the necessity of solving an elliptic equation at a much higher computational cost.

There are different methods available for calculating pressure [5, 45, 48]. Two were tested in UCOAM, and are documented in [6]. In the first method [57], known as the Marker and Cells (MAC) method, the divergence of the vector momentum equation is calculated. Later, the continuity equation is used to derive an elliptic equation for pressure. At each time step, this elliptic equation is first solved using the value of the variables at time step $n$. Once the pressure is known, using this newly calculated pressure field, all variables are advanced to time step $n + 1$. For this method, the mathematically-derived equation for pressure can be written as:

$$\nabla^2 p = \frac{\nabla \cdot \vec{w}^n}{\delta t} - \nabla \cdot [(\vec{w}^n \cdot \nabla)\vec{w}^n]$$

$$+ \nabla \cdot \left[ \frac{1}{Ro} (v^n \hat{i} - u^n \hat{j}) \right] - \frac{1}{F_r^2},$$

where $Ro$ is the Rossby number and the superscript $n$ means the current time, and $\delta t$ is the non-dimensional time step,

$$\tilde{F}_r = \frac{U^*}{NL^*},$$

(2.14)
and $N$ is the Brunt–Väisälä frequency, defined as:

$$N^2 = -\frac{g \partial \rho}{\rho^* \partial z^*}.$$  (2.15)

Note that the dilation term, i.e. $\nabla^2 (\nabla \cdot u)$, is not present in equation 2.13. This is a result of the continuity equation, which states that this term should be zero. Although physics says that this term should be zero, it is wise to keep it in the pressure equation, as it provides better numerical stability and more accurate results [59].

Implementing this term is difficult, and executing it in curvilinear coordinates is even more challenging. However, including this term results in a more accurate and stable model. (The alternate solution to this approach is the fractional step method [66].)

In the second method, i.e. the fractional step method, [35, 66], all terms in the momentum equation except the pressure term, are evaluated first, and an intermediate value for the three velocity components is calculated. Using these intermediate values and the continuity equation, an elliptic equation for the pressure is derived, as follows:

$$\nabla^2 p = \frac{1}{\delta t} \nabla \cdot u_{\text{intermediate}}.$$  (2.16)

Equation 2.16 is simpler, faster, and more numerically accurate. To compare the difference between the two methods in calculating pressure, a classical seamount problem is chosen. A linearly-changing velocity profile from 0 to 1 is applied on the left side of the domain, and $\nabla u \cdot n = 0$ is chosen as the boundary condition on the right side of the domain. The top lid is set as a free-slip and the bottom is set as a no-slip condition, while the front and back sides are set to be periodic. The fluid is initialized to be at rest.
As the force remains unchanged throughout the simulation, the energy of the system is expected to stabilize at a certain level (both methods behave the same way in this matter). As seen in Figure 2.2, they started at a certain value and quickly stabilized at a certain level. The graph for method two is the same; thus, it is not repeated.

The first measure, which indicates the energy level, shows that both models are working as expected; however, it does not differentiate between the two methods or prove which is superior to the other. Therefore, the ratio of the barotropic to the kinetic energy, which is a non-dimensional variable, is defined as the second measure. It is important that this graph show no oscillations in the ratio, or, if there are oscillations, they damp as time progresses or the oscillation are bounded. Again, both methods show the same general behavior; but the second method, i.e., the fractional method, stabilizes much faster, as shown in Figure 2.3 and Figure 2.4. This means that the second method stabilizes much faster and also requires much less spin-up time.

Figure 2.2: Energy of the System using method I for pressure.
2.3 Test Cases: Neutral Conditions

2.3.1 Grid Generation

A classical seamount under neutral conditions, i.e., homogenous density where the buoyancy term is ignored, is chosen. (Neutral conditions are very common in winter.) The bottom bathymetry is defined by:

\[ D(x, y) = L^* \left(-1 + ae^{-b(x^2+y^2)}\right), \]  

(2.17)
where $L^* = 1000m$ is the length scale and the maximum depth. In equation (2.17), it was assumed that $x$ and $y$ are non-dimensionalized by $L^*$, and $(a, b) = (0.5, 8)$. Furthermore, the $x$- and $y$-coordinates are shifted, such that the zenith of the seamount is at the origin, i.e., $(x, y) \in [-1.8, 1.8] \times [-1.4, 1.4]$. This results in a domain of size $3.6 km \times 2.8 km$, where the depth varies between $1 km$ at its deepest and $0.5 km$ at the origin, i.e. the most shallow region.

To better represent the bottom variations, more grid points are chosen in the middle of the domain. The nodes at the bottom are distributed according to:

$$x_i = D \left\{ 1 + \frac{\sinh[\beta(\xi_i - A)]}{\sinh \beta A} \right\}$$

(2.18)

and

$$A = \frac{1}{2\beta} \ln \left[ \frac{1 + (e^\beta - 1)(D/L_i)}{1 + (e^{-\beta} - 1)(D/L_i)} \right]$$

(2.19)

where $L_i$ is the non-dimensional domain size, i.e. $(L_1, L_2) = (3.6, 2.8)$, $\beta$ is the clustering parameter in the range of $0 < \beta < \infty$ (here $\beta = 5$), and $D$ is the location where clustering is desired (here $D = L_i/0.5$). Two sets of grids were generated: the
first grid used the sigma-coordinate in the vertical $S_1$, and the second grid was a fully curvilinear grid $C_1$, Figure 2.5. The grid properties are summarized in Table 2.1.

2.3.2 Boundary and Initial Conditions

The main flow-stream was assumed to be along the $x$-axis from the left face of the domain to the right face. A linearly increasing non-dimensional velocity from 0 at the bottom to 1 at the top was applied at the left face of the cube and kept constant during the entire simulation. The derivative of the component of the velocity normal to the right face along the vector also normal to the right face is set to zero; therefore, a Neumann boundary condition is used on the right face. The front and back faces of the cube (the sides perpendicular to the $y$-axis) were considered to be periodic. The lower face, or the bottom side, used a no-slip boundary condition and the top side used a free-slip boundary condition. All fields were initialized to zero, i.e. the fluid started at rest, and the Coriolis forces were neglected in these simulations.

2.3.3 Results and Discussion

UCOAM was written entirely in FORTRAN 90. The total simulation time was set at 20,000s, i.e. slightly more than 5 hours and 30 minutes. UCOAM is a LES model, and uses a fully non-hydrostatic approach to solve for pressure; meaning that the majority of the calculation time is spent solving the elliptic equation derived for pressure, i.e. equation (2.16). UCOAM uses an iterative scheme to solve for pressure; hence, the grid’s effect on the number of iterations required to reach the predefined accuracy in
Figure 2.6: Grid affecting the number of iterations required to solve for pressure.

Figure 2.7: Contour lines for velocity along the $x$-axis passing through the zenith of the seamount using $C1$. Slide 100 is at $t = 20000$; distances are in kilometers.

Pressure is important. Figure 2.6 shows the logarithm of the number of iterations required to solve for the pressure. At the beginning of the simulation many iterations are required to reach the desired accuracy in pressure. However, as the simulation advances, the number of iterations decreases. Both grids show the same pattern of increase or decrease in the number of iterations. The bumps in the graph loosely coincides with the times at which the waves reaches the seamount. Generally $C1$
Figure 2.8: Contour lines for velocity along the x-axis passing through the zenith of the seamount using S1. Slide 100 is at $t = 20000$; distances are in kilometers.

Figure 2.9: Changes of vertical velocity share of the total energy.

requires fewer iterations than S1; C1 requires approximately $1.5 \times 10^6$ less iterations.

In Figures 2.7 and 2.8 the contour lines of the velocity on the plane passing through the zenith of the seamount along the x-axis is shown for C1 and S1, respectively. Both codes behave exactly the same on the upstream of the seamount, but C1 shows more eddies on the downstream. Moreover, the wake behind the C1 pushes farther above the seamount’s crest height on the downstream, while in S1,
the seamount mostly affects the field between the bottom and the seamount’s crest height. When time advances, $S_1$ gradually catches up with $C_1$, and the seamount’s influence reaches beyond the crest height.

Interesting results are acquired from an analysis of the vertical velocity’s share
of the total energy as it changes over time. As previously explained, the vertical component of the velocity in geophysical fluids is much smaller than the two horizontal components; hence, the share of energy resulting from the vertical component is very small. \( C1 \) and \( S1 \) both show this low percentage in figure 2.9. However, \( S1 \) generally attributes a higher share to vertical components than \( C1 \). These differences become smaller as time advances, but \( S1 \) continues to produce a stronger vertical velocity. Figure 2.10 shows the spatial distribution of the differences in the share of the vertical velocity of the total energy between \( C1 \) and \( S1 \). Figures 2.11 and 2.12 show the same distribution, but at the plane passing \( y = \pm 150m \). These figures demonstrate that, at the symmetry plane, the \( C1 \) grid generates stronger vertical velocity, but that the \( S1 \) grid generates stronger vertical velocity farther from the symmetry plane. These figures also show that, in coarse resolution studies (e.g., 3.6km × 2.8km), the share of energy due to the vertical velocity is small. However, in very fine resolution studies (e.g., 30m × 30m), due to interaction with the bottom bathymetry, there are regions where the vertical velocity becomes significant, and contains most of the
energy. Hence, solving the full momentum equation for all three components of the velocity is of much importance.

2.4 Another Test Case: Temperature Profile Interacting with The Seamount

Ocean flows are generally considered to be stratified, meaning the density increases and temperature decreases with the depth. Many numerical models also support this behavior. To better understand how bottom bathymetry and grid resolution affects these profiles a test was performed. The numerical domain and the forcing were kept the same as in the previous case; however, in this case the temperature, as an active scalar, was initialized to decrease linearly from close to 20°C on top to 11 degrees Celsius at the bottom. The model ran for the same amount of time as in the previous test case.

At the end of the simulation, both temperature and density were averaged horizontally, to reduce the model resolution to 3.6km × 2.8km, a common resolution in ocean modeling. These two averaged profiles are shown in Figures 2.13 and 2.14. The flow looks completely stratified. Later, the temperature at each grid cell was deducted from the horizontally-averaged temperature at the depth that the cell was located. Note that before the seamount, this difference is close to zero; however, after the seamount, there is significant deviation from the averaged temperature. The stratification is a grid-dependent assumption; although oceans may look like stratified at coarser scales, when high resolution models include interactions with bottom bathymetry, this assumption no longer holds. These findings are supported by past studies and available ocean measurements [17, 37, 49, 76, 110, 123, 138], and by the fact that the mixing in the ocean improves with the presence of even small undulations at the bottom bathymetry.
2.5 Conclusions

The effect of the vertical grid on the vertical component of the velocity has been discussed. As expected, the vertical grid has an important effect on the vertical velocity.
The results are in agreement with those discussed in [52]; further, it is common to drop the forcing terms that are a result of the Earth’s rotation and are multiplied by the vertical velocity, since it is believed that the vertical component of the velocity is minor relative to the other two components. But, as explained by Marshal [81], these terms ”... become uncomfortably large as the equator is approached.” This paper confirms Marshal’s findings, and also suggests that, due to bottom topography and flow field interactions in studies with ultra-high resolution in the horizontal, there would be regions in fine-resolution studies in which the vertical component of the velocity was strong enough; Hence, the forcing terms mentioned above must be addressed.
Chapter 3

Regridding Data: A Package to Interpolate, Extrapolate, and Fit a Curve

3.1 Introduction

The idea of representing a data field via parameterization of a function is not new. In fact, if the functions or basis are sine and cosine, the procedure is known as the Fourier transformation. Many meshless methods make frequent use of radial basis functions (RBF) such as the Gaussian function, and polynomial functions are also used frequently. If the degree of the polynomial is equal to one, then the procedure is called a linear/bilinear/trilinear interpolation in a 1D/2D/3D domain. In geophysical fluid models and many other numerical models, the data is often stored in one location, but its value is needed at multiple locations. For example, in the A-grid approach (see Chapter 2), all variables, including the velocity and pressure, are stored at the cell centers; however, in order to calculate the fluxes, they are also needed at the cell faces. Therefore, some type of interpolation must be performed. Depending on the curvature of the boundary in curvilinear grids, one might need to use extrapolation. Another approach is to fit a surface to the data points and evaluate it at the desired locations. Hereafter, all of these methods will be used interchangeably (although
mathematically they are not the same procedure), or they may simply referred to as "regridding" or "reprojecting."

The choice of interpolating functions can also affect the stability of the numerical scheme in use. As mentioned above, there are numerous choices available; however, these procedures are quite time-consuming (e.g., in fluid models, the same procedure has to be performed millions of times). In some methods, including Krigings, the values of the data must be known before beginning the method. However, in most approaches, the most time-consuming aspect of the method depends on the locations of the points, and not on the value of the data. (The polynomial approach is one such method.) Programs such as SCRIP [2], NCL [1], and ESMF [95], pre-calculate the "interpolation weights." Once the weights are known, the re-projecting procedure would be reduced to a single matrix multiplication. While ESMF and NCL allow for scattered data interpolation, SCRIP only works on structured data. In addition, all of these packages support only interpolation, and not extrapolation or a curve-fitting approach.

Due to their importance, a MATLAB package was developed to calculate these interpolation weights. They can now be interpolated, extrapolated, or fit into a local surface of the data. This package can be used for structured, non-structured, and even scattered grids. The following section describes the equations, how the matrix is formed, and explains important results.

### 3.2 Governing Equations

Let us assume \( f_j \) is a measured or computed parameter, such as pressure or temperature, at locations denoted by \((x_j, y_j)\) for 2D distribution and \((x_j, y_j, z_j)\) for 3D distribution, where \( j \in [1, n] \) and \( n \) is the number of points. These points, with a
known value, are called source points or the source grid. Note that there is no assumption made as to how these points are distributed; therefore, the source grid can be structured, unstructured, or even scattered. In polynomial presentation, the aim is to describe the spatial variation of the parameter with a polynomial functions; This function can be written as follows:

\[ f = f(x, y) = \sum_{p_1=0}^{p_d} \sum_{p_2=0}^{p_d} a_{p_1, p_2} x^{p_1} y^{p_2}, \]  

(3.1)

for 2D grids and in 3D grids as:

\[ f = f(x, y, z) = \sum_{p_1=0}^{p_d} \sum_{p_2=0}^{p_d} \sum_{p_3=0}^{p_d} a_{p_1, p_2, p_3} x^{p_1} y^{p_2} z^{p_3}, \]  

(3.2)

where \( p_d \) is the polynomial degree, and \( a_{p_1, p_2} \) or \( a_{p_1, p_2, p_3} \) are the polynomial coefficients that must be determined. These coefficients are also called free parameters. There are various well-known approaches to determine these coefficients, including Lagrange, Newton, and Barycentric interpolations. Once these coefficients are determined, one can replace \((x^*_i, y^*_i, z^*_i)\) at the point, where there is no measurement or computed value, in order to estimate a value for the parameter, i.e. \( f^*_i \), where \( i \in [1, m] \) and \( m \) is the number of points. These points, whose values are unknown, are called the destination grid.

In 2D there are \( n_s = (p_d + 1)^2 \) free parameters and in 3D there are \( n_s = (p_d + 1)^3 \); by substituting the value and location of the points on the source grid we can form a system of equations to determine the free parameters and uniquely define the polynomial. If exactly \( n_s \) points from the source grid are chosen to form the system of the equation, the resulting polynomial passes exactly through the points on the source grid. Therefore, it would either be interpolation (once the destination point falls within the convex hull of the source grid) or extrapolation (once the destination
point falls outside). If more points than the minimum required are used we are actually fitting a curve to the source grid.

If all points at the source grid are used to determine the polynomial coefficients, the resulting polynomial is considered "global"; however, if only a subset of points on the source grid are used to form the polynomial for each point on the destination grid, then the polynomial is termed "local."

This entire procedure can be thought of as a transform function, which projects/transforms/regrids data from the source grid onto the destination grid. Note that no assumptions on any of the grids are made, so they can be structured, unstructured, or even scattered. The software package introduced here uses the local polynomial approach by inverting the resulting Vandermonde matrices. Vandermonde matrices are known to be poorly conditioned. Another issue regarding high $p_d$ is the Runge phenomenon and the oscillation arising at the boundaries. Both of these issues requires the user to choose $p_d$, carefully.

This entire procedure can be simplified as a sparse matrix, which can then be used to re-project data from the source grid onto the destination grid with a single matrix multiplication, i.e.:

$$ f_{m \times 1}^* = P_{m \times n} f_{n \times 1}, $$

(3.3)

where $f_{n \times 1} = [f_1, f_2, \cdots, f_n]^T$ and $f_{m \times 1}^* = [f_1^*, f_2^*, \cdots, f_m^*]^T$. The following outlines how this matrix was constructed; for the simplicity’s sake, the procedure is described for 2D source grids with $p_d = 1$.

Let us assume $s_i$ points were used on the source grid to project data onto the $i^{th}$ point on the destination grid. Note that $s_i$ could be a different value for each point; however, $s_i \geq n_s$. Let us also assume $S_i = [f_{j_1}, f_{j_2}, \cdots, f_{j_{s_i}}]^T$ are the $s_i$ closest points

37
to the \(i^{th}\) point on the destination grid. We then determine

\[
f^*_i = P_i \begin{pmatrix} f_{j1} \\ f_{j2} \\ \vdots \\ f_{js_i} \end{pmatrix},
\]

where \(P_i\) is a \(1 \times s_i\) matrix computed as follows:

\[
P_i = A^* \left( A^T A \right)^{-1} A^T,
\]

and:

\[
A_{s_i \times n_s} = \begin{pmatrix}
1 & x_{j1} & y_{j1} & x_{j1}y_{j1} \\
1 & x_{j2} & y_{j2} & x_{j2}y_{j2} \\
\vdots & \vdots & \vdots & \vdots \\
1 & x_{js_i} & y_{js_i} & x_{js_i}y_{js_i}
\end{pmatrix},
\]

and:

\[
A^*_{1 \times n_s} = \begin{pmatrix}
1 & x_{i}^* & y_{i}^* & x_{i}^*y_{i}^*
\end{pmatrix}.
\]

\(P_i\) must be calculated for each point on the destination grid. Each element of \(P_i\), i.e. \(<P_i>_1, c\) is then substituted at \(<P>_i,j_c\), or:

\[
<P_i>_{1,1} \rightarrow <P>_{i,j1},
\]

\[
<P_i>_{1,2} \rightarrow <P>_{i,j2},
\]

\[
\vdots
\]

\[
<P_i>_{1,s_i} \rightarrow <P>_{i,js_i},
\]

and all other elements of \(P_{m \times n}\) are set to zero. Note that \(P\) is completely independent of the data and is only a function of the locations. This means that, as long as the
points on both the destination and source grids have not moved relative to each other, \( P \) will not change. Again, this movement is relative, i.e. points on both grids can move, but not relative to each other. Hence, the grids can be translated or even rotated, as long as the same rotation or translation is applied to both grids. Under certain conditions, the grids can also be scaled, while continuing to use the same weights.

