In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature: _____

Date: _____

Scalable Iterative GMRES-FFT Method for Subsurface Scattering Problems

by

Yun Teck Lee

Dissertation

submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy in the Department of Mathematics and Statistics

Idaho State University

Spring 2023

To the Graduate Faculty:

The members of the committee appointed to examine the dissertation of Yun Teck Lee find it satisfactory and recommend that it be accepted.

Dr. Yury Gryazin,
Major Adviser

Dr. Bennett Palmer,
Committee Member

Dr. Wenxiang Zhu,
Committee Member

Dr. Ken Bosworth,
Committee Member

Dr. Marco Schoen,
Graduate Faculty Representative

Acknowledgment

I would like to thank my major advisor, Dr. Yury Gryazin, for his advice, support, and more importantly, patience throughout my time as a graduate student.

I would like to acknowledge my defense committee, Dr. Bennett Palmer, Dr. Wenxiang Zhu, Dr. Ken Bosworth, and Dr. Marco Schoen, for being so flexible in their schedules and for going through the grueling process of reading, commenting, and correcting this manuscript. I appreciated their time, insight, and expertise.

I would like to acknowledge a fellow doctoral student, Ronald Gonzales, for your collaboration and technology advice throughout our research project.

I want to thank all the staff in the Mathematics and Statistics department at ISU for the years of continuous support, education, and an overall excellent resource.

Table of Contents

List of Figures

List of Tables

List of Acronyms

2D	Two-dimensional
3D	Three-dimensional
API	Application Programming Interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
DST	Discrete Sine Transform
EMI	Electromagnetic Induction
FFT	Fast Fourier Transform
FFTW	Fastest Fourier Transform in the West
GHz	Gigahertz
GMRES	General Minimum Residual
GPR	Ground Penetrating Radar
GPU	Graphics Processing Unit
HPC	High Performance Computing
IB	InfiniBand
IC	integrated Circuit
INL	Idaho National Lab
LBNL	Lawrence Berkeley National Lab
MATLAB	Matrix Laborator
MIT	Massachusetts Institute of Technology

MPI	Message Passing Interface
OpenMP	Open Multi-Processing
PETSc	Portable, Extensible Toolkit for Scientific Computation
PDE	Partial Differential Equation
PML	Perfectly Matched Layer
RAM	Random Access Memory
Re	Reynolds Number
SI	Système International

List of Symbols

$\mathbb{C}$	Set of complex numbers
$\mathbb{N}$	Set of natural numbers, the nonnegative integers
$\mathbb{R}$	Set of real numbers
$\mathbb{Z}$	Set of integers
$\mathbb{P}_n$	Set of all polynomials of degree n or smaller
$\mathcal{K}_n$	n-th order Krylov subspace
Δ	Laplace operator
δ_x	First order central difference approximation with respect to x
δ_x^2	Second order central difference approximation with respect to x
$\frac{\partial}{\partial x}$	Partial derivatives with respect to x
Δ_h	Sum of second order central difference approximation with respect to x, y and z
A^T	Transpose of the matrix A
A_p	Preconditioner matrix for A
Ω	Rectangular computational domain
$\partial\Omega$	Boundary of the computational domain
$Re(x)$	Real part of x
$Im(x)$	Imaginary part of x

Scalable Iterative GMRES-FFT Method for Subsurface Scattering Problems

Dissertation Abstract - - Idaho State University (2023)

The objective of this dissertation is to present an efficient parallel implementation of the iterative compact high-order approximation numerical solver for the forward problem of the subsurface scattering problems derived from the 3D Helmholtz equation. The high-order parallel iterative algorithm is built upon a combination of the generalized minimum residual method (GMRES) method with a direct Fast Fourier transform (FFT) type preconditioner from the authors' previous work in [37]. High-order compact finite differences schemes are used to compute high-resolution numerical solutions. The performance of the proposed algorithm will be tested by computationally simulating data with realistic ranges of parameters in soil and mine-like targets. Additionally, the application of the proposed numerical solver can be extended to computer numerical solutions for other partial differential equations (PDE) such as 3D convection-diffusion equations. The proposed algorithm represents a highly parallelizable iterative algorithm suitable for excellent performance under various parallel environments.

Key Words: Helmholtz equation, convection-diffusion equation, subsurface imaging, GPR, high-order compact finite difference schemes, GMRES, FFT, fast scalable preconditioners, parallel algorithms, OpenMP, MPI, PETSc

1 Introduction

1.1 Overview

Landmine detection has always been a challenging but essential endeavor for the protection of civilian and military lives. While they are fairly easy to plant, detecting them can be relatively difficult. This is in part due to the advent of plastic explosives that cannot be discovered using energy-based detection methods, such as the kind used in metal detectors[7]. In [7], the author remarks that present-day methods mostly rely on signal analysis and image processing techniques to identify unique shapes and signals that are representative of landmine signatures beneath the earth. These techniques have also been employed in both vehicle-based and hand-held sensing devices.

Nowadays many landmine detectors employ numerous sensing methods simultaneously. Some of the most frequently used include Electromagnetic Induction (EMI), and more recently, Ground Penetrating Radar (GPR). Unlike the former, the main advantage of GPR lies in its ability to detect both metallic and non-metallic targets which EMI is incapable of sensing. Furthermore, GPR also has the advantage that its signals can be reconstructed into an image. This image can then be analyzed using image processing and machine learning to recognize visual patterns within the radar [7].

To understand the physical behavior of a system, numerical modeling is a practical and solid approach. For the case of landmine detection with GPR, numerical modeling serves as a practical tool for designing and optimizing antennas in synthetic but realistic conditions

[13]. The problem of numerical simulation of the subsurface imaging usually accounts to the solution of PDEs, which can be discretized by applying finite difference, finite elements, or finite volume methods. The resulting linear system however is generally large and sparse. The system can be solved via a direct or iterative method. However, this process is time-consuming due to a large amount of computation and in general, it represents a majority of the total simulation time.

Direct methods such as Gaussian elimination with pivoting are seen as the most accurate and robust solvers for general linear systems as they obtain the exact solution for the linear systems. However, it is unsuitable for large systems because of the memory requirements and the lack of parallelism. Especially in the case of sparse matrices, Gaussian elimination would produce fill-in which may destroy the sparsity, resulting in higher storage cost. On the other hand, iterative methods require low memory costs. Typically they would only store the matrix, right-hand side vector and a few additional vectors to approximate the solution of a linear system. Thus, the iterative methods are favored in the case of large sparse linear systems, especially arising from the discretization of three-dimensional PDEs. However, they only give approximate solutions, and their efficiency depends on the matrix structure. The usage of appropriate preconditioners improves the convergence rates of the iterative methods.

The GMRES method proposed by Saad and Schultz in 1986 [32] is one of the most popular iterative algorithms for solving a system of linear equations. It is an iterative solver that finds approximations within a Krylov subspace of the system. The development of an efficient iterative solver is crucial, as large general matrices arise in a variety of applications. One way of doing this is to parallelize the numerical method.

Moore's Law states that the number of transistors in a dense integrated circuit (IC) would double about every two years. Abiding by the law, modern computers have evolved from machines with a single processing component to complex architectures combining the likes of central processing units (CPU) with multiple cores, graphics processing units (GPU), and digital signal processors (DSP). Most modern computers have multiple processing cores that enable multiple programs to run simultaneously or allow one program to use multiple cores. As computational modeling and its resources become commonplace, numerical solutions for PDE can be solved with increasing resolution and accuracy. Concurrently, the technology development also enables more variables and processes to be taken into account, and spatial and temporal resolutions are increased to model real field-scale events models become more complex yet resource efficiency remains an important requirement. Multiscale and multiphysics modeling encompasses these factors and relies on High-Performance Computing (HPC) resources and services to solve problems effectively [24]. Within this context, complexity, and scalability both play an important role. It is much preferable to have an algorithm with low complexity and easy to implement in large computing clusters. Additionally, it needs to be flexible enough to capitalize highly optimized low-level libraries and support new hardware architectures.

1.2 Objectives

Coherent to the objective of the Ph.D. program in Engineering and Applied Science, I would like to present a scalable iterative solver for computing the numerical solution of PDE for this dissertation. The proposed numerical algorithm is based on a GMRES-FFT-type algorithm

which can be implemented with readily available libraries. The proposed numerical algorithm would be implemented in a large computer cluster to demonstrate its scalability as well.

The main target application of the proposed numerical solver is to find the solution for the forward electromagnetic scattering problems arising in the area of detection, identification, and imaging of subsurface objects. The subsurface scattering problem considered here can be formulated in the form of a 3D Helmholtz equation

$$\Delta u(x,y,z) + k^2(x,y,z)u(x,y,z) = f(x,y,z), \quad \text{in } \Omega,$$

with either the Dirichlet, Neumann or Sommerfeld-like boundary conditions

$$\Gamma u = g, \quad \text{on } \partial\Omega,$$

where Ω is a three-dimensional rectangular domain, $k^2(x,y,z)$ is a complex-valued variable function, $\partial\Omega$ is the boundary of Ω, and Γ is a differential operator corresponding to the Dirichlet, Neumann or Sommerfeld-like boundary conditions.

The development of forward and inverse methods of subsurface scattering problems continues to impose difficult mathematical, computational, statistical, and signal-processing challenges. One of the main challenges is to find the numerical solution to the forward scattering problem with large values of the angular frequencies, i.e., for small wavelengths. This poses a challenge to the existing numerical solvers as a large number of grid points would be required for the numerical solver to obtain optimal results. Additionally, with the technology advancement, computational efficiency i.e., computing the approximate solution in a fast and scalable manner, is now of concern for many numerical algorithms. The proposed numerical algorithm in this dissertation will tackle this issue by introducing a new state-of-the-art highly parallelizable approach that can fully capitalize on the recent development of

multi-core technologies.

In this dissertation, the development of an efficient iterative method for the solution of the forward scattering problem will be presented. In the hope to serve as a necessary and preliminary step to the solution of the inverse problem, the proposed iterative method would represent a fast and accurate numerical method for the solution of the forward problem.

In [37], the author and co-researchers developed a parallel FFT-type direct method that uses high-order compact approximation schemes. The proposed iterative method would utilize this highly parallelized direct solver as a preconditioner for efficient implementation. The scalability of the direct solvers would significantly increase the efficiency of high-resolution iterative methods and enable computation on large grid sizes. Overall the high resolution of the iterative method is achieved by two contributing factors. Firstly, the application of a higher order scheme and the other contributing factor is the ability to use finer computational grids due to the increased computational power of computer clusters. The sequential numerical method was first developed in 2014 [34]. Efforts were concentrated on making the proposed algorithm faster and more efficient by the means of parallelization.

The proposed iterative method will be implemented using some of the most common protocols such as Open Multi-Processing (OpenMP), Message Passing Interface (MPI), and Hybrid MPI/OpenMP within the C programming language. A protocol such as MPI allows communication between multiple nodes. In this dissertation, the way the data and communication are handled is discussed in detail, as well as how the proposed iterative solver was developed sequentially and it is implemented in a parallel algorithm.

1.3 Outline

Chapter 2 will introduce the PDE considered in this dissertation where the iterative method could be used for computing the numerical solution. An overview of these PDEs and their applications that motivated the development of the iterative method will be presented.

Chapter 3 shows the discretization of the PDE discussed in Chapter 2 based on a compact approximation scheme derived from Taylor series expansion. The details of the discretization for the second, fourth and sixth order scheme will be provided. This chapter will also include how the Sommerfield radiation boundary condition is implemented for the subsurface inclusion model problem.

Chapter 4 will presents the direct FFT solver as a direct method for solving PDE in Chapter 2 under certain restrictions. The direct FFT solver would serve as direct solver as the preconditioner solver used in the proposed iterative algorithm. In-depth detail of the direct solver would be presented.

Chapter 5 presents the outlines for the proposed iterative method. A general literature review of the iterative method will be provided. The structure of the iterative method is shown and its implementation via the Portable, Extensible Toolkit for Scientific Computation (PETSc) is discussed in this chapter.

Chapter 6 focused on the parallel implementation of the proposed iterative method with parallel techniques such as OpenMP and MPI. The algorithm and details of these implementations will be given.

Chapter 7 would showcase the performance of the direct FFT solver and proposed iterative

solver by presenting the numerical results from various test problems based on PDE introduced in Chapter 2. These include the accuracy, computational time, and scalability of the aforementioned solvers. The result from the numerical experiments would be presented and analyzed.

2 Model Equation

2.1 Helmholtz Equation

In this dissertation, the main targeted application of the proposed iterative method is to find the numerical solution for the forward problem of the subsurface scattering problem, specifically as a landmine detector. Electromagnetic subsurface imaging can be formulated as an optimization problem constrained by a set of PDE, specifically Maxwell's equations, which govern electromagnetic wave propagation. The Helmholtz equation which can be derived from Maxwell's equations can be used as an approximation of the wave equation. As such the primary focus of the iterative method is to develop a fast and efficient numerical solver for the three-dimensional Helmholtz equation. Let $\mathbf{x} = (x, y, z)$ be the coordinates in the three-dimensional euclidean space $\mathbb{R}^3$, the 3D Helmholtz equation is given by,

$$\Delta u(\mathbf{x}) + k^2(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}), \qquad \text{in } \Omega \tag{2.1}$$

where $\Omega = \left\{\mathbf{x} \mid L_x^l \leq x \leq L_x^r; L_y^l \leq y \leq L_y^r; L_z^l \leq z \leq L_z^r\right\}$, $L_x^l < L_x^r, L_y^l < L_y^r, L_z^l < L_z^r$ and $k^2(\mathbf{x})$ is called the coefficient function. The Dirichlet boundary condition is considered, that is

$$u = v, \qquad \text{on } \partial\Omega \tag{2.2}$$

where v is a known function defined on the boundary $\partial\Omega$. The stability, existence, and uniqueness analysis of this problem can be found, for example, in [3, 14]. The boundary condition considered in this paper algorithms can be extended to the Neumann boundary

conditions on the sides of the computational domain (see [8, 9]) or to the first and second-order absorbing boundary conditions [17].