In many cases, the fact that \( P \) stays constant can be used to reduce computational need. In our case, i.e., fluid mechanics, the grid cells stay constant throughout the entire simulation; with the only changes being in the values of the velocity, pressure, and other variables. Therefore, we can calculate \( P \) once, then reuse it as many times as needed at a cost of a single, sparse matrix multiplication. Although this procedure was explained for polynomials, the author has successfully applied the same concept to RBF, the Gradient plus Inverse Distance Squared (GIDS) method, and the Inverse Distance Weighted (IDW) method (paper in preparation).

### 3.3 MATLAB Command

The command to create a 2D projector is as follows:

\[
P=\text{ConstructProjector2D}(xs,ys,xd,yd,nPoly,nInterp),
\]

where:

- \( xs \) and \( ys \) are the source grid points in the physical domain,
- \( xd \) and \( yd \) are the destination grid points in the physical domain,
- \( nPoly \) defines the degree of the polynomial to use, and
- \( nInterp \) determines how many points from the source grid must be used to determine the free parameters.
If nInterp is set to \( n_s \), i.e. the minimum required, the resulting matrix, i.e. \( P \) would perform true interpolation or extrapolation. If nInterp is set to a larger number, for each point on the destination grid, the program finds nInterp-closest points on the source grid, fits a surface with a degree given by nPoly and calculates the weights.

The command for 3D grids is the same as for 2D Grids:

\[
\text{P=ConstructProjector3D(xs,ys,zs,xd,yd,zd,nPoly,nInterp),0}
\]

where \( zs \) and \( zd \) are the third dimension coordinates for points on both the source grid and the destination grid. In the geosciences, inverse distance weighted (IDW) is very commonly used; the IDW can be formulated as:

\[
f^* = \frac{\sum \frac{1}{d_i} f_i}{\sum \frac{1}{d_i}} \tag{3.9}
\]

Most of the computation time in IDW is spent locating the nInterp closest points. Using a similar approach, the IDW can also be simplified into a single sparse matrix. Since IDW is very common in the geosciences, a similar function was developed for IDW interpolation for in both 2D and 3D.

Once \( P \) is calculated, we can re-grid data from the source grid onto the destination grid, as follows:

\[
f^*_{mx1} = P_{m \times n} f_{n \times nf}, \tag{3.10}
\]

where \( n_f \) is the number of data fields; i.e., if there are multiple data fields, we can either re-grid them one by one, or each data field can be represented as a column and all re-gridded at once.
3.4 Results

To test the code a structured grid was generated where \((x, y) \in [0, 2\pi] \times [0, 2\pi]\). Four data fields were developed, as follows:

\[
F_1(x, y) = \sin \left( \sqrt{x^2 + y^2} \right),
\]

\[(3.11)\]

\[
F_2(x, y) = \sin (x) \cos (y),
\]

\[(3.12)\]

\[
F_3(x, y) = e^{-\sqrt{x^2+y^2}},
\]

\[(3.13)\]

\[
F_4(x, y) = e^{-\sqrt{(x-x_0)^2+(y-y_0)^2}}.
\]

\[(3.14)\]

Using the developed MATLAB package, a Projector, \(P\), was computed, which interpolates from the nodal grid to the cell centers. \(P\) was computed only once, and then the same matrix was applied to all four functions. The results are shown in Figure 3.1. To test the package for scattered grids, two sets of random data were generated, Figure 3.2. The projector \(P\) was constructed only once and the above mentioned functions were evaluated on source grid points to produce a data field. Using the \(P\) generated above, the data field was projected to the destination grid and its accuracy was computed. The accuracy is shown in Figure 3.3. Note that since both the destination and source grids are randomly generated, each execution of the code will generate a different graph.

The effect of \(n_{\text{Interp}}\) is interesting. By choosing an \(n_{\text{Interp}}\) bigger than \(n_s\), we can actually fitt a local surface on the source grid and evaluate the resulting function on the destination grid to get a value. This means that, unlike the interpolation, the
function representing the surface does not necessarily match the values of the source grid. To check the effect of this approach on the error, a series of tests were performed.
Two scattered grids were randomly generated; the degree of the polynomial was set to 4, i.e. $p_d = 4$; and multiple projector, $P$, was generated using different nInterps ranging from $n_s = 25$ to 45. Figure 3.4 shows how the root mean squared error (RMSE) changes with nInterp. The grid is generated randomly each time the code is executed; and, depending on how the points in the source and destination grids are distributed, different errors will be achieved. However, in all cases, RMSE shows the same behavior relative to nInterp. As shown in the graph, the RMSE is decreased by increasing the nInterp. This behavior was consistent in all the executions that we tried. In Chapter 6 we will show that how the accuracy of the numerical solution to Poisson’s equation changes by changing the nInterp.

Once nInterp is set to the minimum required value, i.e. $n_s$, the system is forced to have zero errors at the source grids, i.e. true interpolation. Therefore, all errors are forced to be distributed only on the destination grid points. However, by setting nInterp to a number larger that of the minimum, we can relax the distribution of
the error; thereby, compensating for some of the errors on the source grid. This explanation has been confirmed by subsequent tests.

![Figure 3.4: Effect of nInterp on interpolation accuracy.](image)

### 3.5 Software Availability

This MATLAB package is available for download at:


We recommend beginning with one of the test functions to get acquainted with the code.
Part II

Investigating CGM Operators
Chapter 4

Accuracy of CGM Operators in 2D and 3D Curvilinear Domains


4.1 Introduction

Poisson’s equation has broad applications in many fields, including fluid mechanic, electrostatics, physics, and image processing. Poisson’s equation also plays an important role in the solution of Navier-Stokes’ (NS) equation. The NS equation is used to express the motion of any fluid, such as those in the atmosphere and oceans. Pressure, particularly the non-hydrostatic pressure, is an a-priori known parameter in NS equations. One common method to solve for pressure is to combine the continuity equation and the NS equation to derive Poisson’s equation, which will describe the non-hydrostatic pressure [4, 45, 48, 74]. Poisson’s equation for nonhydrostatic pressure has no physical meaning and is a purely mathematical means of calculating pressure; however, solving Poisson’s equation as accurately and as efficiently as possible is of the utmost importance, since, its solution plays an important role in
guaranteeing the continuity of both the momentum and mass. After all, Poisson’s
equation for the nonhydrostatic pressure was originally derived by applying the con-
tinuity equation to either the momentum equation or the NS equation.

There are several methods for deriving Poisson’s equation for nonhydrostatic
pressure [74]; however, they all result in the same elliptic equation found in Equation
4.1; with a different function on the right hand side (rhs):

\[
\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} = \text{rhs}(u, v, w). \tag{4.1}
\]

To solve this equation, we must apply one of many available discretization schemes,
(e.g., finite differences (FD) scheme), to obtain a system of linear equations, as follows:

\[
Ap = b. \tag{4.2}
\]

Matrix A has different properties depending on the discretization scheme in use.
Matrix A would normally be very large, in order to be directly inverted; hence,
iterative methods are typically used. Numerical models for NS equations employ
most of their time solving Poissons equation.

In this paper we investigate both the performance (measured by the number
of required iterations) and accuracy of Castillo-Grones mimetic (CGM) difference
operators to solve Poissons equations in fully curvilinear 2D and 3D grids. For the
sake of completeness, in the following section Poissons equation is rewritten in matrix
form, using gradient and divergence operators; its transformation from a curvilinear
physical grid to a regularly spaced computational grid is then reviewed. Later, CGM
difference operators will be introduced, full details of discretization in 2D grids will
be described for both conservative FD and CGM schemes, and the results will be
shown and discussed.

47
4.2 Poisson’s Equation in Fully Curvilinear Grids

Poisson’s equation can be written in vector form using the gradient and divergence operator, as follows:

$$\nabla \cdot (\kappa \nabla \varphi) = f,$$  \hspace{1cm} (4.3)

where:

$$\nabla = \left[ \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right]^T.$$  \hspace{1cm} (4.4)

Poisson’s equation can also be written in matrix form, as follows:

$$\nabla^T \kappa \nabla \varphi = f,$$  \hspace{1cm} (4.5)

where $\kappa$ is the identity matrix and $\nabla$ is a $3 \times 1$ matrix with the entries given above.

To solve the above equation, we must transform the equation from a curvilinear physical domain to a regularly spaced computational space. A summary of this procedure is provided, however, a full description can be found in many manuscripts [30, 59, 69]. Assuming that the $(x,y,z)$ domain is transformed to $(\xi,\eta,\zeta)$ space, the operators in these two domains will be related to each other as follows:

$$\tilde{\nabla} = T \nabla,$$  \hspace{1cm} (4.6)

where:

$$\tilde{\nabla} = \left[ \frac{\partial}{\partial \xi}, \frac{\partial}{\partial \eta}, \frac{\partial}{\partial \zeta} \right]^T,$$  \hspace{1cm} (4.7)

and:

$$T = \begin{bmatrix} x_\xi & y_\xi & z_\xi \\ x_\eta & y_\eta & z_\eta \\ x_\zeta & y_\zeta & z_\zeta \end{bmatrix}.$$  \hspace{1cm} (4.8)
$x_\xi$ is the shorthand for $\partial x/\partial \xi$. Inverting the above equation, we get:

$$\nabla = T^{-1} \tilde{\nabla} = \frac{1}{J} C \tilde{\nabla}, \quad (4.9)$$

where $J$ is the Jacobian of the transformation matrix and $C$ is the cofactor of the transformation matrix. By substituting this in Equation 4.5, we can then rewrite Poisson’s equation in the computational space as:

$$\tilde{\nabla} T \tilde{\kappa} \tilde{\nabla} \phi = \tilde{f}, \quad (4.10)$$

where:

$$\tilde{\kappa} = \frac{1}{J} C^T k C, \quad (4.11)$$

and

$$\tilde{f} = J f. \quad (4.12)$$

Note that $\tilde{\kappa}$ is no longer the identity matrix and the off-diagonal entries are not necessarily zero any longer, meaning that Poisson’s equation will now be transformed into a fully elliptic equation in the computational domain with all cross-derivatives present. This adds additional complexity to the solution of Poisson’s equation. For example, in a 2D curvilinear grid, $\tilde{\kappa}$ can be written as follows:

$$J \tilde{\kappa} = \begin{bmatrix} (y_\eta^2 + x_\eta^2) & -(y_\xi y_\eta + x_\xi x_\eta) \\ -(y_\xi y_\eta + x_\xi x_\eta) & (y_\xi^2 + x_\xi^2) \end{bmatrix}. \quad (4.13)$$

As a result, Poisson’s equation is transformed into a diffusion problem, in which the diffusion coefficient is neither homogeneous nor isotropic. It is easily shown that when the grid is orthogonal (or near orthogonal) the off-diagonal elements are zero.
4.3 Castillo-Grone’s Mimetic (CGM) Difference Operators

The Castillo-Grone mimetic approach provides the exact equivalent of the gradient and the divergence operators in a discrete domain [24]. It has been shown that these discrete operators satisfy all properties of their continuous version in the discrete sense; are conservative; and provide the same accuracy within the entire domain, including the boundaries without the need for "dummy" or "ghost" nodes [4, 16, 28]. The end results of these operators are given here. For further information on how they are generated, we refer to the provided references. The Castillo–Grone mimetic approach uses a staggered grid to calculate the gradient and divergence (see Figures 4.1 and 4.3).

The second-order accurate CGM gradient operator is calculated as follows:

\[
\begin{align*}
\hbar G_0 &= \begin{bmatrix} -\frac{8}{3} & 3 & -\frac{1}{3} \end{bmatrix} \begin{bmatrix} u_0 & u_1 & u_1 \end{bmatrix}^T \\
\hbar G_{1 \leq i \leq n-1} &= \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} u_{i-\frac{1}{2}} & u_{i+\frac{1}{2}} \end{bmatrix}^T \\
\hbar G_n &= \begin{bmatrix} \frac{1}{3} & -3 & \frac{8}{3} \end{bmatrix} \begin{bmatrix} u_{n-1\frac{1}{2}} & u_{n-\frac{1}{2}} & u_n \end{bmatrix}^T 
\end{align*}
\]

(4.14)

where \( \hbar \) is the grid spacing. Likewise, the second-order accurate divergence operator is given as follows:
\[ 1 \leq i \leq n - 1; \quad hD_{i-\frac{1}{2}} = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} u_{i-1} & u_i \end{bmatrix}^T \tag{4.15} \]

The CGM gradient and divergence coincide with the cell-centered finite difference in the interior of the mesh; however, this subtle difference at the boundary has been shown to improve the accuracy of the general solution, particularly in solving Poisson’s equation, where the solution is dominated by the boundary condition.

There are different fourth-order accurate CGM gradient and divergence operators available (Castillo and Grone 2003), which are known as three-parameter family operators, i.e. \((\alpha, \beta, \gamma)\). By setting \((\alpha, \beta, \gamma) = (1/24, 0, -1/24)\), we get the following forth-order accurate CGM gradient operator:

\[
\begin{align*}
  hG_0 &= \begin{bmatrix} -1152 & 10063 & 2483 & -3309 & 2099 & 697 \\ 407 & 3256 & 9768 & 3256 & 3256 & 4884 \end{bmatrix} \\
  hG_1 &= \begin{bmatrix} -11 & 17 & 3 & 5 & 1 \\ 12 & 24 & 8 & 24 & 24 \end{bmatrix} \\
  hG_m &= \begin{bmatrix} 1 & 27 & 27 & 1 \\ 24 & 24 & 24 & 24 \end{bmatrix} \\
  hG_{n-1} &= \begin{bmatrix} -1 & 5 & -3 & -17 & 11 \\ 24 & 24 & 8 & 24 & 12 \end{bmatrix} \\
  hG_n &= \begin{bmatrix} 697 & -2099 & 3309 & -2483 & -10063 & 1152 \\ 4884 & 3256 & 3256 & 9768 & 3256 & 407 \end{bmatrix}
\end{align*} \tag{4.16} \]

Likewise, by setting \((\alpha, \beta, \gamma) = (0, 1/24, -1/24)\), the fourth-order accurate CGM divergence operator becomes:

\[
\begin{align*}
  hD_0 &= \begin{bmatrix} -4751 & 909 & 6091 & -1165 & 129 & 25 \\ 5192 & 1298 & 15576 & 5192 & 2596 & 15576 \end{bmatrix} \\
  hD_m &= \begin{bmatrix} 1 & 27 & 27 & 1 \\ 24 & 24 & 24 & 24 \end{bmatrix} \\
  hD_n &= \begin{bmatrix} 25 & -129 & 1165 & -6091 & 909 & 4751 \\ 15576 & 2596 & 5192 & 15576 & 1298 & 5192 \end{bmatrix}
\end{align*} \tag{4.17} \]
### 4.4 Discretization

The discretization is explained in complete detail in a fully curvilinear 2D grid. The same method is used to discretize in 3D.

#### 4.4.1 Using Cell Centered Finite Difference

In 2D, the transformed Poisson’s equation can be written as follows:

\[
\frac{\partial}{\partial \xi} \left( \tilde{\kappa}^{1,1} \frac{\partial \varphi}{\partial \xi} \right) + \frac{\partial}{\partial \eta} \left( \tilde{\kappa}^{2,1} \frac{\partial \varphi}{\partial \eta} \right) + \frac{\partial}{\partial \xi} \left( \tilde{\kappa}^{1,2} \frac{\partial \varphi}{\partial \xi} \right) + \frac{\partial}{\partial \eta} \left( \tilde{\kappa}^{2,2} \frac{\partial \varphi}{\partial \eta} \right) = \tilde{f}. \tag{4.18}
\]

Each term in the above equation is discretized as follows:

\[
\frac{\partial}{\partial \xi} \left( \tilde{\kappa}^{1,1} \frac{\partial \varphi}{\partial \xi} \right) \bigg|_{i,j} = \left[ \tilde{\kappa}^{1,1}_{i-0.5,j} \varphi_{i-1,j} - \left( \tilde{\kappa}^{1,1}_{i-0.5,j} + \tilde{\kappa}^{1,1}_{i+0.5,j} \right) \varphi_{i,j} + \tilde{\kappa}^{1,1}_{i+0.5,j} \varphi_{i+1,j} \right], \tag{4.19}
\]

\[
\frac{\partial}{\partial \eta} \left( \tilde{\kappa}^{2,2} \frac{\partial \varphi}{\partial \eta} \right) \bigg|_{i,j} = \left[ \tilde{\kappa}^{2,2}_{i,j-0.5} \varphi_{i,j-1} - \left( \tilde{\kappa}^{2,2}_{i,j-0.5} + \tilde{\kappa}^{2,2}_{i,j+0.5} \right) \varphi_{i,j} + \tilde{\kappa}^{2,2}_{i,j+0.5} \varphi_{i,j+1} \right], \tag{4.20}
\]

\[
\frac{\partial}{\partial \xi} \left( \tilde{\kappa}^{1,2} \frac{\partial \varphi}{\partial \eta} \right) \bigg|_{i,j} = \left[ \tilde{\kappa}^{1,2}_{i-0.5,j} \varphi_{i-1,j-1} - \tilde{\kappa}^{1,2}_{i+0.5,j} \varphi_{i+1,j} \\
+ \left( \tilde{\kappa}^{1,2}_{i-0.5,j+0.5} - \tilde{\kappa}^{1,2}_{i,j} \right) \varphi_{i,j-1} + \left( \tilde{\kappa}^{1,2}_{i-0.5,j} + \tilde{\kappa}^{1,2}_{i+0.5,j} \right) \varphi_{i,j+1} \\
- \tilde{\kappa}^{1,2}_{i-0.5,j} \varphi_{i-1,j+1} + \tilde{\kappa}^{1,2}_{i+0.5,j} \varphi_{i+1,j+1} \right], \tag{4.21}
\]

and:

\[
\frac{\partial}{\partial \eta} \left( \tilde{\kappa}^{1,2} \frac{\partial \varphi}{\partial \eta} \right) \bigg|_{i,j} = \left[ \tilde{\kappa}^{1,2}_{i,j-0.5} \varphi_{i-1,j-1} - \tilde{\kappa}^{1,2}_{i,j-0.5} \varphi_{i+1,j} \\
\left( \tilde{\kappa}^{1,2}_{i,j-0.5} - \tilde{\kappa}^{1,2}_{i,j+0.5} \right) \varphi_{i-1,j} + \left( \tilde{\kappa}^{1,2}_{i,j-0.5} + \tilde{\kappa}^{1,2}_{i,j+0.5} \right) \varphi_{i+1,j} \\
- \tilde{\kappa}^{1,2}_{i,j+0.5} \varphi_{i-1,j+1} + \tilde{\kappa}^{1,2}_{i,j+0.5} \varphi_{i+1,j+1} \right]. \tag{4.22}
\]
Refer to Figure 4.3 for the index numbers.

Note that:

- $\tilde{\kappa}^{1,1}$ needs to be calculated only in the middle of edges perpendicular to $\xi$.
- $\tilde{\kappa}^{2,2}$ needs to be calculated only in the middle of the edges perpendicular to $\eta$.
- However, $\tilde{\kappa}^{1,2}$ needs to be calculated on both locations.

4.4.2 Using the CGM Difference Operator

Assuming that data is stored at the cell center, the CGM gradient and divergence operators can be used to discretize the Poisson’s equation in a curvilinear domain as follows:

1. Apply the G operator to calculate $\frac{\partial \varphi}{\partial \xi} \bigg|_{i+0.5,j}$.

2. Apply the G operator to calculate $\frac{\partial \varphi}{\partial \eta} \bigg|_{i,j+0.5}$.

3. Interpolate $\varphi_{i,j}$ to $\varphi_{i+0.5,j+0.5}$.

4. Apply the G operator on $\varphi_{i,j+0/5}$ to calculate $\frac{\partial \varphi}{\partial \xi} \bigg|_{i,j+0.5}$.

5. Apply the G operator on $\varphi_{i+0.5,j}$ to calculate $\frac{\partial \varphi}{\partial \eta} \bigg|_{i+0.5,j}$.

6. Multiply by $\tilde{\kappa}$.
7. Apply the D operator to obtain the final discretized left hand side of the Poisson’s equation

The vector operations available in MATLAB or FORTRAN can be used to write an efficient code for the above procedure.

4.4.3 A 2D Example

In 2D curvilinear cases, the right hand side function is chosen to be:

\[ f = \frac{\cos \sqrt{x^2 + y^2}}{\sqrt{x^2 + y^2}} - \sin \sqrt{x^2 + y^2} \]

The exact solution to the above equation, using Dirichlet boundary conditions is \( \sin \sqrt{x^2 + y^2} \). However, the curvilinear grid is generated based on a polar coordinate system; the transformation metrics are all calculated numerically without considering their polar relations. Four different grid resolutions were chosen, with the average grid spacing changing from 0.3265 on the first grid (G1) to 0.0789 on the last grid (G4). The number of nodes in each direction were chosen in such a way that the grid cells resembled a square as closely as possible. An iterative scheme was then used to solve the resulting system of linear equations. In all methods, the maximum number of iterations was set at 20000. The results of the different schemes and the numbers of required iterations is given in Table 4.1. The central difference scheme (CD) reached the maximum limit of iterations in all cases where the Jacobi method was used. However, when the Gauss-Seidel method was used, the scheme converged to the solution in all cases. This behavior was already expected, as the Gauss-Seidel method requires fewer iterations to converge. The mimetic discretization converged to the solution in all cases where the Jacobi method was used (again, the CD scheme did not).
Table 4.1: Comparison between number of required iterations and the existing code.

<table>
<thead>
<tr>
<th>Grid Code</th>
<th>CCDS Jacobi</th>
<th>CCDS Gauss-Seidel</th>
<th>Mimetic Jacobi</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>20000 (C2)</td>
<td>635 (C1)</td>
<td>1148 (C3)</td>
</tr>
<tr>
<td>G2</td>
<td>20000 (C2)</td>
<td>2583 (C1)</td>
<td>4539 (C3)</td>
</tr>
<tr>
<td>G3</td>
<td>20000 (C2)</td>
<td>5745 (C1)</td>
<td>10044 (C3)</td>
</tr>
<tr>
<td>G4</td>
<td>20000 (C2)</td>
<td>10152 (C1)</td>
<td>17044 (C3)</td>
</tr>
</tbody>
</table>

Convergence Code:
- C1: Converged
- C2: Max iteration number reached
- C3: No improvements in the residual

Figure 4.4: Error comparison in 2D curvilinear grid.

The mean square error for the different schemes is shown in Figure 4.4. As can be seen for all cases, the CD scheme has a higher rate of error than the mimetic method. Note that the graph is in the log-log scale. Moreover, the Jacobi and Gauss-Seidel methods used for CD scheme have a similar error, thereby proving that the 20,000 limit set on the number of iterations was correct.

To better understand how the mimetic scheme is behaving, the code was run again, but this time the maximum number of iterations in the Jacobi method for the mimetic scheme was limited to the same number of iterations required by the Gauss-Seidel method on the CD scheme. This is not a fair comparison, as the Jacobi
method used on the mimetic scheme is well known to converge at a much slower rate than the Gauss-Seidel method used on the CD scheme. However, Figure 4.4 illustrates that even in these types of cases, the mimetic scheme provides significantly better results. Comparing the error of the mimetic scheme when it is under-iterated and when it is fully iterated, shows that when the mimetic method is used, the error drops quickly. In fact, the Jacobi method is doing a large number of iterations just to improve the general error by a small fraction; hence, much looser stopping criteria is required for the mimetic scheme, although good results can still be obtained. This lower number of required iterations is an important factor, particularly in cases such as fluid animations, where accurate pressure is not necessary.

4.4.4 A 3D Example

The CGM operators were also tested in a three-dimensional curvilinear grid, Figure 4.5. The physical domain is part of a thin shell with a radius of $2\pi$ and the thickness of $\pi$. The right hand side of the Poisson’s equation was changed to $f = a \cdot b \cdot e^{2-\frac{R}{\pi}}$, where $a = (e^\lambda - 1)^{-1}$, $b = \pi^{-2} - 2\pi^{-1}$, $\lambda = -1$, and $R = \sqrt{x^2 + y^2 + z^2}$. Using a Dirichlet boundary condition the exact solution is $(e^\lambda - 1)^{-1}e^{-\lambda\frac{R}{\pi}}$.

Only the Jacobi iteration was used in this 3D example. The maximum number of iterations was again limited to 20,000. The CGM difference operators behaved in the same way in the 3D example as they did in the 2D example; i.e., in the case of coarser grids, the CGM operators had better accuracy (25 percent higher), with a significantly smaller number of iterations. As in the 2D case, by refining the grid, the performance of the CGM operators became closer to that of the CD operators, Figure 4.6.
Figure 4.5: 3D Curvilinear Grid.

Figure 4.6: Error comparison in 3D grid.
4.5 Conclusions and Recommendations

The performance of CGM operators when solving Poissons equation in both 2D and 3D fully curvilinear grids was investigated. It was proven that CGM operators require a significantly smaller number of iterations, but still provide a highly accurate solution. The difference was even higher in coarse resolutions, which is highly important, since, in real problems (e.g., atmospheric and oceanic simulations), the grid resolution is always too coarse to capture all the scales; hence, having a method which achieves better accuracy in a coarse resolution with a lower number of iterations is of utmost importance.
Chapter 5

Stability Analysis


5.1 Introduction

Both hydrostatic and nonhydrostatic ocean and atmosphere models must deal with different scales of motion, as well as forces with a wide range of frequencies. To capture features with different frequencies, a varying time step is needed. In order to address the wide range of frequencies with a single time step, a time-splitting method is often used [78]. There are various explicit time-splitting schemes available: in 1978 Klemp and Wilhelmson combined a leapfrog scheme with a forward-backward scheme [89] to derive one of the most commonly used schemes [68]. The Crowley scheme is another splitting method for time integration [127], and Wicker and Skamarock have introduced several versions of their time-splitting scheme, based on the two-step Runge-Kutta method (RK2) [131] and the third-order Runge-Kutta method (RK3), which has already proven its superiority in time integration [130]. The accuracy,
efficiency, and ease of implementation of the RK3 scheme was the motivation for our use of the same time-splitting scheme in the Unified Curvilinear Ocean Atmosphere Model (UCOAM) [4]. Note that the RK3 scheme is currently being used in the Weather Research & Forecasting (WRF) model, which is the most commonly used model in the field [67, 94].

However, time integration is only one aspect that must be addressed: Other important factors affecting stability in elastic models and advection equations are (1) spatial discretization [11, 45, 60] and (2) interpolation schemes [130]. The combination of spatial discretization and time integration is particularly important. For example, a central spatial scheme is unconditionally unstable when used with the RK2 or Euler methods [60]. It is also commonly known that, in general, increasing the order of accuracy will limit the stability regions; hence, to avoid this, a smaller time step must be selected [60].

Interpolation has a similar effect; while higher order interpolation schemes are more accurate, they also limit the stability region. For example, Wicker and Skamarock have shown that the stable Courant number decreases from 1.61 to 1.08 when the order of accuracy of the interpolator function is increased from three to six when their RK3 scheme is used in conjunction with a second-order spatial discretization scheme [130].

This chapter investigates the stability of Castillo-Grone Mimetic (CGM) divergence operators with higher orders of accuracy as well as RK3 schemes developed by Wicker and Skamarock. Mimetic schemes are a class of numerical schemes that satisfy the physical properties of their continuous operator in a discrete environment [16, 24, 26, 28, 31, 58, 63]. Here, the fourth-order CGM divergence and gradient operator, along with first-, fourth-, and sixth-order accurate interpolation methods are tested and their performance reported. The reason for considering first-order
accurate linear interpolation comes from the authors’ interest in using NVIDIA’s Graphics Processing Units (GPUs), which have special circuitry for one-, two-, and three-dimensional linear/bilinear/trilinear interpolation. This allows the interpolation to be done very quickly and efficiently at the hardware level; although the same circuitry can be harnessed for higher-order interpolation schemes the help of the appropriate software.

The next section covers governing equations and various discretization schemes. Later, the results of each method will be presented and the region of stability discussed based on the stable Courant number that has been numerically obtained. Additionally, using the Von Neumann stability analysis method, the stability of the CGM fourth-order accurate divergence operator with the RK3 and linear interpolation will be also discussed.

5.2 Numerical Approach

5.2.1 Governing Equation

In the absence of any source or sink term, the advection equation can be formulated in non-conservative form using the gradient operator, as follows:

$$\frac{\partial q}{\partial t} + u \cdot \nabla (q) = 0,$$

(5.1)

where $q$ is a scalar quantity and $u = Const.$ is the advection velocity. Using the continuity equation, i.e. $\nabla \cdot u = 0$, Equation 5.1 can be written in conservative form using the divergence operator, as follows:

$$\frac{\partial q}{\partial t} + \nabla \cdot (u q) = 0.$$

(5.2)

Both forms of the equation are investigated here; the CGM gradient operator is
used with Equation 5.1 and the CGM divergence operator is used with Equation 5.2.

5.2.2 Initial and Boundary Conditions

$q$ is initialized with

$$q(x, t = 0) = \frac{1}{1 + e^{80(z - 0.15)}},$$

where $z = |x - 0.5|$ and $x \in [0, 1]$. A periodic boundary condition is selected, $dx$ and $dt$ are kept constant at 0.02, and $u$ is chosen based on the given Courant number, i.e. 

$$u = C_r \frac{dx}{dt^{-1}}.$$

5.2.3 Spatial Discretization

KWM Scheme

Due to their nature, it is widely believed that a forward or backward step works better for advection terms. Kawamura et al. [65] combined a forward and a backward scheme into a single equation in curvilinear coordinates; this scheme, abbreviated here as KWM, is fourth-order accurate and does not require interpolation. Adapting the KWM scheme for regularly-spaced grids, is written as:

$$(\frac{\partial q}{u \partial x})_i = u_i \frac{-q_{i+2} + 8(q_{i+1} - q_{i-1}) + q_{i-2}}{12dx} + |u_i| \frac{q_{i+2} - 3q_{i+1} + 3q_i - q_{i-1} + q_{i-2}}{4dx}. \quad (5.4)$$

Castillo-Grone Gradient and Divergence Operators

Both second- and fourth-order CGM gradient and divergence operators are used in this paper. For more information on how these operators are constructed, see [24, 26, 28, 31, 63]. Fourth-order accurate CGM gradient and divergence operators are each
part of a three-parameter family of operators. Here, we use only one member of each family, corresponding to the following parameters:

\[
(\alpha, \beta, \gamma) = (0, \frac{1}{24}, -\frac{1}{24})
\]  

(5.5)

5.2.4 RK3 Scheme

A three-step time-splitting scheme, first introduced by Wicker and Skamarock [130], is used to integrate the equations provided in the form:

\[
\frac{\partial u}{\partial t} = f(u, \cdots).
\]  

(5.6)

The solution is advanced from time step \(n\) to time step \(n+1\), as follows:

\[
u^* = u^n - \frac{dt}{3} f(u^n, \cdots)
\]

(5.7)

\[
u^{**} = u^n - \frac{dt}{2} f(u^*, \cdots)
\]

\[
u^{n+1} = u^n - \frac{dt}{1} f(u^{**}, \cdots)
\]

therefore, the right-hand side is evaluated three times.

5.3 Von Neumann Stability Analysis

Von Neumann stability analysis is the most commonly used method, and was developed at Los Alamos National Laboratory in 1944 [109] to study the behavior and stability of a numerical scheme. The method was first distributed amongst a small group of scientists, and was later published publicly [32, 36, 42, 105]. In this method, each Fourier component of the solution is replaced by

\[
q_i^n = V^n e^{ikx(i\Delta x)} = V^n e^{i\theta},
\]  

(5.8)
where \((I, \theta, k_x) = (\sqrt{-1}, k_x \Delta x, 2\pi/\lambda)\) with \(\lambda\) being the wavelength and \(V^n\) is the amplitude function at time step \(n\). Defining \(G = V^{n+1}/V^n\) as the amplification factor, one can discover how the error and solution vary with time. For the solution to remain bounded, one must have \(|G| \leq 1\) for all \(\theta\). This provides the stability criteria for the numerical scheme in use for a given application. If \(|G| < 1\) the scheme would be too damping; hence, must keep \(|G|\) as close to 1 as possible. \(G\) is a complex number, which means that \(G\) should appear as a unit circle once it is shown in a polar system.

Using the fourth-order accurate CGM divergence operator with linear interpolation and the RK3 scheme for time discretization (CGMD4I1RK3) in Equation 5.2, the amplification factor can then be calculated as follows:

\[
G = -\frac{c^3}{6}e^{(-6I)t} + \frac{c^3}{6}e^{(6I)t} + 13c^3e^{(-5I)t} - 13c^3e^{(5I)t} + (0.5c^2 - 338c^3)e^{(-4I)t} + (0.5c^2 + 338c^3)e^{(4I)t} + (-26c^2 + \frac{8749}{3}c^3)e^{(-3I)t} + (-26c^2 - \frac{8749}{3}c^3)e^{(3I)t} + (-c + 338c^2 + 676.5c^3)e^{(-2I)t} + (c + 338c^2 - 676.5c^3)e^{(2I)t} + (26c + 26c^2 - 8814c^3)e^{(-I)t} + (-26c + 26c^2 + 8814c^3)e^{(I)t} + (1 - 677c^2),
\]

(5.10)

where \(c = u \cdot dt/(48dx) = C_r/48\) and \(I = \sqrt{-1}\). Figure 5.1(a) shows the stable region of the amplification factor calculated above. Figure 5.1(b) shows the amplification factor in a polar system. If the solution contains only high or low wave numbers, the stable Courant number using CGMD4I1RK3 can become exceedingly high (even higher than 2). However, in cases where all wave numbers are present, such as in
atmospheric and oceanic modeling, the maximum stable Courant number is approximately 1.66. This also agrees with the experimental approach which is presented in the following section in Figure 5.7. Figure 5.1 also shows that, by increasing the Courant number for a given wavenumber, the amplification factor first decreases and then increases. Looking more carefully on the figure, it becomes evident that when \( \theta = 0 \) there are few cases that the amplification factor has exceeded one (for example, see the small white patches on the left side of the figure around \( C_r = 0.98 \)). Although, it is reported that the stable Courant number is 1.66, it is suggested to keep the Courant number lower than 0.98. This behavior explains why in previous studies [111], it was determined that when the time step is increased, i.e. the Courant number is increased, lower errors are obtained. Although the scheme is stable for Courant numbers up to 1.66 (with few exceptions); when the Courant number is higher than one there is damping in certain wavelengths; as a result, again it is suggested to keep the Courant number lower than one. However, since the solution is not known, it is normally suggested to keep the Courant number lower than one as a safeguard for those cases that the Courant number temporarily can exceed its maximum permitted, Figure 5.1. Temporary increasing of Courant number introduces some damping but will not make the model unstable and does not crashes the code. Being able to keep Courant number as high as possible is important. This means that the scheme allows for larger time steps and yet be stable.