Highly efficient numerical algorithms for solving the Helmholtz equation remain an area of active research. Although this dissertation will be focusing on its application to subsurface scattering problems, an efficient Helmholtz solver is of great interest for any wave-based inverse problem, given that majority rely on the ability to solve the forward problem efficiently. Helmholtz equation finds its applications in many physics problem-solving concepts such as seismology, acoustics, and electromagnetic radiation. Moreover let $k^2(\mathbf{x}) = 0$ then the 3D Helmholtz equation becomes the 3D Poisson equation

$$\Delta u(\mathbf{x}) = f(\mathbf{x}), \qquad \text{in } \Omega \tag{2.3}$$

which has broad utility in theoretical physics. In this dissertation, the Poisson equation will be treated as a special case of the Helmholtz equation where $k^2(\mathbf{x}) = 0$.

The coefficient function $k^2(\mathbf{x})$ is called the wave number when applying the equation to waves. Solving the Helmholtz equation at high wave numbers is challenging because the solutions are highly oscillatory. Even though the equation is linear, the resulting discrete linear system is indefinite and ill-conditioned. Another difficulty in simulating the numerical solutions at high wave number problems is the "pollution effect" in almost all computational schemes. A higher-order compact scheme can reduce the "pollution effect"; alternatively decreasing the step size h is effective.

In the past two decades, high-order compact difference methods have generated renewed interest and a variety of specialized techniques have been developed. Previous research has developed compact fourth-order accurate finite difference approximation for the Helmholtz

equation in two or three dimensions (see e.g. [6, 15]). For 3D Helmholtz equations, Turkel et al. [5] developed the sixth-order scheme for the Helmholtz equation based on equation-based differencing, where derivatives of the Helmholtz equation are used to eliminate higher-order derivatives in the discretization error. In 2014, Gryazin [34] developed the numerical iterative Helmholtz solver based on these high-order schemes was developed. In this dissertation, we aimed to improve the iterative Helmholtz solver by the means of parallelization.

The forward problem for the subsurface scattering problem is defined as follows: given the physical properties of the medium, in this case, the coefficient function $k^2(\mathbf{x})$ and the location of the source, compute the wave-field u at the surface. Subsection 2.1.1 will present a mathematical model to simulate this application.

2.1.1 Subsurface inclusion model

In this subsection, a simplified model of the GPR signal propagation will be presented. The landmine-like target will be modeled as small abnormalities embedded in an otherwise uniform media with an air-ground interface. These abnormalities are characterized by the electrical permittivity ε and the conductivity σ, whose values differ from those of the host media. On top of that, realistic ranges of parameters will be considered.

Define $\mathbf{x} = (x, y, z)$ as before. Let the electrical field E_0 originated by a GPR be a linearly polarized plane wave with the direction of propagation parallel to the positive direction of the z-axis, $E_0 = (0, e^{i\omega z}, 0)$, where ω is the angular frequency of the signal, $\mu_0 = 4\pi \cdot 10^{-7}$ *Henry/m* is the magnetic permeability of free space and $\varepsilon_0 = 8.854 \cdot 10^{-12}$ *Farad/m* is the dielectric permittivity of free space. It is assumed that the mine-like targets are located

in the ground. Then the following 3D Helmholtz equation for the function $v(\mathbf{x}, \omega)$ can be derived from Maxwell's system

$$\Delta v + k^2(\mathbf{x}, \omega)v = 0. \tag{2.4}$$

Here the coefficient function $k^2(\mathbf{x}, \omega)$ has the form

$$k^2(\mathbf{x}, \omega) = \omega^2 \mu_0 \varepsilon(\mathbf{x})(1 + i\frac{\sigma(\mathbf{x}, \omega)}{\omega\varepsilon(\mathbf{x})}) \tag{2.5}$$

where $\varepsilon(\mathbf{x})$ and $\sigma(\mathbf{x}, \omega)$ represent the electrical permittivity and the electrical conductivity of the medium. It is assumed that $\varepsilon = \varepsilon_0$ in the air and $\varepsilon = \varepsilon_0\varepsilon_r$, where ε_r is the relative dielectric constant. In the air $\varepsilon_r \equiv 1$ and $\sigma(\mathbf{x}, \omega) \equiv 0$. Consider the "loss tangent"

$$\tan\delta = \frac{\sigma(\mathbf{x}, \omega)}{\omega\varepsilon(\mathbf{x})} \tag{2.6}$$

Substitute everything into Equation (2.5) the function $k^2(\mathbf{x}, \omega)$ now has the form

$$k^2(\mathbf{x}) = \omega^2 \mu_0 \varepsilon_0 \varepsilon_r (1 + i\tan\delta) \tag{2.7}$$

The loss tangent is assumed to be independent of ω, i.e., $\frac{\partial}{\partial\omega}[\tan\delta] = 0$. It is satisfied with sufficient accuracy in many practical scenarios of land mine detection. Next, the typical parameter ranges for the coefficient k^2 are presented. All units below are given in the SI system. The frequency of the signal $f = \omega/2\pi$ is between 0.5 GHz and 3GHz. When the frequency $f = 1$ GHz, the approximate values of the parameters $\varepsilon_r, \tan\delta$, and $k^2(\mathbf{x})$ and the wavelength $\lambda = 2\pi/Re(k)$ are given in Table 2.1.

Table 2.1: Approximate values of $\varepsilon_r, \tan\delta, k^2(\mathbf{x})$ and λ for different mediums at $f = 1$ GHz

Medium	ε_r	$\tan\delta$	$k^2(\mathbf{x})[\frac{1}{\mathrm{m}^2}]$	λ [cm]
Air	1	0	439.2	30
Ground/Soil	2.9	0.025	$1273 + 31 \cdot i$	17
TNT	2.86	0.0018	$1256 + 2.26 \cdot i$	17.7

To calculate the forward problem accurately, one should use at least ten grid points per wavelength. Suppose, for example, that one wants to calculate the function v in a square region of 2m $\times$ 2m $\times$ 2m one should at least use $117 \times 117 \times 117$ grid for $\lambda = 17$ cm. This is a great motivation for the development of a new efficient parallel algorithm for the solution of the Helmholtz equation with complex-valued coefficients.

Statement of the forward problem

The electrical parameters ε and σ are assumed to have constant background values everywhere in the ground except in the mine-like targets, whose sizes will be small in comparison to the size of the computational domain Ω. Let $k_0^2(z)$ be the function $k^2(\mathbf{x})$ without any inclusions, that is,

$$k_0^2(z) = \begin{cases} 439.2, & \text{in air,} \\ 1273 + 31i, , & \text{in ground.} \end{cases}$$

Hence $k_0^2(z)$ has constant values both in air and ground with a discontinuity at the air-ground interface and is identical to the function $k^2(\mathbf{x})$ in the computational domain except within the area where inclusions are present (if any). Furthermore, let $u_0 = u_0(\mathbf{x})$ be the solution of homogeneous Helmholtz Equation (2.1) which corresponds to the initializing plane wave with no inclusions. Then u_0 consists of the initial, reflected, and transmitted plane waves,

$$u_0 = \begin{cases} e^{ik_0 z} + Ae^{-ik_0 z}, & \text{if } 0 < z < 0.5, \\ Be^{ik_0 z}, & \text{if } 0.5 \leq z < 1, \end{cases}$$

where A and B are the reflection and transmission coefficients given by:

$$A = \frac{k_0^- - k_0^+}{k_0^- + k_0^+}, \qquad B = \frac{2k_0^-}{k_0^- + k_0^+}.$$

Here k_0^- and k_0^+ are the values of k_0 for air and ground respectively. The presence of these coefficients ensures the continuity of the function u_0 together with its first derivatives at the air-ground interface.

Seeking a solution to Equation (2.1) in the form $u(\mathbf{x}) = u_0(\mathbf{x}) + v(\mathbf{x})$, where the function v represents the wave scattered by the mine-like targets. Then u satisfies Equation (2.1) with the right hand side

$$f = f(\mathbf{x}) = \begin{cases} 0, & \text{outside inclusions,} \\ -(k^2(\mathbf{x}) - k_0^2(z))u_0(\mathbf{x}), & \text{inside inclusions.} \end{cases}$$

Boundary condition for subsurface scattering problem

For the subsurface inclusion model, the actual problem is solved in the infinite domain $\mathbb{R}^3$, and the numerical solution must satisfy the so-called Sommerfeld radiation condition, which in three-dimensional has the form

$$\lim_{r\to\infty} r\left(\frac{\partial}{\partial r} - ik\right) u = 0 \tag{2.8}$$

where $r = \sqrt{x^2 + y^2 + z^2}$ and i is the imaginary unit. Mathematically, this condition is required to ensure the uniqueness of the solution (and hence the well-posedness of the problem). In a physical context, the condition ensures that the scattering of an incoming wave only produces outgoing not incoming waves from infinity. If one performed the computation in a finite computation domain such as Ω, spurious wave reflections are likely to be generated at the artificial boundary $\partial\Omega$ of the computational domain.

The perfectly matched layer (PML) method truncates the euclidean space $\mathbb{R}^3$ into the finite computational domain Ω by surrounding Ω with a layer of "absorbing" material. In

theory, the outgoing waves are absorbed without creating any artificial reflected waves at the interface between the PML layer and the computational domain. The PML method is implemented by modifying the coefficient function $k^2(\mathbf{x}, \omega)$ in (2.5) as follows

$$k^2(\mathbf{x}, \omega) = k^2(\mathbf{x}, \omega) + i\sigma_x(x) + i\sigma_y(y) + i\sigma_z(z) \tag{2.9}$$

where i is the imaginary unit and

$$\sigma_x(x) = \begin{cases} 2\pi a_0 \sigma_0 \left(\frac{L_x^r + \delta - x}{\delta} \right), & x < L_x^l + \delta \\ 0, & L_x^l + \delta < x < L_x^r - \delta \\ 2\pi a_0 \sigma_0 \left(\frac{x - L_x^l + \delta}{\delta} \right), & L_x^r - \delta < x \end{cases}$$

with the dominant frequency σ_0 of the source, the thickness δ of the PML layer, and a constant a_0. The functions $\sigma_y(y)$ and $\sigma_z(z)$ are defined in similar way. Now Equations (2.1) would represent a Helmholtz equation with PML boundary conditions. In this dissertation, $2\pi a_0 \sigma_0 = 7$ is considered.

2.2 Convection-Diffusion Equation

To demonstrate the robustness of the numerical method, the numerical solution of the steady-state 3D convection-diffusion equation is also considered. Define $\mathbf{x} = (x, y, z)$ to be the coordinates in $\mathbb{R}^3$ as before, the 3D convection-diffusion equation is given by,

$$\Delta u(\mathbf{x}) + \alpha \frac{\partial}{\partial x} u(\mathbf{x}) + \beta \frac{\partial}{\partial y} u(\mathbf{x}) + \gamma \frac{\partial}{\partial z} u(\mathbf{x}) = f(\mathbf{x}), \qquad \text{in } \Omega \tag{2.10}$$

where Ω is the same rectangular grid defined in Section 2.1 and α, β and γ are constants known as convection coefficients in the x-, y- and z-directions respectively and f is a forcing function. For simplicity, f is assumed to be twice continuously differentiable. The Dirichlet

boundary condition with $u = v$ is imposed on the boundary of the computational domain where v is a known function defined on the boundary, $\partial\Omega$.

The convection-diffusion equation is very important in computational fluid dynamics to model the transport phenomena, including heat transfer and fluid flows [28, 29]. The unknown function u may represent the concentration of a pollutant being transported (or "convected") along a stream moving at velocity (α, β, γ) that is also subjected to diffusion effects. Alternatively, it may represent the temperature of a fluid moving along a heated wall, or the concentration of electrons in models of semiconductor device [10].

The values for the convection coefficients do not have to be constant. In this dissertation, a variant of the 3D convection-diffusion equation is considered in the form

$$\Delta u(\mathbf{x}) + \gamma(\mathbf{x})\frac{\partial}{\partial z}u(\mathbf{x}) = f(\mathbf{x}), \qquad \text{in } \Omega \tag{2.11}$$

In this variation, γ is now a function of $\mathbf{x}$ while the value of α and β are small enough to be assumed as zero. The proposed numerical solver would compute the numerical solution of Equation (2.11) with the same computational domain and boundary condition as Equation (2.10).

The magnitude of the convection coefficient is also usually referred to as the Reynolds number in much literature. Typically, diffusion is a less significant physical effect than convection. For large values of the convection coefficients, Equation (2.10) is said to be convection-dominated and is generally difficult to solve numerically [18, 20].

For convection-dominated problems, basic iterative methods fail to converge for solving the linear systems arising from the second-order central difference scheme, or it may produce nonphysical oscillations for large Reynolds numbers [23]. On the other hand, the upwind

difference scheme is usually stable but reduces the order of accuracy to the first order. In either case, the traditional numerical method has low accuracy and thus needs fine discretization to obtain the desired accuracy. These pose many computational challenges due to the prohibitive computer memory and CPU time requirements, especially for solving three-dimensional problems. Similar to the Helmholtz equation, using a higher-order finite difference approximations scheme would address these challenges. In this dissertation, the proposed iterative method will implement the numerical algorithm based on a fourth-order approximation scheme to find the numerical solutions for 3D convection-diffusion equation in the form of Equation (2.10) and (2.11).