Equation 5.9 makes it clear that the amplification factor for the CGM operators combined with the RK3 scheme is not a simple function. This is often the case once multiple-steps time integration schemes are used. Therefore, in the next section, an experimental approach is used to determine a stable Courant number.
Figure 5.1: Amplification factor using the fourth-order accurate CGM divergence operator with linear interpolation and the RK3 as the time discretization scheme.

5.4 Results and Discussions

MATLAB was used to solve Equation 5.1 and 5.2 numerically using the discretization schemes shown in the previous sections. Periodic boundary conditions were applied in all cases, and the number of iterations was chosen so that the initial signal would pass through the domain twice before reaching its original position. In theory, since the velocity is kept constant throughout both space and time, in a perfect case, one should get exactly the same signal, each time; however, in practice, due to numerical errors that occur during actual practice, this proves impossible. To measure the error,
we used the root mean square error (RMSE).

The KWM scheme produced smooth and relatively accurate results; however, it was stable only up to a Courant number of 0.6. Once $Cr > 0.63$ was chosen, the scheme became unstable. The RMSE for this method stays relatively constant and does not change with an increase of the Courant number; hence, unlike other methods, there is no sign that the scheme is becoming unstable. This is not a desirable behavior as there is no warning that the method is getting unstable. Figure 5.2 shows the solution using the Kawamura scheme.

The second-order accurate CGM divergence operator also provided smooth results. When a fourth-order interpolation scheme was used, the scheme was stable up to a Courant number of 1.3; when a sixth-order interpolation scheme was used, the scheme was stable up to a Courant number 1.1. Yet, the sixth-order interpolation scheme resulted in increased accuracy only at lower Courant numbers, and once the Courant number reached 0.7, both methods behaved approximately the same. In Figure 5.3 the numerical solution is compared with the analytic solution using the second-order CGM divergence operator with fourth- and sixth-order interpolation schemes, respectively.

Before discussing the results obtained using the CGM gradient operator, it should be noted that the CGM gradient operator was developed for use with staggered
Figure 5.3: Sample solution using second-order accurate CGM divergence operator with fourth-order interpolation scheme, $Cr = 1.2$ (left), and sixth-order interpolation scheme, $Cr = 1.1$ (right).

meshes. As a result, the CGM gradient operator requires the data to be in the center of the cells, as well as on the first and last node on the boundary. However, the gradient itself is calculated at the nodes, refer to Figure 5.4. Recall that, in this paper, the variable $q$ is stored only at the cell centers and not on the nodes. Hence, to solve Equation 5.1 using the CGM gradient, we must first calculate the gradient at the cell’s centers where the data exists. This leaves us with a different implementation.

In the first approach, i.e. $V1$, the data is first interpolated from the cell centers to the boundary points using either a linear or fourth-order interpolation scheme, which requires the usage of a periodic boundary condition. Once the gradient is calculated at the nodes using the CGM gradient operator, it is linearly interpolated to the cell centers, refer to Figure 5.5. Hence, in the V1 approach, the calculated gradient is interpolated to the cell centers.

In the second approach, i.e. $V2$, we assume that the domain begins at the center of the first cell and ends at the center of the last cell. The data is then interpolated from the cell centers to all the interior nodes. As a result, once the CGM gradient operator is applied, the gradient is then calculated at the cell’s centers of the original grid and there is no need to interpolate the calculated gradient, refer to Figure 5.5. It should be noted that in this case, only higher-order interpolation is possible; otherwise, the
Figure 5.4: CGM gradient, where $f$ denotes the locations that the data must be held, and $Gf$ shows the location where the gradient is calculated using the CGM gradient operator.

boundary conditions will not enter the solution.

The third approach, i.e. $V3$, is similar to $V1$, except that once the gradient is calculated at the nodes, a fourth-order interpolator is used to obtain the gradient at the cell centers. All three methods produce a relatively smooth solution; however, $V1$’s solutions shown the most error; although it was stable up to a Courant number of 1.8 for both the linear and fourth-order interpolations. $V2$ and $V3$ were both stable up to a Courant number of 1.3, and were also more accurate than $V1$. However, $V2$ achieved less accuracy at lower Courant numbers relative to $V3$. See Figure 5.6 for a sample output of the different implementations. By increasing the order of accuracy of the operators, the stable Courant number decreases; however, for the fourth-order accurate CGM operators, the lowest stable Courant number was 1.1. See Figure 5.7 for a comparison of the RMSE and the stable Courant number for all of the methods.

5.5 Using Other Time Schemes

The same test was performed using other time discretization schemes. The results are summarized in Table 5.1. Also, the same test was performed on solving acoustic equations, which can be written as follows:
Figure 5.5: V1 Approach (left), data is located at the cell centers only. Using the boundary condition, data is interpolated at the boundary nodes to satisfy the grid requirements of the CGM gradient operator. The gradient is then calculated at the nodes and linearly interpolated to the cell centers. V2 Approach (right), data is interpolated to all the interior nodes. The CGM gradient operator is used to calculate the gradient at the cell centers directly. Hence, there is no need to interpolate the calculated gradient.

Figure 5.6: Second-order accurate CGM gradient operator with fourth-order interpolation scheme using the V1 approach, $Cr = 1.65$ (left); using the V2 approach, $Cr = 1.2$ (middle); using the V3 approach, $Cr = 1.2$ (right).
Figure 5.7: All methods compared together.
\[
\frac{\partial u}{\partial t} = -\frac{1}{\rho} \Delta (p) \\
\frac{\partial p}{\partial t} = -K \Delta \cdot u
\]  

(5.11)

The results of stability analysis is shown in Table 5.2

### Table 5.1: Stability using other time discretization schemes in solving conservative and non-conservative advective equation.

<table>
<thead>
<tr>
<th>Time Discretization</th>
<th>2nd order</th>
<th>4th order</th>
<th>6th order</th>
</tr>
</thead>
<tbody>
<tr>
<td>RK3</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SSPRK(3,3)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SSPRK(5,4)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

### Table 5.2: Stability in solving acoustic equations.

<table>
<thead>
<tr>
<th>Time Discretization</th>
<th>2nd order</th>
<th>4th order</th>
<th>6th order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Staggered 2nd order</td>
<td>yes</td>
<td>yes</td>
<td>No</td>
</tr>
<tr>
<td>RK3</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>RK4</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SSPRK(3,3)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>SSPRK(5,4)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>RKS3</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>RKS4</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

### 5.6 Conclusion

Different CGM operators with varying orders of accuracy in conjunction with RK3 time discretization were used to solve both the conservative and non-conservative forms of the advection equation. The effect of various interpolation schemes was studied. It was seen that the proposed scheme resulted in a stable scheme for Courant
number higher than one in all cases, with few exceptions. It was also shown that if the solution consists of either low or high wavenumber one can use even higher Courant number. However, if all scales of motion exist in the solution, which is common in atmospheric and oceanic modeling, the maximum stable Courant number was found to be 1.8, experimentally. This can lead to a better performance since it gives more flexibility in choosing the time step by decreasing the spatial discretization size, i.e. $dx$. Despite being stable for Courant number more than one, it is recommended to limit the Courant number to one; since, the method shows damping once Courant number goes beyond one.

In Chapter 8 this study is extended to 2D domains.
6.1 Introduction

Recall that Poisson’s equation is an elliptic, partial differential equation with wide applications in many fields, including electrostatics, magnetism, mechanical engineering, fluid dynamics, ground water, and dispersion of pollutants, [4, 45, 48]. Poisson’s equation is written as follows:

$$\nabla \cdot \nabla f = F,$$

where (in this paper) \( f \) is an unknown 2D function, i.e. \( f(x, y) \) that would be determined after solving the above equation, and \( F(x, y) \) is a 2D function, called a right-hand side (RHS) function, which is either known analytically or a discretized estimation of it is known. \( L = \nabla \cdot \nabla \) is called the Laplace operator, not to be mistaken with the Laplace equation;

Due to its importance, many studies have been devoted to Poisson’s equation, and
a multitude of solvers are available online. However, most of these studies and software focus on Poisson’s equation in Cartesian rectilinear grids, i.e., a regularly-spaced rectangular domain. The majority of available software uses a specific discretization scheme, and converts Poisson’s equation into matrix form, as follows:

\[ \nabla \cdot \nabla f = F \rightarrow Af = F, \]  

(6.2)

where \( f \) and \( F \) are \( n \times 1 \) vectors:

\[ \nabla = \begin{bmatrix} \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \end{bmatrix}^T, \]  

(6.3)

\( n \) is the number of discretized points, and \( A \) is an \( n \times n \) matrix that is obtained by the numerical discretization method in use. Castillo-Grone’s Mimetic (CGM) difference operators are used here to construct a discrete analog of Poisson’s equation. CGM has performed well in various studies [29]. Abouali and Castillo have also shown that the discrete form of Poisson’s equation, using second-order CGM operators and linear interpolation, performs up to 25% better in terms of the accuracy of the results [3], relative to the conservative discretization method of curvilinear grids. This paper focuses on higher-order CGM operators in conjunction with higher-degree interpolating polynomials. Since, in Chapter 4 a baseline of performance is established, the other variants of the CGM operators will be compared to that established baseline.

The next section describes how matrix \( A \) is constructed. The CGM operators will also be tested in various conditions, and the results are presented.
6.2 Methods

6.2.1 Poisson’s Equation in Curvilinear Grids

Poisson’s equation can be written in vector form using the gradient and divergence operator, as follows:

\[ \nabla \cdot (\kappa \nabla f) = F, \tag{6.4} \]

where \( \kappa \) is the identity matrix. Poisson’s equation can also be written in matrix form, as follows:

\[ \nabla^T \kappa \nabla f = F. \tag{6.5} \]

To solve the above equation, we must first change the equation from the curvilinear physical domain to a regularly-spaced computational space. A summary of this procedure is provided here, but detailed descriptions of this procedure can be found in many manuscripts [30, 59, 69]. Assuming that the \((x, y, z)\) domain is transformed into \((\xi, \eta, \zeta)\) space, then the operators in these two domains are related to each other by \( \tilde{\nabla} = T \nabla \), where \( \tilde{\nabla} = \left[ \frac{\partial}{\partial \xi}, \frac{\partial}{\partial \eta}, \frac{\partial}{\partial \zeta} \right]^T \), and

\[ T = \begin{bmatrix} x_\xi & y_\xi & z_\xi \\ x_\eta & y_\eta & z_\eta \\ x_\zeta & y_\zeta & z_\zeta \end{bmatrix}. \tag{6.6} \]

\( x_\xi \), in the above equation, is shorthand for \( \partial x / \partial \xi \). By inverting this equation, we get \( \nabla = T^{-1} \tilde{\nabla} = \frac{1}{J} C \tilde{\nabla} \), where \( J \) is the Jacobian of the transformation matrix and \( C \) is the cofactor of the transformation matrix. By substituting this in equation (6.4), we can rewrite Poisson’s equation as:

\[ \nabla^T \kappa \nabla f = \tilde{\nabla}^T \tilde{\kappa} \tilde{\nabla} f = F, \tag{6.7} \]

where:
\[ \tilde{\kappa} = \frac{1}{J^2} C^T k C. \]

(6.8)

It should be noted that \( \tilde{\kappa} \) is no longer the identity matrix, and the off-diagonal entries are not necessarily zero; which means that Poisson’s equation will now be transformed into a fully elliptic equation in the computational domain, with all cross-derivatives present. This adds additional complexity to the solution of Poisson’s equation; For example, in a 2D curvilinear grid, \( \tilde{\kappa} \) can be written as follows:

\[
J^2 \tilde{\kappa} = \begin{bmatrix}
(y_\eta^2 + x_\eta^2) & -(y_\xi y_\eta + x_\xi x_\eta) \\
-(y_\xi y_\eta + x_\xi x_\eta) & (y_\xi^2 + x_\xi^2)
\end{bmatrix}. \tag{6.9}
\]

As a result, Poisson’s equation will be transformed into a diffusion problem, in which the diffusion coefficient is neither homogeneous nor isotropic. It can then be easily shown that when the grid is orthogonal (or near orthogonal) the off-diagonal elements are zero.

### 6.2.2 Castillo-Grone’s Mimetic (CGM) Difference Operators

The Castillo–Grone mimetic approach provides an exact equivalent of the gradient and divergence operators in a discrete domain [24, 29]. These discrete operators have been proven to satisfy all the properties of their continuous version in the discrete sense; are conservative; and provide the same accuracy in the entire domain, including the boundaries, without use of dummy or ghost nodes [4, 16, 26, 28]. For further details on these operators and how they are generated, refer to one of the provided references. The Castillo–Grone mimetic approach uses a staggered grid to calculate the gradient and the divergence, Figures 6.1 and 6.2 for a 1D representation, and Figures 6.3 and 6.4 for a 2D representation).

If the nodes in the physical domain are not regularly spaced, a transformation is needed. This is a simple procedure in 1D, and the physical grid is transformed into a
Figure 6.1: Staggered computation of the CGM gradient operator in 1D.

Figure 6.2: Staggered computation of the CGM divergence operator in 1D.

Figure 6.3: Staggered computation of the CGM gradient operator in 2D.
computational space, as shown in Figure 6.1. In 2D, as well as in cases where general curvilinear grids are used, more than one grid is required. The next section describes this procedure for 2D curvilinear grids.

6.2.3 Computational Grids in 2D Transformed Poisson’s Equations

Equation 6.4 describes Poisson’s equation in the physical domain. It was previously shown that the Laplacian operator in the physical curvilinear domain can be written as:

\[ \nabla^T \kappa \nabla = \tilde{\nabla}^T \tilde{\kappa} \tilde{\nabla} = L^{2D}; \quad (6.10) \]

therefore, \( L^{2D} \) is the Laplacian operator in the physical curvilinear grid, using operators in the computational domain. Hence, all the transformations are handled internally, with no need to transform the equation, or the right-hand side of Poisson’s equation. If the curvilinear Laplacian operator, i.e. \( L^{2D} \), is expanded in 2D, it can
then be written as follows:

\[
\nabla^T \kappa \nabla f = L^{2D} f = \begin{bmatrix}
L_{p1}^{2D} & 0 & 0 & 0 \\
0 & L_{p2}^{2D} & 0 & 0 \\
0 & 0 & L_{p3}^{2D} & 0 \\
0 & 0 & 0 & L_{p4}^{2D}
\end{bmatrix} \begin{bmatrix}
D_{\xi}K_{1,1}G_{\xi} + D_{\eta}K_{2,2}G_{\eta} + D_{\xi}K_{1,2}G_{\eta} + D_{\eta}K_{1,2}G_{\xi} \\
D_{\xi}K_{1,1}G_{\eta} + D_{\eta}K_{2,2}G_{\eta} + D_{\xi}K_{1,2}G_{\eta} + D_{\eta}K_{1,2}G_{\xi} \\
D_{\xi}K_{1,1}G_{\eta} + D_{\eta}K_{2,2}G_{\eta} + D_{\xi}K_{1,2}G_{\eta} + D_{\eta}K_{1,2}G_{\xi} \\
D_{\xi}K_{1,1}G_{\eta} + D_{\eta}K_{2,2}G_{\eta} + D_{\xi}K_{1,2}G_{\eta} + D_{\eta}K_{1,2}G_{\xi}
\end{bmatrix} f,
\]

(6.11)

where \( D = \begin{bmatrix} \partial / \partial \xi, \partial / \partial \eta \end{bmatrix} = [D_\xi, D_\eta] \) is the CGM divergence operator in the computational domain, and \( G = \begin{bmatrix} \partial / \partial \xi, \partial / \partial \eta \end{bmatrix} = [G_\xi, G_\eta] \) is the CGM gradient operator in the computational domain. The first term of the Laplacian operator can be expanded in matrix form, as follows:

\[
L_{p1}^{2D} = D_{\xi}^{CGM_{\text{Grid}}} \leftarrow U_{\text{Grid}} K_{1,1}^{U_{\text{Grid}}} G_{\xi}^{U_{\text{Grid}}} \leftarrow CGM_{\text{Grid}},
\]

(6.12)

where \( CGM_{\text{Grid}} \) represents the computational domain, i.e. transformed physical space, Figure 6.5; \( U_{\text{Grid}} \) is defined on the computational space, Figure 6.6; and \( D_{\xi}^{CGM_{\text{Grid}}} \leftarrow U_{\text{Grid}} \) is a \( N_p \times N_u \) matrix that computes the \( \partial / \partial \xi \) on \( CGM_{\text{Grid}} \), assuming the data is stored on \( U_{\text{Grid}} \), i.e. the 2D equivalent of a 1D CGM divergence operator as shown in Figure 6.2. Likewise, \( G_{\xi}^{U_{\text{Grid}}} \leftarrow CGM_{\text{Grid}} \) is an \( N_p \times N_u \) matrix that calculates the gradient along \( \xi \) axis on the \( U_{\text{Grid}} \) assuming the data is stored on \( CGM_{\text{Grid}} \), i.e. the 2D equivalent of the CGM gradient operator as shown in Figure 6.1. Finally, \( K_{1,1}^{U_{\text{Grid}}} \) is a \( N_u \times N_u \) diagonal matrix storing the transformation coefficients on \( U_{\text{Grid}} \). \( N_p = 2(M + N) + M \times N \) is the number of points on \( CGM_{\text{Grid}} \) and \( N_u = (M + 1) \times N \) is the number of points on \( U_{\text{Grid}} \). \( M \) and \( N \) are the number of cells along the \( \xi \)- and \( \eta \)-axis.

Likewise, one gets:

\[
L_{p2}^{2D} = D_{\eta}^{CGM_{\text{Grid}}} \leftarrow V_{\text{Grid}} K_{2,2}^{U_{\text{Grid}}} G_{\eta}^{V_{\text{Grid}}} \leftarrow CGM_{\text{Grid}},
\]

(6.13)

where \( V_{\text{Grid}} \) is shown in Figure 6.7.
The second two terms are a bit different, and are defined as follows:

\[ L_{p3}^{2D} = D_{\xi}^{CGM_{Grid}\rightarrow U_{Grid}} K_{1,2}^{U_{Grid}\rightarrow U_{Grid}} G_{\eta}^{U_{Grid}\rightarrow \eta_{Grid}} P_{\eta_{Grid}\rightarrow CGM_{Grid}}, \]

and:

\[ L_{p4}^{2D} = D_{\eta}^{CGM_{Grid}\rightarrow V_{Grid}} K_{1,2}^{V_{Grid}\rightarrow V_{Grid}} G_{\xi}^{V_{Grid}\rightarrow \xi_{Grid}} P_{\xi_{Grid}\rightarrow CGM_{Grid}}. \]

\( \xi_{Grid} \) and \( \eta_{Grid} \) are shown in Figures 6.9 and 6.8. \( P_{\xi_{Grid}\rightarrow CGM_{Grid}} \) and \( P_{\eta_{Grid}\rightarrow CGM_{Grid}} \) project, transform, or re-grid data from \( CGM_{Grid} \) to \( \xi_{Grid} \) and \( \eta_{Grid} \), respectively. These projectors are constructed as described in Chapter 3; hence, depending on the location of the nodes and the selected parameters they can interpolate, extrapolate, or fit a curve.

The above explanation can be confusing, as it takes time to determine how these different operators are inter-related. However, writing the operator out in detail helps to better explain its behavior, and also shows why using an orthogonal curvilinear grid over a non-orthogonal curvilinear grid (general curvilinear grid) is recommended. Recall that, once the orthogonal curvilinear grid is used, the off diagonal elements of the transformation matrix become zero. This means that \( K_{1,2}^{U_{Grid}} \) and \( K_{1,2}^{V_{Grid}} \) are now zero, so \( L_{p3}^{2D} \) and \( L_{p4}^{2D} \) also become zero (the only two terms that required interpolation or extrapolation). Therefore, if the data is stored on the CGM grid in the physical domain, and the grid is an orthogonal curvilinear grid, there is no need for interpolations or extrapolations; hence, no numerical errors are introduced into the solutions as a result of the interpolation or extrapolation step.

### 6.2.4 MATLAB® Command

The MATLAB® command that generates the discrete extended Laplace operator based on the CGM operators is:
Figure 6.5: 2D CGM grid in computational space and numbering.

Figure 6.6: 2D U grid in computational space and numbering.

Figure 6.7: 2D V grid in computational Space and numbering.
Figure 6.8: 2D $\eta$ grid in computational space and numbering.

Figure 6.9: 2D $\xi$ grid in computational space and numbering.
constructCGMLap(xn,yn,BC, OpOrder,InterpOrder,nInterp,verbose)

where:

- \( xn \) and \( yn \) are the coordinates of the nodal curvilinear grid in the physical domain, Figure 6.10,
- \( BC \) defines the boundary condition,
- \( OpOrder \) is the CGM operator order (default value is 2),
- \( InterpOrder \) is the degree of the interpolating polynomial (default is 1),
- \( nInterp \) is the number of nodes needed to construct the interpolator, and
- \( verbose \), is the logical variable that defines the level of the output to be printed on the screen.

\( BC \) is an \( n_b \times 2 \) matrix, where \( n_b = 2(M + N) \) is the number of nodes at the boundary, and \( M \) and \( N \) are the number of cells along the \( \xi \) and \( \eta \) directions. The first \( 2M \) entries are related to the bottom and top boundaries, i.e. \( \eta = 1 \) and \( \eta = N + 1 \), and the last \( 2N \) entries are related to the left and right boundaries, i.e. \( \xi = 1 \) and \( \xi = M + 1 \). The first column of the \( BC \) defines the \( \alpha \), and the second column defines the \( \beta \) coefficient in the Robin boundary condition, i.e.:

\[
\alpha f + \beta \nabla f \cdot \vec{n} = \gamma,
\]

(6.16)

where \( \vec{n} \) is the normal vector pointing outward at the boundary, FIGURE 6.11. If \( \beta \) is set to zero, i.e. the second column of the \( BC \), then Dirichlet’s boundary condition is obtained; and if \( \alpha \) is set to zero, i.e. the first column of the \( BC \), then Neumann
boundary condition is obtained. Once the command is executed, the resulting Lapla-
cian operator, which includes the mimetic boundary conditions, is calculated along
with the coordinates of the CGM grid, the cell centers, and the index of the points
that fall on the border.

6.3 Results and Discussion in 2D Cases

In order to determine the sparseness of the operators, as well as the amount of time
required to build them, a series of tests was performed on a uniformly-spaced rectan-
gular domain with $80 \times 80$ nodes. These tests showed that all operators require the
same amount of time to build, regardless of their order. For example, on a machine
with an Intel® i3 CPU and 16 GB of memory the second order operator took 14.7
[s], the fourth order operator took 15.1 [s], and sixth order operator took 15.59 [s].
Although this difference becomes larger, when the problem size, i.e. number of nodes
in the grid, is increased, it is still a negligible amount. However, the number of non-
zero elements increases depending on the order of the operators. In this example,
the second order operator had 31,521 non-zero elements, the fourth order operator
had 81,449 non-zero elements, and the sixth order operator had 130,429 non-zero
elements. This means that there was a 258% increase in non-zero elements when a
fourth order operators is chosen relative to the 2nd order operator and 414% increase when a sixth order operator is chosen. However, the density of the matrix, i.e. the number of non-zero elements to the total number of elements, did not reached even 0.5% for the densest operator, i.e. the sixth order operator. Additionally, the density of these matrices decreases the more nodes there are on the grid. This is summarized in the table (6.1).

Table 6.1: Required time in seconds and number of non-zero operator.

<table>
<thead>
<tr>
<th>Operator Order</th>
<th>Time [s]</th>
<th># non-zero</th>
<th>density</th>
<th>% increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd</td>
<td>14.7</td>
<td>31521</td>
<td>0.07</td>
<td>-</td>
</tr>
<tr>
<td>4th</td>
<td>15.1</td>
<td>81449</td>
<td>0.19</td>
<td>258</td>
</tr>
<tr>
<td>6th</td>
<td>15.59</td>
<td>130429</td>
<td>0.30</td>
<td>414</td>
</tr>
</tbody>
</table>

Here, the error is calculated at each node by subtracting the analytic exact values from those obtained numerically. Therefore, a measure of the error is required to
understand the true accuracy of the operators; three different measures were chosen, as follows:

- **Root Mean Square Error** or RMSE, calculated as follows:

  \[
  RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (f_i - f_i^*)^2},
  \]

  \[\text{Equation 6.17}\]

- **L}_2 \text{ Norm,} computed as follows:

  \[
  ||\text{Error}||_2 = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (f_i - f_i^*)^2},
  \]

  \[\text{Equation 6.18}\]

- **Max Error** or \(L}_\infty \text{ norm,} calculated as follows:

  \[
  ||\text{Error}||_\infty = \max_i |f_i - f_i^*|.
  \]

  \[\text{Equation 6.19}\]

In the above equations, \(n\) is the number of nodes, \(f_i\) is the numerical solution, and \(f_i^*\) is the analytic or exact solution. In order to test the accuracy, the right hand side of Poisson’s equation, i.e. \(\nabla \cdot \nabla f = F\), was set to:

\[
F(x, y) = \frac{\cos(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}} - \sin(\sqrt{x^2 + y^2}),
\]

\[\text{Equation 6.20}\]

where \(x, y \in [0, 2\pi] \times [0, 2\pi]\) and Dirichlet’s boundary condition was used. The exact solution, i.e. \(f\), would then be written as follows:

\[
f(x, y) = \sin(\sqrt{x^2 + y^2}).
\]

\[\text{Equation 6.21}\]

Figure 6.12 shows a sample solution to the above equation using the fourth order Castillo-Grone’s operators. The grid had 40 \times 40 nodes and an Algebraic Multi-Grid solver, implemented by Yvan Notay [93, 96, 97, 98], was used to solve the resulting system of equations. Figure 6.13 shows how the error changes when the operator
order is increased. These graphs were generated over a uniform rectangular grid with 20 × 20 nodes; therefore, \( n = 190969 \).

One property of each numerical method is the global operator order, which shows how the error decreases by the refining of the grid spacing. We used four different grid spacing, as follows:

- Grid 1: 10 × 10 \( \rightarrow dx = dy = 0.6981 \),
- Grid 2: 20 × 20 \( \rightarrow dx = dy = 0.3307 \),
- Grid 3: 40 × 40 \( \rightarrow dx = dy = 0.1611 \),
- Grid 4: 80 × 80 \( \rightarrow dx = dy = 0.0795 \).

Figure 6.14 shows how the error decreases when the grid spacing is refined or the number of nodes is increased. Although different error measures show different rates
of decrease, depending on the degree of grid refinement, nearly all show that the rate is greater in coarser grids and smaller in finer grids. There is only one exception: when the fourth-order operator is used with the max error. It appears that this error drops faster in finer grids than in coarser grids, figure 6.14(b).

Thus far, only Dirichlet boundary condition has been used; however, further tests were also performed to test for Neumann boundary condition. Neumann boundary condition is defined as:

$$\frac{\partial f}{\partial \vec{n}} = \nabla f \cdot \vec{n} = \gamma(x, y),$$  \hspace{1cm} (6.22)$$

where $\vec{n}$ is the normal vector to the boundary facing outward. To test the Neumann boundary condition, the same right-hand side function was used; however, at $y = 0$
and \( y = 2\pi \) Dirichlet’s boundary condition was used and at \( x = 0 \) and \( x = 2\pi \) Neumann boundary condition were used. The sparse matrix thus generated, i.e. the matrix \( A \) in \( \nabla \cdot \nabla f = F \rightarrow Af = F \), included the mimetic boundary conditions. Again, as the operator order increased the RMSE was lowered; however, in all cases where Neumann boundary condition was utilized, more errors resulted. This outcome suggests that higher-order operators are more susceptible to Neumann’s boundary condition, figure 6.15. Similar results were obtained when Dirichlet boundary condition was used at \( x = 0 \) and \( x = 2\pi \) and Neumann boundary condition was used at \( y = 0 \) and \( y = 2\pi \).

The Robin boundary condition was also tested. Robin boundary condition is defined as:
\[
\alpha f + \beta \nabla f \cdot \vec{n} = \gamma.
\] (6.23)

When \( \beta \) set to zero Robin boundary condition is reduced to Dirichlet boundary condition, and when \( \alpha \) set to zero Neumann boundary condition is obtained. Hence, Robin boundary condition is considered to be the most complete boundary condition. To test Robin boundary condition the right hand side function was changed to:

\[
F(x, y) = \frac{\lambda^2 e^{\lambda x}}{e^\lambda - 1},
\] (6.24)

where \( \lambda = -1 \) and \( x, y \in [0, 2\pi] \times [0, 2\pi] \). Dirichlet boundary condition was then applied at \( y = 0 \) and \( y = 2 \times pi \), and Robin boundary condition was applied at \( x = 0 \) and \( x = 2\pi \). The exact or analytic solution to this problem is therefore:

\[
f(x, y) = \frac{e^{\lambda x} - 1}{e^\lambda - 1}.
\] (6.25)
Figure 6.16 shows a sample solution to the above equation using a fourth order operator and $40 \times 40$ grid. As in the two previous cases, the error decreased when the operator order was increased. However, in previous cases, i.e. when Dirichlet and Neumann boundary condition were used, the sixth order operator did not show much of an improvement in accuracy; in fact, its accuracy was comparable to that of the fourth order operator. Yet, when Robin boundary condition was used, the sixth-order operators improvement in accuracy was considerable, relative to that of the fourth-order operator, Figure 6.17.

Figure 6.16: Sample numerical and analytic solution using fourth order operators for equation (6.25).

Once Robin boundary condition is used, the relation between the number of nodes and the operator order shows no significant change for the second order operator, i.e. the rate of improvement in the accuracy tends to slow down as the grid is refined. However, the fourth order operator shows a completely different pattern. As shown
Figure 6.17: The accuracy improves by increasing the operator order when Robin boundary condition is used.

In figure 6.18, the rate at which the accuracy increases as a result of the refining of the grid tends to be slight. The sixth-order operators also behave in a completely different manner. Although the rate of improvement in the accuracy slows down at first once Robin boundary condition is used, this slowdown is not as great as when Dirichlet boundary condition is used. However, once a very fine grid is used, i.e. an $80 \times 80$ grid, the error suddenly increases if $L_2$ norm is used and the improvement in the accuracy is not noticeable for $L_{\infty}$ or RMSE, Figure 6.18.

In order to better test the operators, a more challenging problem has been chosen. The right-hand side function is set to:

$$F(x, y) = (f_1 + f_2 + f_3 + f_4 - f_5 - f_6)/R^3,$$  \hspace{1cm} (6.26)
Figure 6.18: Improvement in the accuracy as the grid is refined once Robin boundary condition is used.

where:

\[
\begin{align*}
 f_1 &= 12R^2 - 40R^3, \\
 f_2 &= 90\pi x^4 \cos(\varphi(R)), \\
 f_3 &= 90\pi y^4 \cos(\varphi(R)), \\
 f_4 &= 180\pi x^2 y^2 \cos(\varphi(R)), \\
 f_5 &= 1800\pi^2 x^2 \sin(\varphi(R)) R^5, \\
 f_6 &= 1800\pi^2 y^2 \sin(\varphi(R)) R^5,
\end{align*}
\]  

(6.27)

where:

\[
\varphi(R) = 20\pi R^3, \quad (6.28)
\]
and:

\[ R = \sqrt{x^2 + y^2}. \]  

(6.29)

Dirichlet boundary condition was used at all boundaries and \( x, y \in [0, 1] \times [0, 1] \).

The analytic or exact solution to the above equation is:

\[ f(x, y) = 1 + 12R(x, y) - 10R(x, y)^2 + \frac{1}{2}\sin(\varphi(R(x, y))). \]  

(6.30)

A grid consisting of 120 \( \times \) 120 nodes was used to discretize the domain. A sample solution using the fourth order operators and its analytic solution is shown in Figure 6.19. This problem contains many oscillations and various frequencies, making it particularly challenging. As a result of these type of high-frequency oscillations in the solution, a very fine grid is required. The profile of both the numerical and analytic solution along the diagonal of the domain is shown in Figure 6.20. The two solutions appear to fall on each other. The error is also shown in Figure 6.20, which shows that the error is zero on the left, where there are few oscillations, but begins to grow as the solution starts to oscillate at a higher frequency. Note that most of the error is due to lack of proper sampling in the domain, i.e. the lack of grid resolutions. The error in that region can be reduced by refining the grid in that sector, which makes this a perfect example for an adaptive grid problem.

The operators were also tested on curvilinear grids with curved boundaries. A sample coarse resolution curvilinear grid is shown in Figure 6.21. The operators behaved exactly the same as in non-curvilinear grids. Figure 6.22 shows how RMSE changes with the operator order.

Testing the operators on non-smooth grids also provided interesting results. To obtain a non-smooth grid, a regularly spaced orthogonal non-curvilinear grid was
Figure 6.19: A hard sample problem with lots of oscillations.

Figure 6.20: Diagonal profile of the hard sample sample case and the error.

first generated on $[0, 2\pi] \times [0, 2\pi]$. Once the coordinates of the nodes were obtained, a Gaussian noise was added to the coordinates. Once the domain was discritized using $20 \times 20$ nodes, the lowest possible Signal to Noise Ratio (SNR), without causing the grid to fold on itself, was 25. This SNR was defined manually by trial and error using
only round integer numbers. However, the lowest possible SNR increased to 30 once the domain was discretized with $40 \times 40$ nodes. A sample noisy grid is shown in Figure 6.23. A sample numerical solution and the analytic solution is shown in figure (6.24).

It should be noted that Castillo et al. did study the non-smooth grids [27]; however, all the cases studied by them had perfectly straight boundaries, and only the internal nodes were allowed to move.

Since the grid is non-orthogonal, both the operator order and the degree of in-
Figure 6.23: Sample Noisy Grid ($SNR=25$).

Figure 6.24: Numerical and analytic solutions on the sample noisy grid.

terpolating polynomial affects the accuracy. Table 6.2 shows the RMSE and how it changes with the operator order and the degree of the interpolating polynomial. In the case of the non-smooth grids, the RMSE increases as the operator order increases, i.e. the higher order operators are more susceptible to non-smoothness in the grid.
However, increasing the degree of the interpolating polynomial causes a decrease in the RMSE, with one exception in the second order operator and fourth degree interpolating polynomial. Note that as the SNR increases, i.e. having a smoother grid, the results becomes very similar as seen in previous cases.

The minimum number of nodes was used to construct the interpolating polynomial in Table 6.2. An interpolating polynomial of degree $p_d$ has $(p_d + 1)^2$ coefficients that need to be determined; hence, the minimum number of nodes to form this polynomial would be equal to the number of coefficients. It is also possible to use more nodes than the minimum required to construct the interpolating polynomials; this is known as curve fitting and the resulting polynomial does not necessarily match the function value at each node. In order to understand the effect of using curve fitting instead of interpolation, we decided to use more points than needed. We decided to use $(p_d + 2)^2 - 1$ nodes. Table 6.3 shows the RMSE in this case. We clearly see that the RMSE has been considerably reduced. By increasing the operator order, the RMSE will still be increased; however, there would be no considerable changes in the accuracy once the degree of the interpolating polynomial was also increased, i.e. the RMSE becomes independent of the degree of the interpolating polynomial. Again, it should be noted that as the SNR increases the results becomes very similar to those found in the previous cases.

Table 6.2: RMSE on noisy grid using the minimum required number of nodes for interpolation.

<table>
<thead>
<tr>
<th>Operator Order</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0479</td>
<td>0.0161</td>
<td>0.0074</td>
<td>0.0229</td>
</tr>
<tr>
<td>4</td>
<td>0.0184</td>
<td>0.1042</td>
<td>0.0156</td>
<td>0.0128</td>
</tr>
<tr>
<td>6</td>
<td>0.5129</td>
<td>0.0855</td>
<td>0.0351</td>
<td>0.0154</td>
</tr>
</tbody>
</table>
Table 6.3: RMSE on Noisy Grid using more nodes than the minimum required for interpolation.