3 Compact Scheme Discretizations

3.1 Introduction

In the field of applied mathematics, discretization is used for approximating the differential term in PDE with a prescribed accuracy. In this chapter, the model problem introduced in Chapter 2 will be discretized based on a high-order compact finite difference scheme. The finite difference scheme would convert the PDE into a system of linear equations that can be solved with matrix algebra techniques. This is the crucial first step for implementing a numerical solver for PDE problems on digital computers.

In numerical analysis, a stencil is a geometric arrangement of a nodal group that relates to the point of interest by using a numerical approximation routine. The stencil algorithm serves as a basis for a variety of algorithms to numerically solve PDE problems and is commonly found in parallel applications. The stencil algorithm operates on nodes that divide a computational domain. These nodes would hold one or multiple values, and they each have neighboring nodes. In the case of a compact stencil, the stencil would only use the center node and all its adjacent nodes. In the two-dimensional case, the stencil would be presented as a 9-point stencil and a 27-point stencil in the three-dimensional case. They are non-overlapping and of equal size to improve load balancing. The design of the stencil algorithm makes it easy for a parallel implementation of optimal running problems with large computational domains on multiple nodes.

The process of developing the second, fourth and sixth-order compact finite difference schemes

for the discretization of the Helmholtz equation will be presented. Moreover, the coefficient matrix of the linear system arising from the discretization would be presented in stencil notation.

This chapter will be focusing on the 3D Helmholtz equation (2.1) define in the computational domain

$$\Omega_h = \{(x,y,z) \in \mathbb{R}^3 \mid x_i = L_x^l + ih_x, y_j = L_y^l + jh_y, z_l = L_z^l + lh_z, \\ i = 1,...,N_x, j = 1,...,N_y, l = 1,...,N_z\} \tag{3.1}$$

where $h_x = \left(L_x^r - L_x^l\right)/(N_x+1), h_y = \left(L_y^r - L_y^l\right)/(N_y+1)$ and $h_z = \left(L_z^r - L_z^l\right)/(N_z+1)$ are grid steps in x-, y- and z-directions respectively. The corresponding result for the two-dimensional case will be shown without much emphasis.

The last section of this chapter will showcase the fourth-order discretization for the 3D convection-diffusion equation (2.10) and (2.11) as an extended application for the iterative solver to solve a wider class of PDE problems.

3.1.1 Taylor series expansion

Compact schemes are derived from Taylor series expansion. With the Taylor series, the derivatives of a function at a single point can be expressed as a linear combination of the values of the function around and at the same point. This enables the differential problems to be turned into a system of linear equations. The scheme is called compact because it only involves the 26 neighboring grid points (8 in the case of 2D) nearest to the reference grid point in a unit cube as seen in Figure 3.1.

In general the Taylor series for the function $u(x,y,z)$ at the point (x_i, y_j, z_l) can be presented

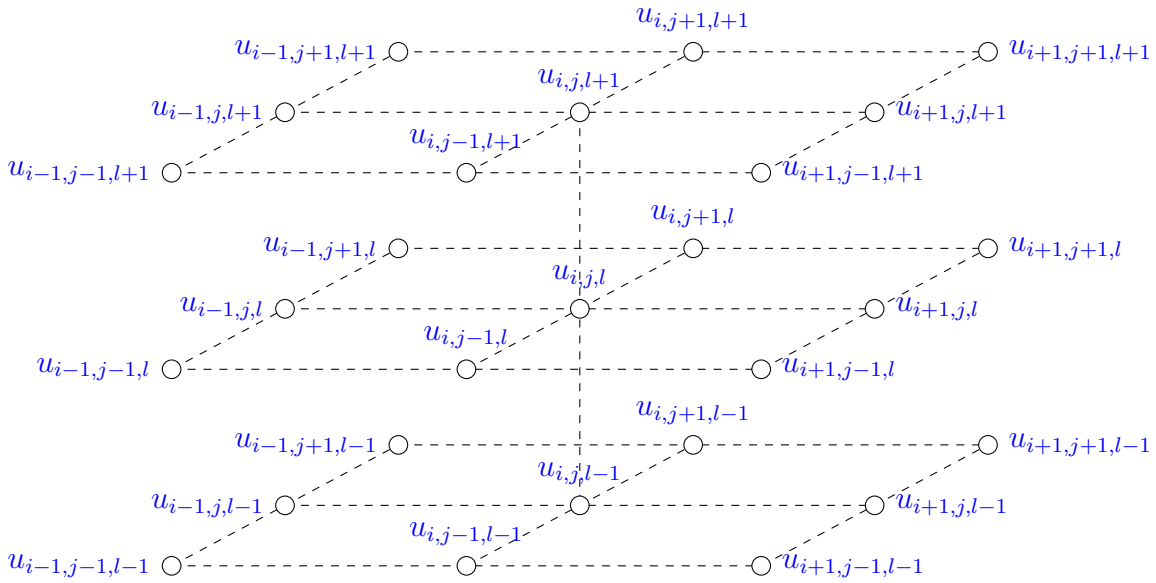

Figure 3.1: 27-point stencil operator for three-dimensional PDE

as

$$u_{i\pm1,j,l} = u_{i,j,l} \pm h_x \frac{\partial u_{i,j,l}}{\partial x} + \frac{h_x^2}{2}\frac{\partial^2 u_{i,j,l}}{\partial x^2} \pm \frac{h_x^3}{3!}\frac{\partial^3 u_{i,j,l}}{\partial x^3} + \frac{h_x^4}{4!}\frac{\partial^4 u_{i,j,l}}{\partial x^4} \pm \frac{h_x^5}{5!}\frac{\partial^5 u_{i,j,l}}{\partial x^5} + \frac{h_x^6}{6!}\frac{\partial^6 u_{i,j,l}}{\partial x^6} \pm \cdots \tag{3.2}$$

where $u_{i,j,l} = u(x_i, y_j, z_l)$. The difference between the two equations in (3.2) would yield

$$u_{i+1,j,l} - u_{i-1,j,l} = 2h_x \frac{\partial u_{i,j,l}}{\partial x} + \frac{2h_x^3}{3!}\frac{\partial^3 u_{i,j,l}}{\partial x^3} + \frac{2h_x^5}{5!}\frac{\partial^5 u_{i,j,l}}{\partial x^5} + \cdots$$

The left-hand side of the equation is defined as the first-order central differences at (i, j, l)-th grid point after dividing both sides of the equation by $2h_x$, that is

$$\delta_x u = \delta_x u_{i,j,l} = \frac{u_{i+1,j,l} - u_{i-1,j,l}}{2h_x} \tag{3.3}$$

On the other hand, the addition of the two equations in (3.2) yield

$$u_{i+1,j,l} + u_{i-1,j,l} = 2u_{i,j,l} + h_x^2 \frac{\partial^2 u_{i,j,l}}{\partial x^2} + \frac{2h_x^4}{4!}\frac{\partial^4 u_{i,j,l}}{\partial x^4} + \frac{2h_x^6}{6!}\frac{\partial^6 u_{i,j,l}}{\partial x^6} + \cdots$$

$$\frac{u_{i+1,j,l} - 2u_{i,j,l} + u_{i-1,j,l}}{h_x^2} = \frac{\partial^2 u_{i,j,l}}{\partial x^2} + \frac{h_x^2}{12}\frac{\partial^4 u_{i,j,l}}{\partial x^4} + \frac{h_x^4}{360}\frac{\partial^6 u_{i,j,l}}{\partial x^6} + \cdots$$

Define the left-hand side of the equation as the second-order central differences at $(i,j,l)-th$ grid point, that is

$$\delta_x^2 u = \delta_x^2 u_{i,j,l} = \frac{u_{i+1,j,l} - 2u_{i,j,l} + u_{i-1,j,l}}{h_x^2} \tag{3.4}$$

Repeat the steps above but for the variables y and z. Then define the first and second central difference operators δ_y, δ_z, δ_y^2 and δ_z^2 similarly. Furthermore, define the following notation

$$\begin{aligned}\Delta_h u = (\delta_x^2 + \delta_y^2 + \delta_z^2)u = &\frac{1}{h_x^2}\left(u_{i+1,j,l} + u_{i-1,j,l}\right) + \frac{1}{h_y^2}\left(u_{i,j+1,l} + u_{i,j-1,l}\right) + \frac{1}{h_z^2}\left(u_{i,j,l+1} + u_{i,j,l-1}\right) \\ &- 2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2}\right)u_{i,j,l}\end{aligned}$$

$$\delta_{xyy} u = \delta_x(\delta_y^2 u) = \frac{u_{i+1,j+1,l} + u_{i+1,j+1,l} - u_{i-1,j+1,l} - u_{i-1,j-1,l} - 2\left(u_{i+1,j,l} - u_{i-1,j,l}\right)}{2h^3}$$

and similarly in other directions too. Before introducing more notation, it is worth showing that the second-order central differences operator is commutative in Lemma 3.1.

Lemma 3.1. *The operators δ_x^2, δ_y^2 and δ_z^2 commute.*

Proof. Let u be a function of x,y and z. By definition

$$\begin{aligned}\delta_x^2\left(\delta_y^2 u_{i,j,l}\right) &= \delta_x^2\left(\frac{u_{i,j+1,l} - 2u_{i,j,l} + u_{i,j-1,l}}{h_y^2}\right) \\ &= \frac{1}{h_y^2}\left(\delta_x^2 u_{i,j+1,l} - 2\delta_x^2 u_{i,j,l} + \delta_x^2 u_{i,j-1,l}\right) \\ &= \frac{1}{h_y^2}\left(\frac{u_{i+1,j+1,l} - 2u_{i,j+1,l} + u_{i-1,j+1,l}}{h_x^2} - 2\frac{u_{i+1,j,l} - 2u_{i,j,l} + u_{i-1,j,l}}{h_x^2}\right. \\ &\quad \left. + \frac{u_{i+1,j-1,l} - 2u_{i,j-1,l} + u_{i-1,j-1,l}}{h_x^2}\right) \\ &= \frac{1}{h_x^2 h_y^2}\left(u_{i+1,j+1,l} - 2u_{i+1,j,l} + u_{i+1,j-1,l} - 2\left(u_{i,j+1,l} - 2u_{i,j,l} + u_{i,j-1,l}\right)\right. \\ &\quad \left. + u_{i-1,j+1,l} - 2u_{i-1,j,l} + u_{i-1,j-1,l}\right)\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{h_x^2}\left(\frac{u_{i+1,j+1,l} - 2u_{i+1,j,l} + u_{i+1,j-1,l}}{h_y^2} - 2\frac{u_{i,j+1,l} - 2u_{i,j,l} + u_{i,j-1,l}}{h_y^2}\right. \\
&\quad \left. + \frac{u_{i-1,j+1,l} - 2u_{i-1,j,l} + u_{i-1,j-1,l}}{h_y^2}\right) \\
&= \frac{1}{h_x^2}\left(\delta_y^2 u_{i+1,j,l} - 2\delta_y^2 u_{i,j,l} + \delta_y^2 u_{i-1,j,l}\right) \\
&= \delta_y^2 \frac{\left(u_{i+1,j,l} - 2u_{i,j,l} + u_{i-1,j,l}\right)}{h_x^2} = \delta_y^2\left(\delta_x^2 u_{i,j,l}\right)
\end{aligned}
$$

This proves that the operators δ_x^2 and δ_y^2 commute without any dependency on the spatial direction. In short, both of the operators can be replaced with any other operator and be proven similarly. Without any loss of generality, we may conclude that all three operators δ_x^2, δ_y^2, and δ_z^2 would commute with each other. □

Now the following operators would be equal by Lemma 3.1

$$
\begin{aligned}
\delta_x^2\delta_y^2 u = \delta_y^2\delta_x^2 u = \frac{1}{h_x^2 h_y^2}\Big[&u_{i+1,j+1,l} - 2u_{i+1,j,l} + u_{i+1,j-1,l} - 2\left(u_{i,j+1,l} - 2u_{i,j,l} + u_{i,j-1}\right) \\
&+ u_{i-1,j+1,l} - 2u_{i-1,j,l} + u_{i-1,j-1,l}\Big] \\
\delta_x^2\delta_z^2 u = \delta_z^2\delta_x^2 u = \frac{1}{h_x^2 h_z^2}\Big[&u_{i+1,j,l+1} - 2u_{i+1,j,l} + u_{i+1,j,l-1} - 2\left(u_{i,j,l+1} - 2u_{i,j,l} + u_{i,j,l-1}\right) \\
&+ u_{i-1,j,l+1} - 2u_{i-1,j,l} + u_{i-1,j,l-1}\Big] \\
\delta_y^2\delta_z^2 u = \delta_z^2\delta_y^2 u = \frac{1}{h_y^2 h_z^2}\Big[&u_{i,j+1,l+1} - 2u_{i,j+1,l} + u_{i,j+1,l-1} - 2\left(u_{i,j,l+1} - 2u_{i,j,l} + u_{i,j,l-1}\right) \\
&+ u_{i,j-1,l+1} - 2u_{i,j-1,l} + u_{i,j-1,l-1}\Big]
\end{aligned}
$$