<table>
<thead>
<tr>
<th>Operator Order</th>
<th>Degree of Interpolating Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.0042</td>
</tr>
<tr>
<td>4</td>
<td>0.0057</td>
</tr>
<tr>
<td>6</td>
<td>0.0086</td>
</tr>
</tbody>
</table>

6.4 Conclusions in 2D

In previous studies, the accuracy of the extended Laplacian operator resulting from the second-order CGM operators and linear interpolation, were investigated, and a baseline was established showing that these operators perform better. In this chapter, that study is extended to higher-order CGM operators and higher-degree interpolating polynomials. Our results show that, in general, as the operator order increases, the accuracy of the numerical solution also increases. However, the amount of increase in accuracy achieved by switching from fourth-order operators to sixth-order operators is not significant. And, although most operators take approximately the same amount of time to be constructed (the difference is negligible), the amount of memory required to store the operators increases by a factor of nearly 1.6. Moreover, the fourth-order operators showed the highest rate of convergence for both fine and course grids.

Based on these factors, we recommend using the fourth-order operator. It was also shown that by increasing the degree of the interpolating polynomial, the accuracy of the numerical solution is also increased. However, if a curve fitting approach is used instead of interpolation, the accuracy increases significantly (in some cases by a factor of 10); more importantly, the accuracy becomes relatively independent of the degree of the interpolating polynomial in use. This finding is an important one, particularly involving cases in which non-smooth grids are being used.
In this study, an aggregated multi-grid solver was used to solve the resulting system of linear equations; however, we recommend studying the effect of various solvers with different preconditioners.

6.5 3D Curvilinear Grids

3D curvilinear grids complicates Matters. If the grid is non-orthogonal all the off-diagonal elements of \( \bar{\kappa} \) are present; which means that all the cross-derivatives are also present. Hence, in 3D, the Laplacian operator consists of 9 terms:

\[
\nabla^T \kappa \nabla f = L^{3D} f = D_{\xi} \bar{\kappa}_{1,1} G_{\xi} + D_{\eta} \bar{\kappa}_{2,2} G_{\eta} + D_{\zeta} \bar{\kappa}_{3,3} G_{\zeta}
\]

The \( CGM_{Grid} \) in 3D is shown in Figure 6.25. \( L_{p1}^{3D}, L_{p2}^{3D}, \) and \( L_{p3}^{3D} \) make use of \( U_{Grid}, V_{Grid}, \) and \( W_{Grid}, \) without needing any interpolation or extrapolations, Figure 6.26. However, \( L_{p4}^{3D} \) and \( L_{p5}^{3D} \) need to re-grid data from \( CGM_{Grid} \) onto \( \xi \eta_{Grid} \) and \( \xi \zeta_{Grid}, \) Figure 6.27, \( L_{p6}^{3D} \) and \( L_{p7}^{3D} \) need to re-grid data onto \( \eta \xi_{Grid} \) and \( \eta \zeta_{Grid}, \) Figure 6.28, and finally \( L_{p8}^{3D} \) and \( L_{p9}^{3D} \) need to re-grid data onto \( \zeta \xi_{Grid} \) and \( \zeta \eta_{Grid}, \) Figure 6.29.

Once again, if the grid is an orthogonal curvilinear grid, except the first three terms all the other terms are zero and there is no need for interpolations or extrapolations.
6.6 Results in 3D

We first tested the 3D operator on a regularly spaced grid. A coarse grid with $20 \times 20 \times 20$ cells was selected, where $(x, y, z) \in [0, 2\pi] \times [0, 2\pi] \times [0, 2\pi]$. The right-hand side was set to:

$$F(x, y) = \frac{\cos \left( \sqrt{x^2 + y^2} \right)}{\sqrt{x^2 + y^2}} - \sin \left( \sqrt{x^2 + y^2} \right),$$  \hspace{1cm} (6.32)$$

and Dirichlet boundary condition was used. The exact solution, i.e. $f$, is written as:

$$f(x, y) = \sin \left( \sqrt{x^2 + y^2} \right) + z^2.$$  \hspace{1cm} (6.33)$$

Table 6.4 demonstrates how different measures of error changes with respect to changing the operator order. The general trend is similar to that seen in 2D grids: the time needed to construct the Laplacian operator does not alter with a change in the operator order in any significant way. Hence, it can be assumed that they take approximately the same amount of time. In this example, as the operator order increases, the error drops, except in the case of the $||-||_\infty$ measure, where a considerable increase in error is seen with an increase from a fourth- to a sixth-order operator. From
the second-to-fourth-order, there is nearly a ninety-five percent reduction in the error; however, once the operator order is increased to the sixth order, only a thirty-two percent reduction is evident.

To further test the operator the right-hand side was set to:

\[
F(x, y) = \frac{\cos\left(\sqrt{x^2 + y^2}\right)}{\sqrt{x^2 + y^2}} - \sin\left(\sqrt{x^2 + y^2}\right) \\
+ 2\cos((z - \pi)^2) - 4(\pi - z)^2 \sin\left((z - \pi)^2\right),
\]

(6.34)
Figure 6.27: ξη and ξζ grids and node numbering.

Table 6.4: Effect of operator order in 3D regularly spaced cubes in Example 1.

<table>
<thead>
<tr>
<th>Measure of Error</th>
<th>Operator Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.0030822</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

with an analytic solution of:
Table 6.5 shows the changes in error for the second example. In these examples, the errors are much higher than in the previous examples, suggesting that the grid resolution is too low; however, the same trend is seen (i.e., a decrease in error following an increase in the operator order) here, as in the previous examples. When the grid resolution was increased to $40 \times 40 \times 40$ and the errors calculated, Table 6.6, increase...
in the grid resolution caused a considerable decrease in the error. The second norm of
the error shows a significant improvement when the operator order is increased from
2-to-4. Although the time needed to construct the operator did not change when
the operator order was increased, the time required to solve the system changed
substantially. Still, the reduction in error is so advantageous that it justifies the extra
time needed to solve the system. This result is an example of the benefits of studying
the effects of using different solvers with different preconditioners.

Table 6.5: Effect of operator order in 3D regularly spaced cubes in Example 2.

<table>
<thead>
<tr>
<th>Measure of Error</th>
<th>Operator Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.14405</td>
</tr>
<tr>
<td>$|\cdot|_2$</td>
<td>11.9303</td>
</tr>
<tr>
<td>$|\cdot|_\infty$</td>
<td>0.032656</td>
</tr>
</tbody>
</table>

To further test the operator in 3D curvilinear grid, the right-hand side of the
equation is set to:
Table 6.6: Effect of operator order in 3D regularly spaced cubes in Example 2 using a higher resolution grid.

<table>
<thead>
<tr>
<th>Measure of Error</th>
<th>Operator Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.023329</td>
</tr>
<tr>
<td>$|\cdot|_2$</td>
<td>5.6819</td>
</tr>
<tr>
<td>$|\cdot|_\infty$</td>
<td>0.043338</td>
</tr>
</tbody>
</table>

\[
F(x, y, z) = \frac{2R^2(x, y, z) \cos(R(x, y, z)) - R^3(x, y, z) \sin(R(x, y, z))}{R^3(x, y, z)},
\]

where:

\[
R(x, y, z) = \sqrt{x^2 + y^2 + z^2}.
\]

The analytic solution would be:

\[
f(x, y, z) = \sin\left(\sqrt{x^2 + y^2 + z^2}\right).
\]

A curvilinear grid with $80 \times 20 \times 10$ nodes was generated. Table 6.7 shows how the error changes when the operator order. Again, the same behavior is observed as was seen in previous cases. Later, the grid was refined to have $140 \times 30 \times 15$ nodes; as expected, the errors were reduced in all cases, and the same behavior was seen with respect to choosing the operator order, Table 6.8. Figure 6.30 shows the sample grid and how the actual solution compares to the analytic solution.

As previously explained, if the curvilinear grid is completely orthogonal, the transformation coefficient for the cross-derivatives becomes zero, which saves computation time, increases accuracy, and makes the entire process easier. And, although a fully orthogonal grid is impossible in reality, it is still suggested to make grids as orthog-
Table 6.7: Effect of operator order in 3D curvilinear grid.

<table>
<thead>
<tr>
<th>Measure of Error</th>
<th>Operator Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.0035243</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.8: Effect of operator order in 3D curvilinear grid with higher resolution.

<table>
<thead>
<tr>
<th>Measure of Error</th>
<th>Operator Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.0016792</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.30: An example of 3D curvilinear grid and the solution.

Thus far, cross derivatives have been ignored, and it has been assumed that the curvilinear grid in this example is very close to an orthogonal grid; however, it is a coarse grid.

Thus far, cross derivatives have been ignored, and it has been assumed that the
effect of these terms are negligible. In order to study the effect of cross-derivative terms on the accuracy, the terms were calculated, and then added to the Laplacian operator (1st-, 2nd-, and 3rd-degree polynomials were utilized). And, instead of interpolation, a curve-fitting approach was used, i.e. to construct the projector, more than the minimum number of required points was used.

Table 6.9 shows the effect of the operator degree and the interpolating polynomial. As in 2D cases, the error is nearly the same when a curve-fitting approach is used to project data from one grid onto another. The error improved consistently with an increase in the degree of the polynomial, but not by a significant amount.

By comparing the results we see that, once a second-degree operator is used, the effect of the cross-derivatives is almost negligible. However, the improvement in the accuracy for higher -order operators, i.e. fourth and sixth, is quite significant. For example the ratio of RMSE for a second-order operator when cross-derivatives are ignored versus a case when the terms are not ignored is approximately 1.0004 (i.e., negligible). However, this ratio increases to 4.6838 for fourth-order operators, and to 25.907 for sixth-order operators. This shows that the effects of these cross-derivatives are more pronounced for higher-degree operators. Still, for lower-degree operators, cross-derivatives can be ignored once the grid is near orthogonal. Our test has shown that cross-derivatives can have a significant impact on the accuracy of higher-order operators; yet, to the best of our knowledge, these cross-derivatives are ignored in the majority of codes available for ocean and atmosphere models.

6.7 Software Availability

Software and MATLAB codes from this project have been made available free of charge for academic and non-commercial use online at MATLAB File Exchange Cen-
Table 6.9: Effect of interpolation order on the operator in 3D curvilinear grids.

<table>
<thead>
<tr>
<th>Interp. Order</th>
<th>Measure of Error</th>
<th>Operator Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RMSE</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0.0035229</td>
<td>9.7487e-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>RMSE</td>
<td>0.0035208</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RMSE</td>
<td>0.0035204</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The link to download the code is:


We recommend beginning with one of the test scripts in order to become acquainted with how to use the code. If you use this code in any form, you are expected to acknowledge and reference the above link.
Part III

Applications in Geophysical fluid models
Chapter 7

Shallow Water Equations
Implemented on GPUs

7.1 Introduction & Governing Equations

Shallow Water Equations (SWE) are obtained by integrating the Navier-Stokes’ equation in the vertical direction. The use of shallow in the name can be misleading, as these equations can be used in simulating both deep and shallow regions; in fact, shallow relates only to the scale. In general, the vertical scale of the motion in atmospheric and oceanic flows is much smaller than the horizontal scale of the motion; therefore, they have a much smaller length scale in the vertical than in the horizontal. Hence, the shallow in the name: the vertical scale of the motion is much smaller than the horizontal scale of the motion. It should be noted that, as the flow depth increases, a smaller time scale must be used to keep the model stable.

7.1.1 Continuity Equation

The first SWE is the continuity equation, which is written as follows:

\[
\frac{\partial h}{\partial t} + \frac{\partial hu}{\partial x} + \frac{\partial hv}{\partial t} = S_i - S_o,
\]

(7.1)

where \(u\) is the eastward velocity component, or the velocity component along the
Figure 7.1: Shallow Water Equation (SWE) variables in vertical profile.

x-axis; \( v \) is the northward velocity component, or the velocity component along the y-axis; \( h \) is the water depth; \( S_i \) is the source term; and \( S_o \) is the sink term.

Currently, the code developed in this dissertation does not support a wetting and drying scheme; however, in order to facilitate the addition of such a scheme, the continuity equation should be written in terms of the water table height. In this case, the bathymetry, \( B \), and the water table height, \( w \), must both be measured from the same reference level. We employed the Geoid reference level, which is typically chosen, see Figure 7.1 for a more detailed explanation of these variables. Here, \( h = w - B \), with the continuity equation rewritten as follows:

\[
\frac{\partial w}{\partial t} + \frac{\partial hu}{\partial x} + \frac{\partial hv}{\partial t} = S_i - S_o + \frac{\partial B}{\partial t}, \tag{7.2}
\]

where, if the morphological changes in the bottom floor are ignored, the continuity equation can then be simplified to:

\[
\frac{\partial w}{\partial t} + \frac{\partial hu}{\partial x} + \frac{\partial hv}{\partial t} = S_i - S_o. \tag{7.3}
\]
7.1.2 Momentum Equations

The u-momentum equation in its conservative form can be written as follows:

\[
\frac{\partial (hu)}{\partial t} + \frac{\partial h u^2}{\partial x} + \frac{\partial h u v}{\partial y} = -gh \frac{\partial B}{\partial x} - \frac{gu \sqrt{u^2 + v^2}}{C_z^2} + \tau_w^u, \quad (7.4)
\]

where \( g = 9.81 \ [ms^{-2}] \) is the Earth’s gravitational acceleration constant, \( C_z \) is the Chezy constant, and \( \tau_w^u \) is the forcing resulting from the wind stress. Likewise, the v-momentum equation can be written as follows:

\[
\frac{\partial (hv)}{\partial t} + \frac{\partial h u v}{\partial x} + \frac{\partial h v^2}{\partial y} = -gh \frac{\partial B}{\partial y} - \frac{gv \sqrt{u^2 + v^2}}{C_z^2} + \tau_w^v. \quad (7.5)
\]

Note that the momentum equation is written in its conservative form. All variables are defined in the physical domain; however, the calculation is done in the computational domain. The standard approach is to transform all velocity components in the computational domain, and rewrite the SWE equation using only the divergence and gradient defined on the computational domain. However, this would produce numerical errors and compromise the conservation in conventional curvilinear grids [45]. Thus, to reduce this error, all the variables are kept in the physical domain and the operators are transformed internally.

7.1.3 De-singularizing

Note that both momentum equations compute the changes in \( hu \) and \( hv \); however, to obtain \( u \) and \( v \), we must first divide the them by \( h \), i.e., the water depth. This can result in \( h \) becoming a very small number, and, due to finite precision available on computers, this division can be erroneous. To avoid such situations, the water depth must first be compared to a user-defined \( d_{crit} \). If \( h > d_{crit} \), then:
\[(u, v) = (hu, hv)/h, \quad (7.6)\]

otherwise, i.e. \(h \leq d_{\text{crit}}\); the velocity components are calculated as follows:

\[
(u, v) = (hu, hv) \cdot \left(\frac{\sqrt{2h}}{\sqrt{h^4 + \max(h^4, K)}}\right). \quad (7.7)
\]

\(K\) is also a user defined value. The choice of \(K\) has a significant impact on both the performance and accuracy. If it is too large, it will dampen the velocity and, and if it is too small, a smaller time step will be required to keep the scheme stable. Kurganov and Petrova [72, 73] chose \(K\), as follows:

\[
K = \max(\Delta x^4, \Delta y^4), \quad (7.8)
\]

however, Brodtkorb [23] proposed the following method to choose \(K\):

\[
K = K_0 \max(1, \min(\Delta x, \Delta y)), \quad (7.9)
\]

where \(K_0 = 10^{-2}\) for single precision calculations, and it is set to an even smaller value for double-precision calculations.

### 7.1.4 Bottom Roughness

The roughness of the surface can be provided in the model by defining the Chezy coefficient, \(C_z\). The code reads in a 2D array of numbers for \(C_z\); therefore, the user have the possibility to vary the roughness depending on the location. Sometimes, it is easier to use Manning coefficients. In case where Manning coefficients is available, the user can provide the Manning coefficient and the Chezy coefficient is calculated as follows:
\[ C_z = \frac{h^{1/6}}{n}. \quad (7.10) \]

It is also possible to calculate Chezy coefficient using the White Colebrook equation, which is written as follows:

\[ C_z = 18 \log_{10} \left( \frac{12h}{K_s} \right), \quad (7.11) \]

where \( K_s \) is the Nikuradse roughness length. All these options are available to the user. The user also have this option to apply a constant bottom roughness throughout the field. To keep the performance of the code and avoid if-clauses in the code, which will compromise the performance, this must decided pre-compilation.

### 7.1.5 Numerical Scheme and Solution Strategy

Castillo-Grone’s Mimetic operators are used to discretized the equations spatially and the RK3 scheme is used to discretize the system in time. There are two approaches to advance the equations. The first approach is to advance each equation one after each other. This would help to artificially decouple these three equations but it results to interleaving the variables in time. However, for better quality and accuracy, it was decided to solve all the three equations together and simultaneously.

Commonly it is believed that using a C-Grid approach is a better choice for such equations [44]. In C-Grid approach, also known as staggered grids, \( u \) component is stored at the middle of the edges that are perpendicular to \( x \)-axis, \( v \) is stored at the middle of the edges that are perpendicular to \( y \)-axis, and \( w \) is stored at the cell centers. This approach, i.e. C-Grid, is indeed providing many benefits, such as eliminating the need for interpolation, as long as the equations are written in non-conservative forms and the Coriolis terms are ignored. Once a curvilinear grid is used, one usually replaces \((u, v)\) by \((\tilde{U}, \tilde{V})\) to be able to use the C-Grid. \( \tilde{U} \) and \( \tilde{V} \)
are the transformed velocity in computational domain, whereas \( u \) and \( v \) are velocity component in the physical domain. As explained by Ferziger [45], this approach compromises the conservative properties of the equation. Ferziger provides a very nice explanation on how C-Grids loose their benefits and attractiveness once the general curvilinear grids are used. He also explains why it is indeed even better to not transform the variables and use an A-Grid approach once general curvilinear is used. For the same reason, and the author past experience also shows that using the A-Grid in general curvilinear grids has many advantages over C-Gird, the least of all is that once A-Grid is used in general curvilinear grids, there are less interpolation needed, completely the opposite of the case once the Cartesian grids are in use. Therefore, it was decided to use an A-Grid approach where variables are defined at the cell centers.

The general strategy for solving the above mentioned equations can be summarized as follows:

1. Interpolate variable from the cell centers to the middle of the edges in order to calculate fluxes.

2. Calculate the rhs for the two momentum equations and the continuity equations

3. Advance the the variables

4. Apply the boundary conditions

Note that the RK3 is used for the time integration, which has three steps. Therefore, in order to advance time from \( t_n \) to \( t_{n+1} \) one has to repeat the above steps three times. Since the grid is not changing the interpolant is precomputed using the packages explained in 3; therefore, this step is only a sparse matrix computation and

117
is relatively very fast. It is made even faster once it is implemented on GPUs. All
interpolants are generated using the $p_d = 3$, i.e. cubic interpolation.

### 7.2 GPU Implementations

Central Processing Units (CPUs) have come a long way since their inception: they
have become extraordinarily stronger and more efficient with the passage of time.
One factor that determines CPU strength is its clock speed. The higher the clock
speed, the more computations can be performed by a CPU in a given amount of time.
However, since 2005, there has been no improvement in the clock speed of CPUs. Due
to current technological limitations, it is virtually impossible to increase the speed of
CPUs. Therefore, to make stronger CPUs, companies are now fitting more and more
CPUs on a single chip, known as multicore CPUs. This enables the user to harness
multiple CPUs simultaneously, in order to carry out more computations at one time,
termed parallel computation or parallel processing.

There are two general approaches for parallel processing: in the first, all com-
puting cores reside in a single machine, and each core has access to all the data;
in the second, the computing cores are on separate machines or nodes, and must
communicate with each other via network connections.

Graphics Processing Units (GPUs) were originally designed to offload portion of
the computations from the CPU that are required for displaying graphics. Therefore,
the CPU has more time to complete other tasks. Due to their different design it
is possible to cluster many computing cores on one GPU. As a result, users were
interested to perform all sort of computation on the GPUs and instead of merely
graphics-related. The term General Purpose GPU (GPGPU) was coined; and the
GPGPU has since become the center of the high performance computing (HPC) and
many supercomputers around the world are equipped with these devices.

For example, an Intel i7 extreme edition processor costs almost $1100 and it provides 6 cores; while an NVIDIA GTX Titan GPU costs approximately the same and provides 2688 cores. Although it should be mentioned that cores on CPUs are much stronger than the cores on GPUs; however, the sheer number of cores on GPUs make their performance significantly better.

There is yet another important difference between the cores on CPUs and those on GPUs: CPU cores are designed based on Multiple Instruction Multiple Data (MIMD) architecture, while GPU cores feature the Single Instruction Multiple Threads (SIMT) approach. This means that CPU cores are able to act independently, i.e., while one core is evaluating a sine function another can simultaneously evaluate a cosine function; in the case of GPUs, all cores must evaluate the same function, though the data on each core can vary. For example: If half the cores on a given GPU are needed to evaluate a sine function, and the other half needed to evaluate a cosine function, one task must be completed before the other can begin, resulting in half of the cores remaining idle at any one time. This has a significant impact on GPU performance, and substantially increases the amount of time needed to perform a task. Despite this fact, GPU performance is still faster than that of a CPU. While coding the SWE solver for this thesis all of these items were taken into consideration, and the code written in such a way that most of the if-clauses were either removed, or were decided during pre-compilation.

An additional difference between CPUs and GPUs is the implementation of the cache memory. Older versions of the GPU did not contain cache memory; therefore, it was suggested that read-only fields be mapped as textures, which were then cached, and later provided a higher throughput of data. As a result, many fluid solver codes mapped the data field at time $t_n$ as a texture field; however, with the advent of the
newer Fermi and Kepler NVIDIA GPUs, which are equipped with cache memory, users are no longer required to do this.

Many studies have already established that a good written code on a GPU can easily outperform a highly optimized code on a CPU [7]; therefore, the timing provided in this paper is only provided to illustrate that the GPU code developed here is also able to confirm past findings, and uses GPU resources in an optimum manner. See [99, 100, 101, 102, 103] for more information on GPU structure and memory patterns.

7.2.1 Preprocessing and The Input/Output file

All the inputs and required parameters are read from a single NetCDF file. The output is also stored in the same NetCDF file. As a result, NetCDF is required during compiling and linking.

7.3 Results

7.3.1 Timing CPU versus GPU

One of the problems associated with GPUs is the time needed to transfer or upload data from the host memory into the GPUs memory. The data must reside in the GPUs memory before it can perform any calculation, and the transfer between the host and device can be a time-consuming task. In order to optimize this transfer and reduce it as much as possible, the user must make sure there are enough computations being done on the data being sent to the GPU, and refrain from downloading anything back if it is not needed.

The SWE code developed for this thesis requires transfer of data on only two occasions: during the initialization phase of the code, after everything has been read from the disk; and when the user has requested that the data be written to the disk.

The first part of the program to be tested is the interpolation. Since the grid being
utilized is structured, we decided to fix the location of the participating points on the source grid to interpolate onto the destination grid. So, instead of storing the entire sparse matrix and the information on the rows and columns of the nonzero entries, we decided to store only the nonzero entries and compute the location based on the known cell index, i.e., $cell_i$ and $cell_j$. To do this, we first developed a special CUDA kernel to perform this matrix multiplication. To test how well this kernel worked, a grid with $200 \times 200$ cells was selected. Both the CPU and GPU version were timed; the timing on the CPU includes only the computation, i.e., the interpolation, while the timing on the GPU includes the time needed to transfer data to the GPU, perform the computation, and download the results. Note that it is not necessary to transfer data with every iteration. Therefore, we decided to perform multiple interpolations before downloading the results. Two test cases were designed. In the first case, 20 interpolations were performed before downloading the results; and in the second case, 40 interpolations were performed. The CPU version of the code required 78 seconds for the first case and 150 seconds for the seconds case; the time needed for the GPU and its speed is shown in Tables 7.1 and 7.2.

Table 7.1 and Table 7.2 show that the speed up improved when there was more computing to be done. Moreover, using the predefined shared memory also improved the speed up. This suggests that speed up is a function of the requested configuration. It is possible to achieve higher speed up once the user decides to store the results less frequently.

To test the entire code, i.e. the SWE solver, a rotated domain with $205 \times 205$ nodes was generated. Both the CPU and GPU codes simulated one hour and the time needed to store the results was ignored. Table 7.3 shows the timing between the CPU and GPU.

All of the timing results mentioned above are the average of 10 cases. The code
Table 7.1: Timing the interpolation kernel on GPU (20 interpolation before data transform).

<table>
<thead>
<tr>
<th>Block Size</th>
<th>4 × 4</th>
<th>5 × 5</th>
<th>4 × 8</th>
<th>6 × 6</th>
<th>7 × 7</th>
<th>8 × 8</th>
<th>9 × 9</th>
<th>10 × 10</th>
<th>11 × 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurable Shared Memory GTX 480</td>
<td>13.25</td>
<td>10.92</td>
<td>10.07</td>
<td>11.18</td>
<td>11.77</td>
<td>11.61</td>
<td>11</td>
<td>13.32</td>
<td>11.17</td>
</tr>
<tr>
<td>Predefined Shared Memory GTX 480</td>
<td>13.25</td>
<td>10.95</td>
<td>10.04</td>
<td>11.15</td>
<td>11.81</td>
<td>11.4</td>
<td>10.68</td>
<td>13.19</td>
<td>11.62</td>
</tr>
</tbody>
</table>

was run for 12 times. The maximum and minimum time was dropped out and the remaining 10 samples were averaged. As previously mentioned, many studies have already shown that the GPUs compute faster than CPUs. That’s a known fact and it was reconfirmed here. Therefore, instead of focusing on how much faster they are, it is very common to just report how much data the GPU can process in a given time. From now on, instead of reporting the CPU time and the GPU time, we will only report how much faster than real time the GPU was able to process. The timing will also include the time needed to store the results on the storage device. However, to reduce the effect of time needed to store the results on disk, an asynchronous approach is used. To test these effects the same code simulated two hours of simulation with various frequencies of writing the outputs and the timings are shown in 7.4.
Table 7.2: Timing the interpolation kernel on GPU (40 interpolation before data transform).

<table>
<thead>
<tr>
<th>GPU Timing</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Size</td>
<td>4 \times 4</td>
<td>5 \times 5</td>
<td>4 \times 8</td>
<td>6 \times 6</td>
<td>7 \times 7</td>
<td>8 \times 8</td>
<td>9 \times 9</td>
<td>10 \times 10</td>
<td>11 \times 11</td>
</tr>
<tr>
<td>Configurable Shared Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTX 480</td>
<td>22.8</td>
<td>18.23</td>
<td>16.57</td>
<td>18.89</td>
<td>20.35</td>
<td>20.13</td>
<td>18.98</td>
<td>23.81</td>
<td>19.3</td>
</tr>
<tr>
<td>C2050</td>
<td>28.54</td>
<td>22.8</td>
<td>20.69</td>
<td>23.02</td>
<td>25.6</td>
<td>24.98</td>
<td>21.76</td>
<td>29.95</td>
<td>24.1</td>
</tr>
<tr>
<td>Predefined Shared Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTX 480</td>
<td>22.78</td>
<td>18.29</td>
<td>16.51</td>
<td>18.72</td>
<td>20.24</td>
<td>19.57</td>
<td>17.51</td>
<td>23.12</td>
<td>20.46</td>
</tr>
<tr>
<td>C2050</td>
<td>28.51</td>
<td>22.85</td>
<td>20.55</td>
<td>23.15</td>
<td>25.46</td>
<td>24.79</td>
<td>21.59</td>
<td>29.6</td>
<td>24.03</td>
</tr>
<tr>
<td>Speed up</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block Size</td>
<td>4 \times 4</td>
<td>5 \times 5</td>
<td>4 \times 8</td>
<td>6 \times 6</td>
<td>7 \times 7</td>
<td>8 \times 8</td>
<td>9 \times 9</td>
<td>10 \times 10</td>
<td>11 \times 11</td>
</tr>
<tr>
<td>Configurable Shared Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTX 480</td>
<td>6.6</td>
<td>8.2</td>
<td>9.1</td>
<td>7.9</td>
<td>7.4</td>
<td>7.5</td>
<td>7.9</td>
<td>6.3</td>
<td>7.8</td>
</tr>
<tr>
<td>C2050</td>
<td>5.3</td>
<td>6.6</td>
<td>7.2</td>
<td>6.5</td>
<td>5.9</td>
<td>6.0</td>
<td>6.9</td>
<td>5.0</td>
<td>6.2</td>
</tr>
<tr>
<td>Predefined Shared Memory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTX 480</td>
<td>6.6</td>
<td>8.2</td>
<td>9.1</td>
<td>8.0</td>
<td>7.4</td>
<td>7.7</td>
<td>8.6</td>
<td>6.5</td>
<td>7.3</td>
</tr>
<tr>
<td>C2050</td>
<td>5.3</td>
<td>6.6</td>
<td>7.3</td>
<td>6.5</td>
<td>5.9</td>
<td>6.1</td>
<td>6.9</td>
<td>5.1</td>
<td>6.2</td>
</tr>
</tbody>
</table>

Table 7.3: Timing SWE solver.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Time [ms]</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel i3</td>
<td>155843</td>
<td>-</td>
</tr>
<tr>
<td>GTX 480</td>
<td>14104</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 7.4: Timing SWE Solver on GTX 480.

<table>
<thead>
<tr>
<th># Snapshots</th>
<th>Write Interval [m]</th>
<th>Total Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>1</td>
<td>28725</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>28256</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>28248</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>28229</td>
</tr>
<tr>
<td>1</td>
<td>120</td>
<td>28220</td>
</tr>
</tbody>
</table>
7.3.2 Validation

The first test in SWE is to check if the scheme is well-balanced. Fortunately there is an easy test for that. A channel with 3000m width and a length of 15000m was descritized using $dx = dy = 75m$. This will result into a grid with $45 \times 205$ nodes.

The bottom bathymetry was set to have a slope of 0.001 along the x-axis, i.e. the length of the channel. Velocity was initialized to be zero, i.e. fluid is at rest at $t = 0$. The water table was set to be at the same height throughout the entire domain. The periodic boundary condition was used everywhere. The code simulated 10 hours. Since there are no forcing, the water table is at the same elevation everywhere, and the fluid started at rest, there should not have been any flow generated. The model output was checked at the end of simulating 10 hours. As expected there was no flow generated anywhere and the water table appeared undisturbed. This shows that the different terms in the model balance each other.

To better assess the accuracy of the code and validate its results, another test was performed. The same channel as described above was used. However, in this second test, the water table was given the same slope as the bottom bathymetry, resulting to a constant depth of 5 meters throughout the channel. The boundary condition is set to be periodic for u component of the velocity and the v component of the velocity is assumed to be zero at the boundaries. v was initialized to zero; however, u was initialized to be $1 \text{ m s}^{-2}$. The water table at the boundary is set in such a way that the water depth would be 5 meters at the boundaries. Various numerical, theoretical, and lab experiments have all confirmed that under such conditions the flow should reach a constant velocity determined by:

$$u = C_z\sqrt{SD},$$

(7.12)
where $S$ is the bed-slope, $D$ is the water depth, and $C_z$ is the Chezy roughness parameter. Different $C_z$ was tested. In all cases the model simulated 5 hours. Amazingly, it was seen that in all cases the code quickly reached the desired velocity without any oscillations, Figure 7.2. The error was zero. Figure 7.2(b) shows the first 20 minutes of the simulation with temporal resolution of 15 seconds, i.e. every 15 seconds one output is stored to the file. It can be seen that $u$ component of velocity starts to increase to reach the level that it should be without any oscillations. In many other numerical models, during the same test many oscillations are seen and typically the velocity did not reach the exact desired value [21, 132, 133]. Moreover, in these studies it was seen that with the passage of time the deviation became slightly larger. The contour line for water table is shown in Figure 7.3.

Figure 7.2: Changes of error by increasing operator order
7.3.3 Rotated Domain

A domain with $205 \times 205$ nodes was generated. This grid was orthogonal; however, it was rotated by 45 degrees around the center. So that the velocity components do not align with the grid cells, Figure 7.4. Note that $u$ is along the x-axis and $v$ is along the y-axis; therefore, they do not match the grid orientation and the curvilinear computations are necessary.

Both components of the velocity were initialized to zero and the water table was initialized with the following function:

$$w = \frac{1}{\sqrt{1 + e^{80r}}}, \quad (7.13)$$

where $r$ is the normalized distance from the center of the domain. All boundary conditions are periodic. The water table after 5 minute of simulation is shown in Figure 7.5.

7.3.4 Long Channel With Island

One of the goals when using a curvilinear grid is to match the boundaries of the domain properly. Yet, in cases where the domain includes an island in the middle of the ocean, one is required to produce multiple domains, and then link them together. Using multiple domains is beyond the scope of this research; however, the ability to include a few small islands in the simulation, a land/sea mask was added to the
Figure 7.4: Orthogonal Rotated Domain.
scheme. Fluxes were automatically set to zero for all cells defined as dry cells. Note that there was no wet or dry scheme implemented; therefore, all cells should stay either dry or wet for the entire simulation. Also note that, having a dry cell in the middle of the domain is a source of discontinuity in the solution. CGM difference operators, like all other difference operators, are to some extent sensitive to the discontinuity in the solution. Because of this, extra care must be taken so that the model does not become unstable.

To test such situation, a channel, 20 km long and 3 km wide, was discretized using $dx = dy = 50 \text{ m}$. A periodic boundary condition was applied for both $u$ and $v$ component of the velocity. The bathymetry was sloped, $s = 0.001$, along the channel length, i.e. $x$-axis, and $w$ was set to provide constant pressure head at the beginning and the end of the channel. A cubic island with length of 600 m x 600 m was defined in the middle of the channel. No-Slip boundary condition is set along the edges of the island and no flux condition is set for $w$. No computation is performed over dry cells though. On NVIDIA GTX 480 it took 216 seconds to simulate 24 hours of simulation. Figure 7.6 shows the results after 24 hours of simulation with $dt = 1 \text{ [s]}$.

Figure 7.6 does not show any creation of Karman Vortices even after 24 hours of simulation. This could be due to too much dissipation or grid resolution. Here the reason for not seeing these vortices is the low resolution. To show this, a finer grid with $dx = dy = 10 \text{ [m]}$ was selected. Also $C_z$ was set to 20; therefore, we are introducing even more dissipation. To keep the model stable $dt$ was set to 0.1 $\text{ [s]}$. The 2 hours of simulation took about 15 minutes on NVIDIA GTX 480. Note that despite all calculation being double precision, this device is best for single precision calculation and is not even optimized for double precision computation, yet it was possible to achieve 8 times faster than real time. Every 30 seconds a snapshot was written to the disk. By playing these snapshots, i.e. creating a movie, the karman
vortices could clearly be seen. A snapshot at the end of the 2 hours is shown in Figure 7.7.

### 7.3.5 Monterey Bay

Another test case is the Monterey bay. The bathymetry of the Monterey bay is shown in Figure 7.8. Due to the presence of very deep underwater valleys in the bathymetry, the simulation of this region is rather challenging. The fast changing bathymetry results into very steep slopes; making it difficult in the momentum equation to balance the bottom slope term properly. One solution is to use a much higher resolution in those region. We tried a 500 m grid resolution and still it was not fine enough to handle those slopes properly; as a result, the imbalance in the momentum equation generated big waves which causes the model to collapse the moment those very high waves reached the shallower region. This suggest that the current model is too sensitive to bottom bathymetry and special care is needed for those regions.