Moreover when the grid steps are equal, that is $h = h_x = h_y = h_z$, the following operator can be simplified as

$$
(\delta_x^2\delta_y^2 + \delta_x^2\delta_z^2 + \delta_y^2\delta_z^2)u = \frac{1}{h^4}\Big[u_{i+1,j+1,l} + u_{i+1,j-1,l} + u_{i-1,j+1,l} + u_{i-1,j-1,l} + u_{i+1,j,l+1} + u_{i+1,j,l-1}
$$

$$
\begin{aligned}
&+ u_{i-1,j,l+1} + u_{i-1,j,l-1} + u_{i,j+1,l+1} + u_{i,j+1,l-1} + u_{i,j-1,l+1} + u_{i,j-1,l-1} \\
&- 4(u_{i+1,j,l} + u_{i-1,j,l} + u_{i,j+1,l} + u_{i,j-1,l} + u_{i,j,l+1} + u_{i,j,l-1}) \\
&+ 12u_{i+1,j,l}\Big] \\
\delta_x^2\delta_y^2\delta_z^2 u = \frac{1}{h^6}\Big[&u_{i+1,j+1,l+1} + u_{i-1,j+1,l+1} + u_{i+1,j-1,l+1} + u_{i-1,j-1,l+1} + u_{i+1,j+1,l-1} \\
&+ u_{i-1,j+1,l-1} + u_{i+1,j-1,l-1} + u_{i-1,j-1,l-1} - 2(u_{i+1,j+1,l} + u_{i-1,j+1,l} \\
&+ u_{i+1,j-1,l} + u_{i-1,j-1,l} + u_{i+1,j,l+1} + u_{i-1,j,l+1} + u_{i+1,j,l-1} \\
&+ u_{i-1,j,l-1} + u_{i,j+1,l+1} + u_{i,j-1,l+1} + u_{i,j+1,l-1} + u_{i,j-1,l-1}) \\
&+ 4(u_{i+1,j,l} + u_{i-1,j,l} + u_{i,j+1,l} + u_{i,j-1,l} + u_{i,j,l+1} + u_{i,j,l-1}) \\
&- 8u_{i,j,l}\Big]
\end{aligned}
$$

3.2 Second-Order Approximation Compact Scheme for Helmholtz Equation

From the Taylor series expansion, it is evident that the second order central differences δ_x^2, δ_y^2 and δ_z^2 is a direct estimation for the derivatives $\frac{\partial^2 u}{\partial x^2}$, $\frac{\partial^2 u}{\partial y^2}$ and $\frac{\partial^2 u}{\partial z^2}$ with exactly second-order accuracy. Therefore one could simply substitute the central differences into the Helmholtz equation (2.1), that is

$$\delta_x^2 u + \delta_y^2 u + \delta_z^2 u + k^2 u + \mathcal{O}\left(\max\{h_x^2, h_y^2, h_z^2\}\right) = f \tag{3.5}$$

Equation (3.5) is the second-order approximation scheme for the Helmholtz equation (2.1) and the resulting matrix form with this discretization can be presented as a 27-point stencil

notation with the following coefficients at each point

$$u_{i,j,l} : k_{i,j,l}^2 - 2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2}\right); \quad u_{i\pm1,j,l} : \frac{1}{h_x^2}; \quad u_{i,j\pm1,l} : \frac{1}{h_y^2}; \quad u_{i,j,l\pm1} : \frac{1}{h_z^2};$$

$$u_{i,j\pm1,l\pm1} = u_{i\pm1,j,l\pm1} = u_{i\pm1,j\pm1,l} = u_{i\pm1,j\pm1,l\pm1} : 0$$

In a two-dimensional case, the term $\delta_z^2 u$ would be removed from Equation (3.5) and the stencil only consists of the center, corner, and the side points, so the coefficient reduces to

$$u_{i,j} : k_{i,j}^2 - 2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right); \quad u_{i\pm1,j} : \frac{1}{h_x^2}; \quad u_{i,j\pm1} : \frac{1}{h_y^2}; \quad u_{i\pm1,j\pm1} : 0$$

It is worth noting that by setting $k^2 = 0$ to be the zero function, the 3D Helmholtz equation would be reduced to a 3D Poisson equation. This implies that by setting the coefficient k^2 to be zero in the stencil notation, one would obtain a discretization for the Poisson equation of second order. Similarly, the compact scheme for the upcoming fourth and sixth-order scheme for the Helmholtz equation can be modified to become a compact scheme for the Poisson equation.

3.3 Fourth-Order Approximation Compact Scheme for Helmholtz Equation

The fourth-order approximation to the derivatives based on the Taylor series expansion is given by

$$\delta_x^2 u = \frac{\partial^2}{\partial x^2} u + \frac{h_x^2}{12} \frac{\partial^4}{\partial x^4} u + \mathcal{O}(h_x^4) \tag{3.6}$$

In contrast to the second-order scheme, to derive a fourth-order accuracy scheme, one would be required to approximate the values of $\frac{\partial^4}{\partial x^4} u$ up to second-order accuracy. To accomplish

this, the standard fourth-order Padé approximation scheme is considered (see e.g. [27]). The idea is to apply the Taylor series to the derivatives of u with respect to xx. As a result

$$\delta_x^2(\frac{\partial^2}{\partial x^2}u) = \frac{\partial^2}{\partial x^2}\left(\frac{\partial^2}{\partial x^2}u\right) + \mathcal{O}(h_x^2) = \frac{\partial^4}{\partial x^4}u + \mathcal{O}(h_x^2)$$

This implies that the derivative $\frac{\partial^4}{\partial x^4}u$ can be estimated by $\delta_x^2\left(\frac{\partial^2}{\partial x^2}u\right)$ with second order accuracy. Substitute the new expression for $\frac{\partial^4}{\partial x^4}u$ into (3.6) and observed

$$\begin{aligned}\delta_x^2 u &= \frac{\partial^2}{\partial x^2}u + \frac{h_x^2}{12}\frac{\partial^4}{\partial x^4}u + \mathcal{O}(h_x^4)\\ &= \frac{\partial^2}{\partial x^2}u + \frac{h_x^2}{12}\left(\delta_x^2\left(\frac{\partial^2}{\partial x^2}u\right) + \mathcal{O}(h_x^2)\right) + \mathcal{O}(h_x^4)\\ &= \left(1 + \frac{h_x^2}{12}\delta_x^2\right)\left(\frac{\partial^2}{\partial x^2}u\right) + \mathcal{O}(h_x^4)\end{aligned}$$

Define $A_x = \left(1 + \frac{h_x^2}{12}\delta_x^2\right)$. It follows that the fourth-order rational approximation for the second-order derivatives with respect to xx is given by

$$\frac{\partial^2}{\partial x^2}u = A_x^{-1}\delta_x^2 u + O(h_x^4) \tag{3.7}$$

The fourth order approximation for the derivatives $\frac{\partial^2}{\partial y^2}u$ and $\frac{\partial^2}{\partial z^2}u$ can be obtained by repeating the steps above with the variable y and z respectively. Substituting these approximations into the Helmholtz equation yield

$$A_x^{-1}\delta_x^2 u + A_y^{-1}\delta_y^2 u + A_z^{-1}\delta_z^2 u + k^2 u + \mathcal{O}\left(\max\{h_x^4, h_y^4, h_z^4\}\right) = f \tag{3.8}$$

Now all inverse term in Equation (3.8) is required to be removed to express the fourth order scheme in stencil notation. Lemma 3.2 shows that the three operators A_x, A_y, and A_z commute with each other. This will enable one to remove all the inverse terms by multiplying both sides of the Equation (3.8) with the operator $A_x A_y A_z$.

Lemma 3.2. *The operators* $\left(1 + \frac{h_x^2}{12}\delta_x^2\right)$, $\left(1 + \frac{h_y^2}{12}\delta_y^2\right)$ *and* $\left(1 + \frac{h_z^2}{12}\delta_z^2\right)$ *commute.*

Proof. Let u be a function of x,y and z. First we will prove that the operators $\left(1 + \frac{h_x^2}{12}\delta_x^2\right)$

and $\left(1+\frac{h_y^2}{12}\delta_y^2\right)$ commute. By Lemma 3.1 the operator δ_x^2 and δ_y^2 commute. Hence

$$\begin{aligned}
\left(1+\frac{h_x^2}{12}\delta_x^2\right)\left(1+\frac{h_y^2}{12}\delta_y^2\right)u &= \left(1+\frac{h_y^2}{12}\delta_y^2\right)u + \left(\frac{h_x^2}{12}\delta_x^2\right)\left(1+\frac{h_y^2}{12}\delta_y^2\right)u \\
&= u + \frac{h_y^2}{12}\delta_y^2 u + \frac{h_x^2}{12}\delta_x^2 u + \frac{h_x^2 h_y^2}{144}\delta_x^2\delta_y^2 u \\
&= u + \frac{h_x^2}{12}\delta_x^2 u + \frac{h_y^2}{12}\delta_y^2 u + \frac{h_y^2}{12}\frac{h_x^2}{12}\delta_x^2\delta_y^2 u \\
&= \left(1+\frac{h_x^2}{12}\delta_x^2\right)u + \left(\frac{h_y^2}{12}\delta_y^2 + \frac{h_y^2}{12}\frac{h_x^2}{12}\delta_x^2\delta_y^2\right)u \\
&= \left(1+\frac{h_x^2}{12}\delta_x^2\right)u + \left[\frac{h_y^2}{12}\delta_y^2 + \left(\frac{h_y^2}{12}\delta_y^2\right)\left(\frac{h_x^2}{12}\delta_x^2\right)\right]u \\
&= \left(1+\frac{h_x^2}{12}\delta_x^2\right)u + \left(\frac{h_y^2}{12}\delta_y^2\right)\left(1+\frac{h_x^2}{12}\delta_x^2\right)u_{i,,lj} \\
&= \left(1+\frac{h_y^2}{12}\delta_y^2\right)\left(1+\frac{h_x^2}{12}\delta_x^2\right)u
\end{aligned}$$

This prove that the operators $\left(1+\frac{h_x^2}{12}\delta_x^2\right)$ and $\left(1+\frac{h_y^2}{12}\delta_y^2\right)$ commute. Similar to the proof in Lemma 3.1, this proof has no dependency on the spatial direction as well. Without any loss of generality, all three operators would commute with each other. □

Now both side of Equation (3.8) is multiplied by $A_xA_yA_z$ and the fourth order Padé approximation scheme can be written as

$$A_yA_z\delta_x^2u + A_xA_z\delta_y^2u + A_xA_y\delta_z^2u + A_xA_yA_zk^2u + \mathcal{O}\left(\max\{h_x^4, h_y^4, h_z^4\}\right) = A_xA_yA_zf \quad (3.9)$$

Expanding the operator A_xA_y gives us

$$\begin{aligned}
A_xA_y &= \left(1+\frac{h_x^2}{12}\delta_x^2\right)\left(1+\frac{h_y^2}{12}\delta_y^2\right) \\
&= 1 + \frac{h_x^2}{12}\delta_x^2 + \frac{h_y^2}{12}\delta_y^2 + \frac{h_x^2h_y^2}{144}\delta_x^2\delta_y^2
\end{aligned}$$

Since this is a fourth-order approximation scheme, one may remove all terms with $h_x^2h_y^2$. Expanding the terms A_xA_z, A_yA_z and $A_xA_yA_z$ would produce similar results. Overall we

get

$$\begin{aligned}
A_x A_y &= 1 + \frac{h_x^2}{12}\delta_x^2 + \frac{h_y^2}{12}\delta_y^2 \\
A_x A_z &= 1 + \frac{h_x^2}{12}\delta_x^2 + \frac{h_z^2}{12}\delta_z^2 \\
A_y A_z &= 1 + \frac{h_y^2}{12}\delta_y^2 + \frac{h_z^2}{12}\delta_z^2 \\
A_x A_y A_z &= \left(1 + \frac{h_x^2}{12}\delta_x^2 + \frac{h_y^2}{12}\delta_y^2\right)\left(1 + \frac{h_z^2}{12}\delta_z^2\right) \\
&= 1 + \frac{h_x^2}{12}\delta_x^2 + \frac{h_y^2}{12}\delta_y^2 + \frac{h_z^2}{12}\delta_z^2
\end{aligned}$$

Substituting these term into Equation (3.9) yields

$$\begin{aligned}
\Delta_h u + \frac{(h_x^2 + h_y^2)}{12}\delta_x^2\delta_y^2 u + \frac{(h_x^2 + h_z^2)}{12}\delta_x^2\delta_z^2 u + \frac{(h_y^2 + h_z^2)}{12}\delta_y^2\delta_z^2 u + \left(1 + \frac{h_x^2}{12}\delta_x^2 + \frac{h_y^2}{12}\delta_y^2 + \frac{h_z^2}{12}\delta_z^2\right)(k^2 u) \\
= \left(1 + \frac{h_x^2}{12}\delta_x^2 + \frac{h_y^2}{12}\delta_y^2 + \frac{h_z^2}{12}\delta_z^2\right) f + \mathcal{O}\left(\max\{h_x^4, h_y^4, h_z^4\}\right)
\end{aligned} \tag{3.10}$$

The matrix generated by the left-hand side of Equation (3.10) can be presented in stencil notation with the following coefficients

$$u_{i,j,l} : \frac{k_{i,j,l}^2}{2} - \frac{4}{3}\left(\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2}\right);$$

$$u_{i\pm1,j,l} : \frac{k_{i\pm1,j,l}^2}{12} + \frac{2}{3h_x^2} - \frac{1}{6}\left(\frac{1}{h_y^2} + \frac{1}{h_z^2}\right);$$

$$u_{i,j\pm1,l} : \frac{k_{i,j\pm1,l}^2}{12} + \frac{2}{3h_y^2} - \frac{1}{6}\left(\frac{1}{h_x^2} + \frac{1}{h_z^2}\right);$$

$$u_{i,j,l\pm1} : \frac{k_{i,j,l\pm1}^2}{12} + \frac{2}{3h_z^2} - \frac{1}{6}\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right);$$

$$u_{i,j\pm1,l\pm1} : \frac{1}{12}\left(\frac{1}{h_y^2} + \frac{1}{h_z^2}\right)$$

$$u_{i\pm1,j,l\pm1} : \frac{1}{12}\left(\frac{1}{h_x^2} + \frac{1}{h_z^2}\right)$$

$$u_{i\pm1,j\pm1,l} : \frac{1}{12}\left(\frac{1}{h_x^2}+\frac{1}{h_y^2}\right)$$

$$u_{i\pm1,j\pm1,l\pm1} : 0$$

After removing all terms with δ_z^2 is removed from Equation (3.10), the coefficient for the two-dimensional case is given by

$$u_{i,j} : \frac{2k_{i,j}^2}{3} - \frac{5}{3}\left(\frac{1}{h_x^2}+\frac{1}{h_y^2}\right);$$

$$u_{i\pm1,j} : \frac{k_{i\pm1,j}^2}{12} + \frac{5}{6h_x^2} - \frac{1}{6}\left(\frac{1}{h_y^2}\right);$$

$$u_{i\pm1,j} : \frac{k_{i,j\pm1}^2}{12} + \frac{5}{6h_y^2} - \frac{1}{6}\left(\frac{1}{h_x^2}\right);$$

$$u_{i\pm1,j\pm1} : \frac{1}{12}\left(\frac{1}{h_x^2}+\frac{1}{h_y^2}\right)$$

3.4 Sixth-Order Approximation Compact Scheme for Helmholtz Equation

From the Taylor expansion in Equation (3.4) we have

$$\frac{\partial^2 u}{\partial x^2} = \delta_x^2 u - \frac{h_x^2}{12}\frac{\partial^4 u}{\partial x^4} - \frac{h_x^4}{360}\frac{\partial^6 u}{\partial x^6} + \mathcal{O}(h_x^6)$$