Although not realistic, the bottom bathymetry was set to a constant depth and a grid with 210 × 205 nodes was generated. The fluid was initialized to be at rest at time zero. At the north boundary a sinusoidal function with an amplitude of 1 m was set as boundary condition for the water table; and at the south boundary another sinusoidal function was set, which had the same frequency but half the amplitude. The function at the northern boundary and the southern boundary had 180 degree phase shift. At the west boundary the Neumann’s boundary condition was assigned and at the east boundary (the shore line) the no flux boundary condition was assigned. Figure 7.9 shows the water table at two different times. Figure 7.10 shows the changes of water table at three different locations during the entire simulation.
Figure 7.5: Changes of error by increasing operator order
Figure 7.6: Simulating a channel with a dry zone in the middle, $C_z = 35$, $t = 86400 \ [s] = 24 \ [hr]$, $dx = dy = 50 \ [m]$. 
Figure 7.7: Simulating a channel with a dry zone in the middle, $C_z = 20$, $t = 7200 [s] = 2 [hr]$, $dx = dy = 10 [m]$. 

132
Figure 7.8: Monterey Bay Bathymetry.

Figure 7.9: Water table in the Monterey bay with flat bathymetry.

(a) Water Table at Time = 6 hr. (b) Water Table at Time = 18 hr
Figure 7.10: Changes in Water table at three points during 24 hours simulation. (Red) a point closer to norther boundary, (blue) a point closer to souther boundary, and (black) a point almost in the middle of the domain.
Chapter 8

Mimetic UCOAM

8.1 Introduction

In my proposal, I suggested to rewrite the UCOAM model developed in Chapter 2 using mimetic operators. Furthermore, to enhance its performance I suggested to use GPUs. That was mainly based on our preliminary results using Castillo–Grone’s Mimetic (CGM) operator in solving Poisson’s equation. Most of the past study on these operators are also solving only the Poisson’s equations. We did perform a stability analysis on these operators in Chapter 5. It was seen that the amplification factor is very close to one in almost all wavenumbers and different Courant numbers. Although this could be a nice thing but it could be also problematic. This means that if there is any error induced in the model, (mostly due to finite precision available for computers) that would stay there for a long time.

It is widely known that by increasing the dimension of the scheme, the order of the operators, and the order of the interpolation, the stability region is also reduced [130]. For example, Skamarock reported that the stability region of their RK3 scheme (see Chapter 5) is reduced by a factor of $\sqrt{2}$ in 2D space and by a factor of $\sqrt{3}$ for 3D space, based on the spatial discretization that was used. Since the goal is to use the higher order CGM operator (specifically the CGM divergence operator) along with
RK3 scheme to develop a UCOAM model, we need to extend the stability analysis to higher dimensions.

### 8.2 Stability Analysis in 2D domains

In order to test the stability of the numerical scheme a simple linear advection equation is chosen. This equation can be written as follows:

\[
\frac{\partial q}{\partial t} + \frac{\partial (uq)}{\partial x} + \frac{\partial (vq)}{\partial y} = 0,
\]

where \(u = \text{constant}\) is the advection velocity along x-axis; \(v = \text{constant}\) is the advection velocity along y-axis; and \(q\) is a scalar field defined at the center of each cell. A periodic boundary condition is used all over the domain. Two layers of ghost nodes are added to the domain to assist in implementing the boundary condition. A cubic interpolation is used to interpolate \(q\) from the center of the cells to the edges of the cell in order to calculate the fluxes. Fourth-order CGM divergence operator is used for spatial discretization; and RK3 is used for time discretization.

The scalar field, i.e. \(q\), is initialized with the same smooth square function that was defined in Chapter 5. There is no discontinuity in the initial function.

As the first test it was assumed that \(u = v = 0.5 \cdot C_r \cdot dx \cdot dt\), where \(C_r\) is the Courant number. Various Courant number, \(C_r\), were tested. A unit square domain was selected and it was discretized regularly in space with \(dx = dy = 0.02\); and \(dt = 0.02\). In all cases, as the time advanced, small oscillations appeared in the solution and they started to grow with time; until they completely over took the solution. Although a smoothly varying square signal was used for initializing \(q\); it was decided to make use of an even smoother function, such as a Gaussian function. Replacing the initial function with a smoothly varying Gaussian function did not
improved the situations and the same behavior was seen. Figure 8.1 shows the time
evolution of a Gaussian function advected through the domain. The same test was
repeated by lowering the resolution; yet no improvement was achieved.

Figure 8.1: Time evolution of a Gaussian function advected through the domain.

Setting \( u = v \) will result in advecting the signal in 45 degree direction. That is
known to be the direction that the scheme is the least stable. In order to test a more
stable direction it was assumed that \( v = 0 \) and \( u = C_r \cdot dx \cdot dt \). The same behavior
was observed, (see Figure 8.2).

Last test was repeated using Dirichlet boundary condition. Again, it was clear
that small oscillation arises and as they are moving along the x axis (note that \( u \neq 0 \)
and \( v = 0 \)), they start to grow. However, the domain is not big enough and before
these oscillations get too big they will reach the right boundary and exit the domain.
Meanwhile, fresh data (without any oscillations) enters the domain from the left
boundary. Therefore, the oscillations are kept under control. Note that, while periodic
boundary condition was used, the solution along with the oscillations that exit the
boundary on the right boundary; re-enters the domain from the left boundary; hence,
Figure 8.2: Time evolution of a Gaussian function advected through the domain. The oscillations have enough time to keep growing until they are completely out of control.

8.3 Strong Stability-Preserving (SSP) Time Discretization

At this point it was clear that RK3 and CGM divergence operators are not a good match in 2D domain. Since, the stability region would shrink even further in 3D space, it is no need to repeat the previous tests for 3D domains. In the search for another time discretization scheme with more stability, we came across a class of time discretization methods known as Strong Stability-Preserving (SSP) methods [51]. The SSP method provides high order time discretization scheme; and yet preserves the strong stability of the lower order methods. However, the explicit forward Euler
method must be stable, for this class of method to be applicable [51].

As a result, the first step is to analyze whether the fourth-order CGM divergence operator is stable with explicit forward Euler method. Once again, we use the von Neumann’s stability analysis. The fourth order Castillo–Grone’s Mimetic (CGM) divergence operator computes divergence at the center of the cell using the values given at the boundary of each cells. The CGM divergence operator does not require any dummy or ghost nodes at the boundaries. As a result, the method has a different stencil at the boundaries for 4th and 6th order operators. On the internal nodes; however, the CGM divergence operator is equivalent to the regular central finite difference. In 1D space, the forth-order CGM divergence can be written as:

\[
\left. \frac{\partial q}{\partial x} \right|_i = \frac{1}{\delta x} \left( \frac{1}{24} q_{i-3/2} - \frac{27}{24} q_{i-1/2} + \frac{27}{24} q_{i+1/2} - \frac{1}{24} q_{i+3/2} \right),
\]

where Figure 8.3 shows the node numbering in one dimension.

Here, we check if the scheme is stable using linear interpolation and forward Euler time stepping, in solving linear advection equation, which can be written as:

\[
\frac{\partial q}{\partial t} + \frac{\partial (u q)}{\partial x} = 0,
\]

where \(u = constant\) is the advection velocity. By discretizing the above equation, we get:
\[ \frac{q_i^{n+1} - q_i^n}{\delta t} = \frac{-1}{\delta x} \left[ \frac{1}{24} uq_{i-3/2}^n - \frac{27}{24} uq_{i-1/2}^n + \frac{27}{24} uq_{i+1/2}^n - \frac{1}{24} uq_{i+3/2}^n \right] \]

\[ q_i^{n+1} = q_i^n - \frac{u\delta t}{24\delta x} \left[ q_{i-3/2}^n - 27q_{i-1/2}^n + 27q_{i+1/2}^n - q_{i+3/2}^n \right]. \] (8.4)

Using linear interpolation, we have:

\[ q_i^{n+1} = q_i^n - \frac{u\delta t}{48\delta x} \left[ q_{i-2}^n - 26q_{i-1}^n + 26q_{i+1}^n - q_{i+2}^n \right]. \] (8.5)

By setting, \( \alpha = \frac{u\delta t}{48\delta x} = \frac{C_r}{48} \), where \( C_r \) is the Courant number, we get:

\[ q_i^{n+1} = q_i^n - \alpha \left[ q_{i-2}^n - 26q_{i-1}^n + 26q_{i+1}^n - q_{i+2}^n \right]. \] (8.6)

Now we use von Neumann analysis and substitute \( q_i^n = V^n e^{I\theta i} \), where \( I = \sqrt{-1} \).

As a result, we get:

\[ V^{n+1} e^{I\theta i} = V^n e^{I\theta i} - \alpha \left[ V^n e^{I\theta(i-2)} - 26V^n e^{I\theta(i-1)} + 26V^n e^{I\theta(i+1)} - V^n e^{I\theta(i+2)} \right]. \] (8.7)

By dividing both side with \( V^n e^{I\theta i} \), we get

\[ \frac{V^{n+1}}{V^n} = 1 - \alpha \left[ e^{-2I\theta} - 26e^{-I\theta} + 26e^{I\theta} - e^{2I\theta} \right], \] (8.8)

or:

\[ G = 1 - 2\alpha I \left[ 26 \sin(\theta) - \sin(2\theta) \right]. \] (8.9)

\( G = \frac{V^{n+1}}{V^n} \) is the amplification factor. For the scheme to be stable we must have \( |G| \leq 1 \). By examining the above equation (also see Figure 8.4) it becomes evident
that the amplification factor is only equal to one only for $\theta = 0$ and $\theta = \pi$. For any other wave-number and any Courant number the Amplification factor is larger than 1; hence, the scheme is not stable for Forward Euler time discretization.

![Figure 8.4: Amplification Factor.](image)

The forward Euler method must be stable in order to use strong stability-preserving (SSP) method. As a results, that framework can not be used for this scheme.

### 8.4 An Alternative Approach

It is clear that we need another time discretization scheme that proves to be stable with higher order CGM divergence operators in higher dimensions, before we can convert UCOAM using CGM operators. While that could be a lengthy task, (assuming such combination exists), we came up with an alternative approach.

In previous chapters it was shown that CGM operators work perfectly for solving Poisson’s equations. As explained in previous chapters, a Poisson’s equation is solved to obtain pressure field while using incompressible Navier-Stokes’ equations. This step is the most time consuming part of the model. Hence, it was decided to make use of the UCOAM model that was developed in Chapter 2 and convert the part of
the code that solves the Poisson’s equations so that it uses the CGM operators. In future efforts, the results of this alternative approach is investigated.
In this work, the main focus was on using higher order Castillo–Grone’s Mimetic (CGM) difference operators in solving partial differential equations in general curvilinear coordinates with the application in Geophysical fluid models. In Chapter 2 Unified Curvilinear Ocean Atmosphere Model (UCOAM) was introduced. UCOAM, uses central differences to discretize the Navier-Stokes’ (NS) equations that govern both of the atmospheric and oceanic flows. We used the incompressible form of the NS equations with Boussinesq’s approximations. Beside the boundary conditions, the main difference between the oceanic flows and the atmospheric flows in these case is the Buoyancy term. In oceans the Buoyancy term is linked to changes in pressure, salinity, and temperature; while in atmosphere it is related to temperature and water vapor. UCOAM is the fully curvilinear model, even in the vertical, and solves all three momentum equations. To our best of knowledge it is one of its first kind that make use of a curvilinear grid even in the vertical directions. Commonly, along the vertical directions the grid lines are kept straight. The common approach is to use sigma-coordinate or z-level in the vertical direction. This has some advantages, including allowing for larger time step. However, in very high resolution models and in presence of steep slopes, this could be problematic.

The flow in the oceans is widely assumed to be stratified. In Chapter 2, it was
shown that this assumption depends on the grid resolution. While in coarse models, this could be a very well established assumption, in very high-resolution and due to interaction of the flow with the bottom bathymetry or topography the density profile could differ from that of the stratified flows. It is also assumed that the vertical component of the velocity is much smaller than the other two components, due to the scales of the motions. It was shown in that chapter, although this assumption is again true in coarser models; however, in finer scale models there could be regions of the domain that the main transfer of the energy is performed by the vertical component of the velocity.

In all numerical models, a good portion of the computation is spent on interpolation or extrapolation. In Chapter 3 a numerical package was developed that will help the user with this task. It was shown that the most computationally cost effective part of this task is independent of the data; and in fact it is solely a function of how the points on source and destination grids are located relative to each other. Therefore, a procedure was developed that makes use of the polynomial interpolation (or extrapolation) in both 2D and 3D spaces. Using the developed package, the entire procedure of the interpolation (or extrapolation) is summarized to a sparse matrix multiplication, which the matrix could be pre-computed. It was also shown that as long as the points in the source and destination grids have not changed their locations relative to each other this sparse matrix will stay the same; thus, there is no need to recompute it. The package was designed in such a way that it could handle structured grids or unstructured grids as well as scattered data. This software package is available on MATLAB Central File Exchange repository and it was downloaded on average once a day during last year.

There are numerous studies on CGM operators, mostly on solving Poisson’s equations. Many of these papers are in 1D space with few exceptions in 2D domain. But
to my best of knowledge there was no study on the performance of these operators in solving the Poisson’s equations in 3D domain using CGM operators and the first study of using CGM operators in 3D domain is presented in this work.

In Chapter 4 the second-order CGM operators are used to discretize the Poisson’s equations in 2D and 3D domains. The system of equations arising from the discretization is solved using Jacobi iterations; and the results are compared to that of conservative central difference scheme. It was shown that in coarse grids the CGM operators were able to provide more accurate solutions. It was also shown that the system arising from using CGM operators required less number of iterations.

In Chapter 4 only the second-order CGM operators were used. Using the fourth and sixth-order CGM operators will result into a system of equations which is not diagonally dominant; therefore, the Jacobi and Gauss-Seidel iterative method could not be used. In Chapter 6 a MATLAB software package was developed that provides the user with the Laplacian operators in 2D or 3D curvilinear grids. The boundary conditions were included in this operator. This Laplacian operator can be constructed using second, fourth, or sixth-order CGM operators; and it can be used in solving Poisson’s equation in 2D or 3D fully curvilinear grids. The Laplacian operators would be stored as sparse matrix to save memory. To my best of knowledge, this is the first package developed for CGM operators in fully curvilinear grids in 2D and 3D domains.

Usually in curvilinear grids, one needs to transform all the equations from the physical domain into the computational domain. This requires computing various grid transformation metrics and performing multiple interpolations. However, the uniqueness of this approach is that it hides all those complexities from the user. All the user needs to provide is a nodal grid and defining what boundary conditions they require. Then they have to choose what order of operators or interpolation they would desire and everything is calculated automatically. The calculated Laplacian operators,
includes the necessary interpolations, grid transformations matrix and anything else needed to solve the Poisson’s equation. As a result, the user does not need to worry about complexity of the transformation or interpolation.

In Chapter 6 it was discussed how beneficial it would be if the user sticks to orthogonal meshes in curvilinear grids. It was shown that once the grid is orthogonal, there is no interpolation required; hence, no inaccuracies would enter the solution due to interpolation. Furthermore, it was shown that using a curve fitting approach, instead of an interpolation scheme, is beneficial as the solution became independent of the order of the polynomial that was fit to the data. Using curve fitting approach, however, requires least-square solution to determine the free parameters. But in Chapter 3 it was shown that the most costly part of this procedure is independent of the data and again it depends on the location of the points. Hence, as long as the grid is not going to change the entire procedure could be precalculated as a sparse matrix. Again all these steps are included in the calculated Laplacian operators and the user does not need to worry about any of this. To my best of knowledge, this is the most complete (and yet easy to use) package that has been developed as of this day that makes use of CGM operators in 2D and 3D for elliptic and Poisson’s equations.

In Chapter 5 the performance of the CGM operators in solve advective equation was studied. To my best of knowledge the first stability analysis of these operators is provided in this work. While the CGM operators show very good stability in 1D space, in Chapter 8 it was shown that in 2D domains the higher order CGM operators become unstable. The search for an stable combination of high-order time discretization and high-order CGM operators leaded us to a class of time discretization called strong stability-preserving (SSP). However, in order to use such frame work, one must first show that the spatial scheme is stable using first-order forward Euler
scheme. The von Neumann analysis on fourth-order CGM divergence operators shows that the scheme under this condition would be unconditionally unstable; hence, the SSP framework could not be used.

At this moment the search for an appropriate time discretization scheme for high-order CGM operators has not concluded yet. Meanwhile, it is suggested to use the current UCOAM model and only convert the part of the model that solves a Poisson’s equations for pressure to use the CGM operators.

It would take approximately the same amount of time to construct a Laplacian using different order of the operators. However, the sparsity of the resulting matrices varies considerably. The matrices arising from using second-order operators are the sparsest and the sixth-order operators result into the densest matrices. Note that even the density of the densest matrix is less than half a percent for a relatively small domain and the density goes down very fast as the domain increases. Although it takes approximately the same amount of time to construct these matrices regardless of the order of the operators; the time needed to solve the resulting system varies. The system of equations resulted from the sixth-order operators requires the most time and those resulted from the second-order requires the least. In terms of accuracy, as the order of the operators increases the error decreases. While the accuracy improves considerably between fourth-order and second order operators, generally, the sixth-order and fourth-order did not vary that much in terms of accuracy. Taking all these factors into account, it is suggested to use the fourth-order operators to solve Poisson’s equations.

CGM operators are designed in such a way that they do not require any ghost or dummy nodes. While in the interior nodes, far from the boundaries, the CGM operators tend to have exactly the same stencil and weights as the regular central finite differences; close to the boundaries, the scheme switches to a sided scheme in
order to not use any of the ghost or dummy nodes. While this could have many advantages, including that the operators keep the same order of accuracy throughout the entire domain, it has some disadvantages too. One great disadvantage is that there is no mechanism involved to implement periodic boundary condition when solving Poisson’s equations. I have developed some alternative approaches to trick the boundary conditions into being periodic using CGM operators; but it involves further interpolation. These results is not reported here and I am planning to release them as a separate publication.

Clearly, further works is needed to use the CGM operators to solve equations other than elliptic equations. Particularly, the search for an appropriate time discretization is not completed and it is suggested to have further research on that topic.
Bibliography