A similar result can be obtained for the second derivatives in y- and z-directions. Substituting these expressions into the Helmholtz equation yield

$$\Delta_h u - \frac{h^2}{12}\left(\frac{\partial^4 u}{\partial x^4}+\frac{\partial^4 u}{\partial y^4}+\frac{\partial^4 u}{\partial z^4}\right) - \frac{h^4}{360}\left(\frac{\partial^6 u}{\partial x^6}+\frac{\partial^6 u}{\partial y^6}+\frac{\partial^6 u}{\partial z^6}\right) + k^2 u + \mathcal{O}(h^6) = f \qquad (3.11)$$

Define $I_1 = \frac{\partial^4 u}{\partial x^4}+\frac{\partial^4 u}{\partial y^4}+\frac{\partial^4 u}{\partial z^4}$ and $I_2 = \frac{\partial^6 u}{\partial x^6}+\frac{\partial^6 u}{\partial y^6}+\frac{\partial^6 u}{\partial z^6}$. Then to have a sixth-order accuracy scheme for the Helmholtz equation it is required to approximate I_1 to fourth-order accuracy

and I_2 to second-order accuracy.

This sixth-order compact scheme for the approximation of the 3D Helmholtz equation was developed by Turkel et al [5]. Unlike the previous compact schemes, the sixth-order compact scheme requires a uniform grid step that is $h = h_x = h_y = h_z$. Hence in this section only, we will assume that the grid step in all directions is equal and the notation h would be used instead.

Furthermore, the sixth-order scheme is going to require derivatives of the coefficient k^2 and right-hand side f to be known or be able to approximate up to second or fourth-order accuracy. Presently, all derivatives for the function k^2 and f will be assumed to be computable analytically. At the end of this section, the issue concerning the derivatives would be addressed to lessen the condition required for this approach. The work presented in this section closely follows Turkel et al works in [5].

3.4.1 Second-order approximation to I_2

The idea for developing the sixth-order approximation scheme is to introduce the term I_2 into the Helmholtz equation by differentiating the Helmholtz equation. By doing so, we wish to express I_2 in a way it can be computed efficiently. Thus, the Helmholtz equation is differentiated three times, with respect to $xxxx$, $yyyy$, and $zzzz$.

$$\left(\frac{\partial^6}{\partial x^6} + \frac{\partial^6}{\partial x^4 \partial y^2} + \frac{\partial^6}{\partial x^4 \partial z^2}\right) u + \frac{\partial^4}{\partial x^4}(k^2 u) = \frac{\partial^4}{\partial x^4}(f)$$
$$\left(\frac{\partial^6}{\partial y^4 \partial x^2} + \frac{\partial^6}{\partial y^6} + \frac{\partial^6}{\partial y^4 \partial z^2}\right) u + \frac{\partial^4}{\partial y^4}(k^2 u) = \frac{\partial^4}{\partial y^4}(f)$$
$$\left(\frac{\partial^6}{\partial z^4 \partial x^2} + \frac{\partial^6}{\partial z^4 \partial y^2} + \frac{\partial^6}{\partial z^6}\right) u + \frac{\partial^4}{\partial z^4}(k^2 u) = \frac{\partial^4}{\partial z^4}(f)$$

Adding all three equations and rearranging the terms on the left-hand side of the equation

would result in

$$I_2 = \left(\frac{\partial^6}{\partial x^6} + \frac{\partial^6}{\partial y^6} + \frac{\partial^6}{\partial z^6}\right) u = -\left(\frac{\partial^6 u}{\partial x^4 \partial y^2} + \frac{\partial^6 u}{\partial x^4 \partial z^2} + \frac{\partial^6 u}{\partial y^4 \partial x^2} + \frac{\partial^6 u}{\partial y^4 \partial z^2} + \frac{\partial^6 u}{\partial z^4 \partial x^2} + \frac{\partial^6 u}{\partial z^4 \partial y^2}\right) - \left(\frac{\partial^4}{\partial x^4} + \frac{\partial^4}{\partial y^4} + \frac{\partial^4}{\partial z^4}\right)(k^2 u - f) \tag{3.12}$$

By the assumption made earlier, all fourth-order derivatives of f can be calculated analytically. So the challenge lies with the remaining terms. Let $J_1 = \frac{\partial^6 u}{\partial x^4 \partial y^2} + \frac{\partial^6 u}{\partial x^4 \partial z^2} + \frac{\partial^6 u}{\partial y^4 \partial x^2} + \frac{\partial^6 u}{\partial y^4 \partial z^2} + \frac{\partial^6 u}{\partial z^4 \partial x^2} + \frac{\partial^6 u}{\partial z^4 \partial y^2}$ and $J_2 = (\frac{\partial^4}{\partial x^4} + \frac{\partial^4}{\partial y^4} + \frac{\partial^4}{\partial z^4})(k^2 u)$. To find an approximation for the term J_1, the Helmholtz equation is differentiated with respect to $xxyy$, $xxzz$ and $yyzz$ and we have

$$\left(\frac{\partial^6}{\partial x^4 \partial y^2} + \frac{\partial^6}{\partial x^2 \partial y^4} + \frac{\partial^6}{\partial x^2 \partial y^2 \partial z^2}\right) u + \frac{\partial^4}{\partial x^2 \partial y^2}(k^2 u) = \frac{\partial^4}{\partial x^2 \partial y^2}(f)$$
$$\left(\frac{\partial^6}{\partial x^4 \partial z^2} + \frac{\partial^6}{\partial x^2 \partial y^2 \partial z^2} + \frac{\partial^6}{\partial x^2 \partial z^4}\right) u + \frac{\partial^4}{\partial x^2 \partial z^2}(k^2 u) = \frac{\partial^4}{\partial x^2 \partial z^2}(f)$$
$$\left(\frac{\partial^6}{\partial x^2 \partial y^2 \partial z^2} + \frac{\partial^6}{\partial y^4 \partial z^2} + \frac{\partial^6}{\partial y^2 \partial z^4}\right) u + \frac{\partial^4}{\partial y^2 \partial z^2}(k^2 u) = \frac{\partial^4}{\partial y^2 \partial z^2}(f)$$

Adding and rearranging all three equations again would allow us to express J_1 as

$$J_1 = -3\left(\frac{\partial^6}{\partial x^2 \partial y^2 \partial z^2}\right) u - \left(\frac{\partial^4}{\partial x^2 \partial y^2} + \frac{\partial^4}{\partial x^2 \partial z^2} + \frac{\partial^4}{\partial y^2 \partial z^2}\right)\left(k^2 u - f\right)$$

Now all the derivatives are only second order in each spatial direction and the estimation only requires second-order accuracy. This meant that the value of J_1 can now be estimated by replacing the second-order derivatives with the second-order central difference approximations. In particular,

$$J_1 = -3\delta_x^2 \delta_y^2 \delta_z^2 u - \left(\delta_x^2 \delta_y^2 + \delta_x^2 \delta_z^2 + \delta_y^2 \delta_z^2\right)(k^2 u) + \left(\frac{\partial^4}{\partial x^2 \partial y^2} + \frac{\partial^4}{\partial x^2 \partial z^2} + \frac{\partial^4}{\partial y^2 \partial z^2}\right)(f) + \mathcal{O}(h^2) \tag{3.13}$$

Now it remains to approximate J_2 with second-order accuracy. A straightforward expansion of the fourth derivatives of $(k^2 u)$ would require the fourth derivatives of k^2. If k^2 is a

complicated formula then its fourth-order derivatives become exceedingly complex. Hence Turkel et al took an alternative approach and derive an explicit formula that would require only second derivatives of k^2. The following lemmas are established for this purpose.

Lemma 3.3.

$$\frac{\partial}{\partial x}u = \delta_x u + \frac{h^2}{6}\left(\delta_x\delta_y^2 u + \delta_x\delta_z^2 u + \delta_x(k^2 u) - \frac{\partial}{\partial x}f\right) + \mathcal{O}(h^4)$$

Proof. From the Taylor expansion we have

$$\frac{\partial}{\partial x}u = \delta_x u - \frac{h^2}{6}\frac{\partial^3}{\partial x^3}u + \mathcal{O}(h^4) \tag{3.14}$$

Then differentiating the Helmholtz equation with respect to x yield

$$\frac{\partial^3}{\partial x^3}u + \frac{\partial^3}{\partial x\partial y^2}u + \frac{\partial^3}{\partial x\partial z^2}u + \frac{\partial}{\partial x}(k^2 u) = \frac{\partial}{\partial x}f$$

$$\begin{aligned}\frac{\partial^3}{\partial x^3}u = & -\left(\frac{\partial^3}{\partial x\partial y^2}u + \frac{\partial^3}{\partial x\partial z^2}u + \frac{\partial}{\partial x}(k^2 u)\right) + \frac{\partial}{\partial x}f \\ = & -\left(\delta_x\delta_y^2 u + \delta_x\delta_z^2 u + \delta_x(k^2 u)\right) + \frac{\partial}{\partial x}f + \mathcal{O}(h^2)\end{aligned} \tag{3.15}$$

Substitute (3.15) into (3.14) give us exactly

$$\frac{\partial}{\partial x}u = \delta_x u + \frac{h^2}{6}\left(\delta_x\delta_y^2 u + \delta_x\delta_z^2 u + \delta_x(k^2 u) - \frac{\partial}{\partial x}f\right) + \mathcal{O}(h^4)$$

□

Lemma 3.4.

$$\Delta(k^2 u) = k^2 f + \left(\Delta(k^2) - k^4\right)u + 2\left(\frac{\partial}{\partial x}(k^2)\frac{\partial}{\partial x}(u) + \frac{\partial}{\partial y}(k^2)\frac{\partial}{\partial y}(u) + \frac{\partial}{\partial z}(k^2)\frac{\partial}{\partial z}(u)\right)$$

Proof. From the product rules of derivatives

$$\frac{\partial^2}{\partial x^2}(k^2 u) = \frac{\partial^2}{\partial x^2}(k^2)u + 2\frac{\partial}{\partial x}(k^2)\frac{\partial}{\partial x}(u) + k^2\left(\frac{\partial^2}{\partial x^2}u\right)$$

Repeat this for the variables y and z, then the sum of all three equations yield

$$\Delta(k^2u) = \Delta(k^2)u + 2\left(\frac{\partial}{\partial x}(k^2)\frac{\partial}{\partial x}(u) + \frac{\partial}{\partial y}(k^2)\frac{\partial}{\partial y}(u) + \frac{\partial}{\partial z}(k^2)\frac{\partial}{\partial z}(u)\right) + k^2\Delta u$$

Using the fact that u is the solution of the Helmholtz equation, so $\Delta u = f - k^2u$, we can rewrite it as

$$\begin{aligned}\Delta(k^2u) &= \Delta(k^2)u + 2\left(\frac{\partial}{\partial x}(k^2)\frac{\partial}{\partial x}(u) + \frac{\partial}{\partial y}(k^2)\frac{\partial}{\partial y}(u) + \frac{\partial}{\partial z}(k^2)\frac{\partial}{\partial z}(u)\right) + k^2(f - k^2u) \\ &= k^2f + (\Delta(k^2) - k^4)u + 2\left(\frac{\partial}{\partial x}(k^2)\frac{\partial}{\partial x}(u) + \frac{\partial}{\partial y}(k^2)\frac{\partial}{\partial y}(u) + \frac{\partial}{\partial z}(k^2)\frac{\partial}{\partial z}(u)\right)\end{aligned}$$

□

Combining Lemma 3.3 and Lemma 3.4 would provide a fourth order accuracy approximation to the term $\Delta(k^2u)$ as follows

$$\begin{aligned}\Delta(k^2u) = (k^2)f + (\Delta(k^2) - k^4)u &+ 2\frac{\partial k^2}{\partial x}\left(\delta_x u + \frac{h^2}{6}\left(\delta_x\delta_y^2u + \delta_x\delta_z^2u + \delta_x(k^2u) - \frac{\partial f}{\partial x}\right)\right) \\ &+ 2\frac{\partial k^2}{\partial y}\left(\delta_y u + \frac{h^2}{6}\left(\delta_y\delta_x^2u + \delta_y\delta_z^2u + \delta_y(k^2u) - \frac{\partial f}{\partial y}\right)\right) \\ &+ 2\frac{\partial k^2}{\partial z}\left(\delta_z u + \frac{h^2}{6}\left(\delta_z\delta_x^2u + \delta_z\delta_y^2u + \delta_z(k^2u) - \frac{\partial f}{\partial z}\right)\right) + \mathcal{O}(h^4) \qquad (3.16)\end{aligned}$$

Lemma 3.5.

$$\frac{h^2}{12}J_2 = \Delta_h(k^2u) - \Delta(k^2u) + \mathcal{O}(h^4)$$

where $\Delta(k^2u)$ is given to fourth order accuracy by (3.16)

Proof. Replacing the function u for the Taylor series expression in Equation (3.6) with (k^2u) yield

$$\delta_x^2(k^2u) = \frac{\partial^2}{\partial x^2}(k^2u) + \frac{h^2}{12}\frac{\partial^4}{\partial x^4}(k^2u) + \mathcal{O}(h^4)$$

Repeat it for y- and z-directions then adding all three directions to get

$$\left(\delta_x^2+\delta_y^2+\delta_z^2\right)(k^2u) = \left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)(k^2u)+\frac{h^2}{12}\left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}\right)(k^2u)+\mathcal{O}(h^4)$$

$$\Delta_h(k^2u) = \Delta(k^2u)+\frac{h^2}{12}J_2+\mathcal{O}(h^4)$$

$$\frac{h^2}{12}J_2 = \Delta_h(k^2u)-\Delta(k^2u)+\mathcal{O}(h^4)$$

□

It is worth remarking that the assumption where the step sizes are equal is crucial in this lemma to combine all fourth-order derivatives of (k^2u) and obtain the term J_2. Without this assumption, it would not be possible to compute the value of J_2 with this lemma. Now Lemma (3.5) allowed the term J_2 to be estimated up to second order accuracy as follows

$$J_2 = \frac{12}{h^2}\left[\Delta_h(k^2u)-\Delta(k^2u)\right]+\mathcal{O}(h^2) \tag{3.17}$$

The term I_2 can now be approximated by substituting Equation (3.13) and Equation (3.17) into Equation (3.12) as follows

$$\begin{aligned} I_2 &= \left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}\right)f - J_1 - J_2 \\ &= \left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}-\frac{\partial^4}{\partial x^2\partial y^2}-\frac{\partial^4}{\partial x^2\partial z^2}-\frac{\partial^4}{\partial y^2\partial z^2}\right)f+3\delta_x^2\delta_y^2\delta_z^2u \\ &\quad+\left(\delta_x^2\delta_y^2+\delta_x^2\delta_z^2+\delta_y^2\delta_z^2\right)(k^2u)-\frac{12}{h^2}\left[(\delta_x^2+\delta_y^2+\delta_z^2)(k^2u)-\Delta(k^2u)\right]+\mathcal{O}(h^2) \end{aligned} \tag{3.18}$$

where $\Delta(k^2u)$ is given to fourth order accuracy by Equation (3.16).

3.4.2 Fourth-order approximation to I_1

Recall that $I_1 = \frac{\partial^4 u}{\partial x^4} + \frac{\partial^4 u}{\partial y^4} + \frac{\partial^4 u}{\partial z^4}$. Similar to the previous subsection, we begin by differentiating the Helmholtz equation twice with respect to xx, yy, and zz and we observe that

$$\left(\frac{\partial^4}{\partial x^4} + \frac{\partial^4}{\partial x^2 \partial y^2} + \frac{\partial^4}{\partial x^2 \partial z^2}\right) u + \frac{\partial^2}{\partial x^2}(k^2 u) = \frac{\partial^2}{\partial x^2}(f)$$
$$\left(\frac{\partial^4}{\partial y^2 \partial x^2} + \frac{\partial^4}{\partial y^4} + \frac{\partial^4}{\partial y^2 \partial z^2}\right) u + \frac{\partial^2}{\partial y^2}(k^2 u) = \frac{\partial^2}{\partial y^2}(f)$$
$$\left(\frac{\partial^4}{\partial z^2 \partial x^2} + \frac{\partial^4}{\partial z^2 \partial y^2} + \frac{\partial^4}{\partial z^4}\right) u + \frac{\partial^2}{\partial z^2}(k^2 u) = \frac{\partial^4}{\partial z^2}(f)$$

Adding all three equations and rearranging the terms would result in

$$I_1 = \left(\frac{\partial^4}{\partial x^4} + \frac{\partial^4}{\partial y^4} + \frac{\partial^4}{\partial z^4}\right) u = -2\left(\frac{\partial^4}{\partial x^2 \partial y^2} + \frac{\partial^4}{\partial x^2 \partial z^2} + \frac{\partial^4}{\partial y^2 \partial z^2}\right) u - \Delta(k^2 u) + \Delta f \tag{3.19}$$

By Equation (3.16) the term $\Delta(k^2 u)$ has been approximated up to fourth-order accuracy and we assume the derivatives of f can be computed analytically. Hence it remains to compute the term $\left(\frac{\partial^4}{\partial x^2 \partial y^2} + \frac{\partial^4}{\partial x^2 \partial z^2} + \frac{\partial^4}{\partial y^2 \partial z^2}\right) u$ up to fourth order accuracy. The next lemma is developed for this purpose.

Lemma 3.6.

$$\delta_x^2 \delta_y^2 u = \frac{\partial^4}{\partial x^2 \partial y^2} u + \frac{h^2}{12}\left(\frac{\partial^6}{\partial x^4 \partial y^2} + \frac{\partial^6}{\partial x^2 \partial y^4}\right) u + \mathcal{O}(h^4)$$

Proof. Recall from the fourth-order approximation of the second-order derivatives as in Equation (3.6) we have

$$\delta_x^2 u = \frac{\partial^2}{\partial x^2} u + \frac{h^2}{12}\frac{\partial^4}{\partial x^4} u + \mathcal{O}(h^4) \tag{3.20}$$

$$\delta_y^2 u = \frac{\partial^2}{\partial y^2} u + \frac{h^2}{12}\frac{\partial^4}{\partial y^4} u + \mathcal{O}(h^4) \tag{3.21}$$

Replaced the function u in (3.20) with the function $(\delta_y^2 u)$ and then substitute the expression for $(\delta_y^2 u)$ in (3.21) yield

$$\begin{aligned}
\delta_x^2(\delta_y^2 u) &= \frac{\partial^2}{\partial x^2}(\delta_y^2 u) + \frac{h^2}{12}\frac{\partial^4}{\partial x^4}(\delta_y^2 u) + \mathcal{O}(h^4) \\
&= \frac{\partial^2}{\partial x^2}\left(\frac{\partial^2}{\partial y^2}u + \frac{h^2}{12}\frac{\partial^4}{\partial y^4}u + \mathcal{O}(h^4)\right) + \frac{h^2}{12}\frac{\partial^4}{\partial x^4}\left(\frac{\partial^2}{\partial y^2}u + \mathcal{O}(h^2)\right) + \mathcal{O}(h^4) \\
&= \frac{\partial^2}{\partial x^2}\frac{\partial^2}{\partial y^2}u + \frac{h^2}{12}\frac{\partial^2}{\partial x^2}\frac{\partial^4}{\partial y^4}u + \frac{h^2}{12}\frac{\partial^4}{\partial x^4}\frac{\partial^2}{\partial y^2}u + \mathcal{O}(h^4) \\
&= \frac{\partial^4}{\partial x^2 \partial y^2}u + \frac{h^2}{12}\left(\frac{\partial^6}{\partial x^4 \partial y^2} + \frac{\partial^6}{\partial x^2 \partial y^4}\right)u + \mathcal{O}(h^4)
\end{aligned}$$

□

The results of Lemma 3.6 could also be applied to the operators $\delta_x^2\delta_z^2$ and $\delta_y^2\delta_z^2$ for similar results. The sum of all three operators would give us

$$\begin{aligned}
(\delta_x^2\delta_y^2 + \delta_x^2\delta_z^2 + \delta_y^2\delta_z^2)u = &\left(\frac{\partial^4}{\partial x^2\partial y^2} + \frac{\partial^4}{\partial x^2\partial z^2} + \frac{\partial^4}{\partial y^2\partial z^2}\right)u \\
&+ \frac{h^2}{12}\left(\frac{\partial^6}{\partial x^4\partial y^2} + \frac{\partial^6}{\partial x^2\partial y^4} + \frac{\partial^6}{\partial x^4\partial z^2} + \frac{\partial^6}{\partial x^2\partial z^4} + \frac{\partial^6}{\partial y^4\partial z^2} + \frac{\partial^6}{\partial y^2\partial z^4}\right)u \\
&+ \mathcal{O}(h^4) \\
= &\left(\frac{\partial^4}{\partial x^2\partial y^2} + \frac{\partial^4}{\partial x^2\partial z^2} + \frac{\partial^4}{\partial y^2\partial z^2}\right)u + \frac{h^2}{12}J_1 + \mathcal{O}(h^4)
\end{aligned}$$

where J_1 can be approximated up to second-order accuracy with Equation (3.13). So

$$\left(\frac{\partial^4}{\partial x^2\partial y^2} + \frac{\partial^4}{\partial x^2\partial z^2} + \frac{\partial^4}{\partial y^2\partial z^2}\right)u = (\delta_x^2\delta_y^2 + \delta_x^2\delta_z^2 + \delta_y^2\delta_z^2)u - \frac{h^2}{12}J_1 + \mathcal{O}(h^4)$$

Thus,

$$\begin{aligned}
I_1 = &-2\left(\frac{\partial^4}{\partial x^2\partial y^2} + \frac{\partial^4}{\partial x^2\partial z^2} + \frac{\partial^4}{\partial y^2\partial z^2}\right)u - \Delta(k^2 u) + \Delta f \\
= &-2(\delta_x^2\delta_y^2 + \delta_x^2\delta_z^2 + \delta_y^2\delta_z^2)u + \frac{h^2}{6}J_1 - \Delta(k^2 u) + \Delta f + \mathcal{O}(h^4) \qquad (3.22)
\end{aligned}$$

With Equation (3.18) and Equation (3.22), now we have

$$\frac{h^2}{12}I_1 + \frac{h^4}{360}I_2 = -\frac{h^2}{6}(\delta_x^2\delta_y^2 + \delta_x^2\delta_z^2 + \delta_y^2\delta_z^2)u + \frac{h^4}{72}J_1 - \frac{h^2}{12}\Delta(k^2 u) + \frac{h^2}{12}\Delta f$$

$$+\frac{h^4}{360}\left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}\right)f-\frac{h^4}{360}J_1-\frac{h^4}{360}J_2+\mathcal{O}(h^6)$$

$$\begin{aligned}=&-\frac{h^2}{6}(\delta_x^2\delta_y^2+\delta_x^2\delta_z^2+\delta_y^2\delta_z^2)u+\frac{h^4}{90}J_1-\frac{h^4}{360}J_2-\frac{h^2}{12}\Delta(k^2u)\\&+\frac{h^2}{12}\Delta f+\frac{h^4}{360}\left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}\right)f+\mathcal{O}(h^6)\end{aligned}$$

$$\begin{aligned}=&-\frac{h^2}{6}(\delta_x^2\delta_y^2+\delta_x^2\delta_z^2+\delta_y^2\delta_z^2)u-\frac{h^4}{30}\delta_x^2\delta_y^2\delta_z^2u-\frac{h^4}{90}\left(\delta_x^2\delta_y^2+\delta_x^2\delta_z^2+\delta_y^2\delta_z^2\right)(k^2u)\\&-\frac{h^2}{30}(\delta_x^2+\delta_y^2+\delta_z^2)(k^2u)-\frac{h^2}{20}\Delta(k^2u)+\frac{h^2}{12}\Delta f\\&+\frac{h^4}{360}\left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}\right)f+\frac{h^4}{90}\left(\frac{\partial^4}{\partial x^2\partial y^2}+\frac{\partial^4}{\partial x^2\partial z^2}+\frac{\partial^4}{\partial y^2\partial z^2}\right)f\\&+\mathcal{O}(h^6)\end{aligned}\tag{3.23}$$

Let $(k^2)_x=\frac{\partial}{\partial x}(k^2)$, $(k^2)_y=\frac{\partial}{\partial y}(k^2)$, $(k^2)_z=\frac{\partial}{\partial z}(k^2)$ and define $U_\alpha=(1+\frac{k^2h^2}{\alpha})u$ where $\alpha\in\mathbb{R}$.

The left-hand side of the sixth-order approximation scheme is given by

$$\begin{aligned}&\Delta_hU_{30}+\frac{h^2}{6}(\delta_x^2\delta_y^2+\delta_x^2\delta_z^2+\delta_y^2\delta_z^2)U_{15}+\frac{h^2}{10}\left[(k^2)_x\delta_x+(k^2)_y\delta_y+(k^2)_z\delta_z\right]U_6\\&+\frac{(k^2)_xh^4}{60}\left(\delta_{xyy}+\delta_{xzz}\right)u+\frac{(k^2)_yh^4}{60}\left(\delta_{xxy}+\delta_{yzz}\right)u+\frac{(k^2)_zh^4}{60}\left(\delta_{xxz}+\delta_{yyz}\right)u\\&+\frac{h^4}{30}\delta_x^2\delta_y^2\delta_z^2u+\frac{h^2}{20}(\Delta k^2-k^4)u+k^2u\end{aligned}\tag{3.24}$$

On the other hand, the right-hand side of the sixth-order approximation scheme is given by

$$\begin{aligned}&\left(1-\frac{k^2h^2}{20}\right)f+\frac{h^4}{60}\left((k^2)_x\frac{\partial}{\partial x}+(k^2)_y\frac{\partial}{\partial y}+(k^2)_z\frac{\partial}{\partial z}\right)f+\frac{h^2}{12}\Delta f\\&+\frac{h^4}{360}\left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}\right)f+\frac{h^4}{90}\left(\frac{\partial^4}{\partial x^2\partial y^2}+\frac{\partial^4}{\partial x^2\partial z^2}+\frac{\partial^4}{\partial y^2\partial z^2}\right)f\end{aligned}\tag{3.25}$$

Before rewriting Equation (3.24) in terms of stencil notation, both sides of the sixth-order compact scheme would be multiplied by the term h^2. By doing so, it removes the amount of division required in the coefficient. Moreover, in terms of computation cost, the division is much more computationally heavy than multiplication. This allows the implementation of the coefficient to be calculated faster and more efficiently. Consequently, the matrix from the sixth-order scheme would have the following coefficients

$$u_{i,j,l} : -\frac{64}{15} + \frac{14}{15}h^2k_{i,j,l}^2 + \frac{h^4}{20}\left(\Delta(k^2) - k_{i,j,l}^4\right);$$

$$u_{i\pm1,j,l} : \frac{7}{15} - \frac{1}{90}h^2k_{i\pm1,j,l}^2 \pm \frac{h^3}{60}(k^2)_x\left(1 + \frac{1}{2}h^2k_{i\pm1,j,l}^2\right);$$

$$u_{i,j\pm1,l} : \frac{7}{15} - \frac{1}{90}h^2k_{i,j\pm1,l}^2 \pm \frac{h^3}{60}(k^2)_y\left(1 + \frac{1}{2}h^2k_{i,j\pm1,l}^2\right);$$

$$u_{i,j,l\pm1} : \frac{7}{15} - \frac{1}{90}h^2k_{i,j,l\pm1}^2 \pm \frac{h^3}{60}(k^2)_z\left(1 + \frac{1}{2}h^2k_{i,j,l\pm1}^2\right);$$

$$u_{i+1,j\pm1,l} : \frac{1}{10} + \frac{1}{90}h^2k_{i+1,j\pm1,l}^2 + \frac{h^3}{120}\left[(k^2)_x \pm (k^2)_y\right];$$

$$u_{i-1,j\pm1,l} : \frac{1}{10} + \frac{1}{90}h^2k_{i-1,j\pm1,l}^2 - \frac{h^3}{120}\left[(k^2)_x \mp (k^2)_y\right];$$

$$u_{i+1,j,l\pm1} : \frac{1}{10} + \frac{1}{90}h^2k_{i+1,j,l\pm1}^2 + \frac{h^3}{120}\left[(k^2)_x \pm (k^2)_z\right];$$

$$u_{i-1,j,l\pm1} : \frac{1}{10} + \frac{1}{90}h^2k_{i-1,j,l\pm1}^2 - \frac{h^3}{120}\left[(k^2)_x \mp (k^2)_z\right];$$

$$u_{i,j+1,l\pm1} : \frac{1}{10} + \frac{1}{90}h^2k_{i,j+1,l\pm1}^2 + \frac{h^3}{120}\left[(k^2)_y \pm (k^2)_z\right];$$

$$u_{i,j-1,l\pm1} : \frac{1}{10} + \frac{1}{90}h^2k_{i,j-1,l\pm1}^2 - \frac{h^3}{120}\left[(k^2)_y \mp (k^2)_z\right];$$

$$u_{i\pm1,j\pm1,l\pm1} : \frac{1}{30}$$

The value of k^2 is always evaluated at the same stencil point as u. However, the derivatives $(k^2)_x, (k^2)_y, (k^2)_z$ and $\Delta(k^2)$ are always evaluated at the center point of the stencil (i, j, l). In a two-dimensional case, this stencil becomes to

$$u_{i,j} : -\frac{10}{3} + \frac{41}{45}h^2k_{i,j}^2 + \frac{h^4}{20}\left(\Delta(k^2) - k_{i,j}^4\right);$$

$$u_{i\pm1,j} : \frac{2}{3} + \frac{1}{90}h^2k_{i\pm1,j}^2 \pm \frac{h^3}{20}(k^2)_x\left(\frac{1}{6}h^2k_{i\pm1,j}^2 + \frac{2}{3}\right);$$

$$u_{i,j\pm1} : \frac{2}{3} + \frac{1}{90}h^2k_{i,j\pm1}^2 \pm \frac{h^3}{20}(k^2)_y\left(\frac{1}{6}h^2k_{i,j\pm1}^2 + \frac{2}{3}\right);$$

$$u_{i+1,j\pm1} : \frac{1}{6} + \frac{1}{90}h^2k_{i+1,j\pm1}^2 + \frac{h^3}{120}(k^2)_x \pm \frac{h^3}{120}(k^2)_y;$$

$$u_{i-1,j\pm1} : \frac{1}{6} + \frac{1}{90}h^2k_{i-1,j\pm1}^2 - \frac{h^3}{120}(k^2)_x \pm \frac{h^3}{120}(k^2)_y$$

3.4.3 Derivatives term in sixth-order compact scheme

As mentioned earlier, the sixth order scheme (3.24) and (3.25) contains terms such as derivatives of k^2 and f which needed to be calculated either analytically or computationally to second or fourth-order accuracy. This subsection will address how the derivatives term would be computed in the proposed algorithm.

Derivatives of right-hand side function, f

The right-hand side of the scheme (3.25) contains the first and mixed partial derivatives of f. However, they only require second-order accuracy so they could simply be estimated by using the first or second-order central difference. To deal with the fourth derivatives of f in

the right-hand side of the scheme, replace the function (k^2u) with f in Lemma 3.5. Then

$$\left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}\right)f=\frac{12}{h^2}\left[\Delta_h(f)-\Delta(f)\right]+\mathcal{O}(h^2)$$

$$\frac{h^4}{360}\left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}\right)f=\frac{h^2}{30}\left[\Delta_h(f)-\Delta(f)\right]+\mathcal{O}(h^6)$$

Unlike the term $\Delta(k^2u)$ in Lemma 3.5, there is no fourth-order approximation for the term Δf available yet. So we can first combine it with the other Δf term in (3.25). Now Equation (3.25) can be written as

$$\left(1-\frac{k^2h^2}{20}\right)f+\frac{h^4}{60}\left((k^2)_x\delta_x+(k^2)_y\delta_y+(k^2)_z\delta_z\right)f+\frac{h^2}{30}\Delta_h f+\frac{h^4}{90}\left(\delta_x^2\delta_y^2+\delta_x^2\delta_z^2+\delta_y^2\delta_z^2\right)f$$
$$+\frac{h^2}{20}\Delta f+\mathcal{O}(h^6)$$

The remaining derivative term of f yet to be computed is Δf which requires to be approximated up to fourth-order accuracy. It is possible to compute second derivatives of f up to fourth order accuracy by replacing the function u in Equation (3.7) with f. Specifically

$$\frac{\partial^2}{\partial x^2}f=A_x^{-1}\delta_x^2f+O(h^4)$$

$$\Delta f=A_x^{-1}\delta_x^2f+A_y^{-1}\delta_y^2f+A_z^{-1}\delta_z^2f+O(h^4)$$

where $A_\alpha=\left(1+\frac{h_\alpha^2}{12}\delta_\alpha^2\right)$ and $\alpha=x,y,z$. In practice, the numerical algorithm would only require the derivatives of f to be computed once at the start of the algorithm. In addition, the operator A_x, A_y, and A_z would form a tridiagonal or block tridiagonal matrices. Given the value $\frac{\partial^2}{\partial\alpha^2}f$ at the boundary of the computational domain, each of the system $A_\alpha(\frac{\partial^2}{\partial\alpha^2}f)=\delta_\alpha^2f$ can be solved directly via a method such as LU decomposition (or called LR decomposition in [25]). As such the numerical algorithm would only require values of the second-order derivatives of the right-hand side function f analytically at the boundary of the computational domain or impose some additional boundary conditions such as $\frac{\partial^2}{\partial\alpha^2}f=0$

on $\partial\Omega_h$ to the Helmholtz equation.

Derivatives of k^2

The sixth-order compact approximation scheme also requires the first order derivatives and $\Delta(k^2)$ either analytically or computationally up to fourth-order accuracy. $\Delta(k^2)$ can be computed in the same way as how the term Δf was calculated in the previous sub-subsection. The value of $\Delta(k^2)$ would then be stored for each iteration in the proposed iterative algorithm. For the first order derivatives of k^2, we have

$$\begin{aligned}
\delta_x(k^2) =&(k^2)_x + \frac{h^2}{6}\frac{\partial^3}{\partial x^3}(k^2) + \mathcal{O}(h^4) \\
\delta_x(k^2) =&(k^2)_x + \frac{h^2}{6}\left(\delta_x^2(k^2)_x + \mathcal{O}(h^2)\right) + \mathcal{O}(h^4) \\
\delta_x(k^2) =&\left(1 + \frac{h^2}{6}\delta_x^2\right)(k^2)_x + \mathcal{O}(h^4)
\end{aligned}$$

Let $B_x = \left(1 + \frac{h^2}{6}\delta_x^2\right)$ then

$$\frac{\partial u}{\partial x} = B_x^{-1}\delta_x u + \mathcal{O}(h^4)$$

The operator of B_x is similar to A_x and would form a tridiagonal matrix (block tridiagonal matrices for B_y and B_z). Hence it can also be solved with the LU decomposition. Overall this subsection proves that the sixth-order compact approximation scheme would only require the first and second derivatives of the coefficient function k^2 and the second derivatives of the function f at $\partial\Omega_h$. They could be computed analytically or by imposing additional boundary conditions to the Helmholtz equation. For a majority of the test problems in this dissertation, we will assume that the value can be computed analytically.

3.5 Fourth-Order Approximation Compact Scheme for 3D Convection-Diffusion Equation

In this section, the versatility of the proposed numerical solver is demonstrated with the two cases of the 3D convection-diffusion equation (2.10) where the first case consider the 3D convection-diffusion equation (2.10) with constant convection coefficient and the second case where the 3D convection-diffusion equation has a variable convection coefficient but only in the z-direction. In both cases, the computational domain is as defined in Equation (3.1).

3.5.1 Convection-diffusion equation with constant convection coefficients

Replaced the derivatives in the 3D convection-diffusion equation with the central differences of fourth-order accuracy yield

$$\begin{aligned}\Delta_h u - \frac{h_x^2}{12}\frac{\partial^4}{\partial x^4}u - \frac{h_y^2}{12}\frac{\partial^4}{\partial y^4}u - \frac{h_z^2}{12}\frac{\partial^4}{\partial z^4}u \\ + \alpha\left(\delta_x u - \frac{h_x^2}{6}\frac{\partial^3 u}{\partial x^3}\right) + \beta\left(\delta_y u - \frac{h_y^2}{6}\frac{\partial^3 u}{\partial y^3}\right) + \gamma\left(\delta_z u - \frac{h_z^2}{6}\frac{\partial^3 u}{\partial z^3}\right) = f(x,y,z) + \mathcal{O}(h^4)\end{aligned} \tag{3.26}$$

To obtain a fourth-order scheme, the third and fourth-order derivatives of u are required up to second-order accuracy. The standard fourth-order Padé approximation scheme is made harder to use for this problem due to the simultaneous existence of the third and fourth derivatives of u in Equation (3.26). Hence an alternative approach to derive the fourth-order compact scheme is considered. This approach is inspired by work done by Kalita et al in

[16] by extending their work on the two-dimensional convection-diffusion equation to the three-dimension case.

Differentiating the convection-diffusion equation with respect to x yield

$$\frac{\partial^3}{\partial x^3}u + \frac{\partial^3}{\partial x \partial y^2}u + \frac{\partial^3}{\partial x \partial z^2}u + \alpha\frac{\partial^2}{\partial x^2}u + \beta\frac{\partial^2}{\partial x \partial y}u + \gamma\frac{\partial^2}{\partial x \partial z}u = \frac{\partial}{\partial x}f(x,y,z)$$

The second order approximation for $\frac{\partial^3}{\partial x^3}u$ is now given as

$$\begin{aligned}\frac{\partial^3}{\partial x^3}u &= \frac{\partial}{\partial x}f(x,y,z) - \frac{\partial^3}{\partial x \partial y^2}u - \frac{\partial^3}{\partial x \partial z^2}u - \alpha\frac{\partial^2}{\partial x^2}u - \beta\frac{\partial^2}{\partial x \partial y}u - \gamma\frac{\partial^2}{\partial x \partial z} \\ &= \frac{\partial}{\partial x}f(x,y,z) - \delta_x\delta_y^2 u - \delta_x\delta_z^2 u - \alpha\delta_x^2 u - \beta\delta_x\delta_y u - \gamma\delta_x\delta_z u + \mathcal{O}(h^2)\end{aligned}$$

For the fourth derivatives of u, differentiating the convection-diffusion equation with respect to xx yield

$$\frac{\partial^4}{\partial x^4}u + \frac{\partial^4}{\partial x^2 \partial y^2}u + \frac{\partial^4}{\partial x^2 \partial z^2}u + \alpha\frac{\partial^3}{\partial x^3}u + \beta\frac{\partial^3}{\partial x^2 \partial y}u + \gamma\frac{\partial^3}{\partial x^2 \partial z}u = \frac{\partial^2}{\partial x^2}f(x,y,z)$$

So

$$\begin{aligned}\frac{\partial^4}{\partial x^4}u + \alpha\frac{\partial^3}{\partial x^3}u &= \frac{\partial^2}{\partial x^2}f(x,y,z) - \frac{\partial^4}{\partial x^2 \partial y^2}u - \frac{\partial^4}{\partial x^2 \partial z^2}u - \beta\frac{\partial^3}{\partial x^2 \partial y}u - \gamma\frac{\partial^3}{\partial x^2 \partial z}u \\ &= \frac{\partial^2}{\partial x^2}f(x,y,z) - \delta_x^2\delta_y^2 u - \delta_x^2\delta_z^2 u - \beta\delta_x^2\delta_y u - \gamma\delta_x^2\delta_z u\end{aligned}$$

Approximations for the derivatives with respect to y- and z-directions can be obtained similarly. Now (3.26) can be rewritten as

$$\begin{aligned}&(\Delta_h + \alpha\delta_x + \beta\delta_y + \gamma\delta_z)u - \frac{h_x^2}{12}\left(\frac{\partial^4}{\partial x^4} + \alpha\frac{\partial^3}{\partial x^3}\right)u - \frac{h_y^2}{12}\left(\frac{\partial^4}{\partial y^4} + \beta\frac{\partial^3}{\partial y^3}\right)u \\ &\quad - \frac{h_z^2}{12}\left(\frac{\partial^4}{\partial z^4} + \gamma\frac{\partial^3}{\partial z^3}\right)u - \alpha\frac{h_x^2}{12}\frac{\partial^3}{\partial x^3}u - \beta\frac{h_y^2}{12}\frac{\partial^3}{\partial y^3}u - \gamma\frac{h_z^2}{12}\frac{\partial^3}{\partial z^3}u \\ &\quad = f(x,y,z) + \mathcal{O}(h^4) \qquad (3.27)\end{aligned}$$

Substitute the approximation into the equation would give us

$$(\Delta_h + \alpha\delta_x + \beta\delta_y + \gamma\delta_z)u + \frac{h_x^2}{12}\left(\delta_x^2\delta_y^2 + \delta_x^2\delta_z^2 + \beta\delta_x^2\delta_y + \gamma\delta_x^2\delta_z\right)u$$

$$
\begin{aligned}
&+ \frac{h_y^2}{12}\left(\delta_x^2\delta_y^2 + \delta_y^2\delta_z^2 + \alpha\delta_x\delta_y^2 + \gamma\delta_y^2\delta_z\right)u + \frac{h_z^2}{12}\left(\delta_x^2\delta_z^2 + \delta_y^2\delta_z^2 + \alpha\delta_x\delta_z^2 + \beta\delta_y\delta_z^2\right)u \\
&+ \alpha\frac{h_x^2}{12}\left(\delta_x\delta_y^2 + \delta_x\delta_z^2 + \alpha\delta_x^2 + \beta\delta_x\delta_y + \gamma\delta_x\delta_z\right)u \\
&+ \beta\frac{h_y^2}{12}\left(\delta_x^2\delta_y + \delta_y\delta_z^2 + \beta\delta_y^2 + \alpha\delta_x\delta_y + \gamma\delta_y\delta_z\right)u \\
&- \gamma\frac{h_z^2}{12}\left(\delta_x^2\delta_z + \delta_y^2\delta_z + \gamma\delta_z^2 + \alpha\delta_x\delta_z + \beta\delta_y\delta_z\right)u \\
&= f + \left(\frac{h_x^2}{12}\frac{\partial^2}{\partial x^2} + \frac{h_y^2}{12}\frac{\partial^2}{\partial y^2} + \frac{h_z^2}{12}\frac{\partial^2}{\partial z^2}\right)f + \left(\alpha\frac{h_x^2}{12}\frac{\partial}{\partial x} + \beta\frac{h_y^2}{12}\frac{\partial}{\partial y} + \gamma\frac{h_z^2}{12}\frac{\partial}{\partial z}\right)f \\
&+ \mathcal{O}(h^4)
\end{aligned}
$$

The derivatives of f in the right-hand side of the equation can be approximated using the central differences with second-order accuracy. The matrix generated from the left-hand side of the equation can be presented in stencil notations with the following coefficients

$$u_{i,j,l} : -\frac{4}{3}\left(\frac{1}{h_x^2} + \frac{1}{h_y^2} + \frac{1}{h_z^2}\right) - \frac{1}{6}\left(\alpha^2 + \beta^2 + \gamma^2\right);$$

$$u_{i\pm1,j,l} : \frac{1}{6}\left(\frac{4}{h_x^2} - \frac{1}{h_y^2} - \frac{1}{h_z^2}\right) \pm \frac{\alpha}{12}\left(\frac{4}{h_x} - \frac{h_x}{h_y^2} - \frac{h_x}{h_z^2} \pm \alpha\right);$$

$$u_{i,j\pm1,l} : \frac{1}{6}\left(\frac{4}{h_y^2} - \frac{1}{h_x^2} - \frac{1}{h_z^2}\right) \pm \frac{\beta}{12}\left(\frac{4}{h_y} - \frac{h_y}{h_x^2} - \frac{h_y}{h_z^2} \pm \beta\right);$$

$$u_{i,j,l\pm1} : \frac{1}{6}\left(\frac{4}{h_z^2} - \frac{1}{h_x^2} - \frac{1}{h_y^2}\right) \pm \frac{\gamma}{12}\left(\frac{4}{h_z} - \frac{h_z}{h_x^2} - \frac{h_z}{h_y^2} \pm \gamma\right);$$

$$u_{i\pm1,j+1,l} : \frac{1}{12}\left(\frac{1}{h_x^2} + \frac{1}{h_y^2} \pm \frac{\alpha h_x}{2h_y^2} \pm \frac{\alpha}{2h_x}\right) + \frac{\beta}{24}\left(\frac{1}{h_y} + \frac{hy}{hx^2} \pm \frac{\alpha h_y}{2h_x} \pm \frac{\alpha h_x}{2h_y}\right);$$

$$u_{i\pm1,j-1,l} : \frac{1}{12}\left(\frac{1}{h_x^2} + \frac{1}{h_y^2} \pm \frac{\alpha h_x}{2h_y^2} \pm \frac{\alpha}{2h_x}\right) - \frac{\beta}{24}\left(\frac{1}{h_y} + \frac{hy}{hx^2} \pm \frac{\alpha h_y}{2h_x} \pm \frac{\alpha h_x}{2h_y}\right);$$

$$u_{i\pm1,j,l+1} : \frac{1}{12}\left(\frac{1}{h_x^2} + \frac{1}{h_z^2} \pm \frac{\alpha h_x}{2h_z^2} \pm \frac{\alpha}{2h_x}\right) + \frac{\gamma}{24}\left(\frac{1}{h_z} + \frac{h_z}{h_x^2} \pm \frac{\alpha h_z}{2h_x} \pm \frac{\alpha h_x}{2h_z}\right);$$

$$u_{i\pm1,j,l-1} : \frac{1}{12}\left(\frac{1}{h_x^2}+\frac{1}{h_z^2}\pm\frac{\alpha h_x}{2h_z^2}\pm\frac{\alpha}{2h_x}\right)-\frac{\gamma}{24}\left(\frac{1}{h_z}+\frac{h_z}{h_x^2}\pm\frac{\alpha h_z}{2h_x}\pm\frac{\alpha h_x}{2h_z}\right);$$

$$u_{i,j\pm1,l+1} : \frac{1}{12}\left(\frac{1}{h_z^2}+\frac{1}{h_y^2}\pm\frac{\beta h_y}{2h_z^2}\pm\frac{\beta}{2h_y}\right)+\frac{\gamma}{24}\left(\frac{1}{h_z}+\frac{h_z}{h_y^2}\pm\frac{\beta h_z}{2h_y}\pm\frac{\beta h_y}{2h_z}\right);$$

$$u_{i,j\pm1,l-1} : \frac{1}{12}\left(\frac{1}{h_z^2}+\frac{1}{h_y^2}\pm\frac{\beta h_y}{2h_z^2}\pm\frac{\beta}{2h_y}\right)-\frac{\gamma}{24}\left(\frac{1}{h_z}+\frac{h_z}{h_y^2}\pm\frac{\beta h_z}{2h_y}\pm\frac{\beta h_y}{2h_z}\right);$$

$$u_{i\pm1,j\pm1,l\pm1} : 0$$

3.5.2 Convection-diffusion equation with variable convection coefficient in z-direction

In this subsection, the discretization for the 3D convection-diffusion equation in the form of Equation (2.11) is presented. For the discretization of 3D general convection-diffusion equations with variable coefficients, Ananthakrishnaiah et al. [30] proposed a procedure of developing fourth-order compact finite difference schemes using a lot of pencil and paper analysis. The formulas given in [30] are however too implicit and abstract that it would require too much time and effort to derive an explicit scheme. In [19] by employing the computer algebra package Mathematica, Zhang was able to derive an explicit fourth-order scheme for a 3D convection-diffusion equation based on the general implicit formulas from [30].

The author in [19] also assumes that the discretization is done on a uniform grid with mesh size h. A simplified version of the fourth order compact finite difference scheme derived in

[19] for 3D convection-diffusion equation (2.11) can be presented as

$$u_{i,j,l} : -[24 + h^2\gamma_{i,j,l}^2 + h(\gamma_{i,j,l+1} - \gamma_{i,j,l-1})];$$

$$u_{i,j,l+1} : 2 - \frac{h}{4}(2\gamma_{i,j,l} - \gamma_{i+1,j,l} - \gamma_{i-1,j,l} - \gamma_{i,j+1,l} - \gamma_{i,j-1,l} - 3\gamma_{i,j,l+1} + \gamma_{i,j,l-1})$$

$$+ \frac{h}{8}\gamma_{i,j,l}(4\gamma_{i,j,l} + \gamma_{i,j,l+1} - \gamma_{i,j,l-1});$$

$$u_{i,j,l-1} : 2 + \frac{h}{4}(2\gamma_{i,j,l} - \gamma_{i+1,j,l} - \gamma_{i-1,j,l} - \gamma_{i,j+1,l} - \gamma_{i,j-1,l} + \gamma_{i,j,l+1} - 3\gamma_{i,j,l-1})$$

$$+ \frac{h}{8}\gamma_{i,j,l}(4\gamma_{i,j,l} - \gamma_{i,j,l+1} + \gamma_{i,j,l-1});$$

$$u_{i\pm1,j,l} = u_{i,j\pm1,l} : 2;$$

$$u_{i\pm1,j\pm1,l} : 1;$$

$$u_{i\pm1,j,l+1} : 1 + \frac{h}{2}\gamma_{i,j,l} \pm \frac{h}{8}(\gamma_{i+1,j,l} - \gamma_{i-1,j,l});$$

$$u_{i\pm1,j,l-1} : 1 - \frac{h}{2}\gamma_{i,j,l} \pm \frac{h}{8}(\gamma_{i+1,j,l} - \gamma_{i-1,j,l});$$

$$u_{i,j\pm1,l+1} : 1 + \frac{h}{2}\gamma_{i,j,l} \pm \frac{h}{8}(\gamma_{i,j+1,l} - \gamma_{i,j-1,l});$$

$$u_{i,j\pm1,l-1} : 1 - \frac{h}{2}\gamma_{i,j,l} \pm \frac{h}{8}(\gamma_{i,j+1,l} - \gamma_{i-1,j-1,l});$$

$$u_{i\pm1,j\pm1,l\pm1} : 0$$

and the right-hand side is given by

$$\frac{h^2}{2}(6f_{i,j,l} + f_{i+1,j,l} + f_{i-1,j,l} + f_{i,j+1,l} + f_{i,j-1,l} + f_{i,j,l+1} + f_{i,j,l-1}) + \frac{h^3}{4}\gamma_{i,j,l}(f_{i,j,l+1} - f_{i,j,l-1})$$

3.6 Boundary Condition

For the finite domain problem, the discretization shown in this chapter is incomplete. For instance, when $i = 1$, then one can express the second-order approximation compact scheme of the Helmholtz equation in Equation (3.5) explicitly as

$$\frac{u_{2,j,l} - 2u_{1,j,l} + u_{0,j,l}}{h_x^2} + \frac{u_{1,j+1,l} - 2u_{1,j,l} + u_{1,j-1,l}}{h_y^2} + \frac{u_{1,j,l+1} - 2u_{1,j,l} + u_{1,j,l-1}}{h_z^2} + k_{1,j,l}^2 u_{1,j,l} = f_{1,j,l} + \mathcal{O}(h_x^2)$$

The term $u_{0,j,l}$ is called a ghost point and is the value of the numerical solution u at the boundary of the computational domain, $\partial\Omega_h$. When the Dirichlet boundary condition is imposed then the value of these ghost points are known, say $u_{0,j,l} = v_{j,l}$, and no approximation is required. In practice, the ghost points would then be moved to the right-hand side of the equation as follows

$$\frac{u_{2,j,l} - 2u_{1,j,l}}{h_x^2} + \frac{u_{1,j+1,l} - 2u_{1,j,l} + u_{1,j-1,l}}{h_y^2} + \frac{u_{1,j,l+1} - 2u_{1,j,l} + u_{1,j,l-1}}{h_z^2} + k_{1,j,l}^2 u_{1,j,l} = f_{1,j,l} - \frac{v_{j,l}}{h_x^2}$$

For the subsurface inclusion model problem, the Sommerfeld radiation conditions are imposed when the unbounded domain is truncated to a finite domain. The simplified Sommerfeld radiation condition for the boundaries in the x-direction has the form

$$\frac{\partial}{\partial x} u_{1,j,l} - ik_{1,j,l} u_{1,j,l} = 0$$
$$\frac{\partial}{\partial x} u_{N_x,j,l} + ik_{N_x,j,l} u_{N_x,j,l} = 0$$

where i is the imaginary unit. To avoid confusion consider i strictly as $i = \sqrt{-1}$ and not an index; ι will used for indexing for this section.

The Sommerfeld radiation boundary conditions only differ only by the sign of iku on both ends of the computational domain, which is true for all spatial directions. For the subsurface inclusion model considered in this dissertation, one may assume that $k_{\iota,j,l}$ is a constant, say k, near the boundaries in the x-direction. The assumption is valid as the inclusion will not be near the boundary so the function $k_{\iota,j,l}$ would be a step function depending on the variable z on boundaries in both x- and y-directions, and a constant function on boundaries in z-direction.

The Sommerfeld radiation conditions would allow one to approximate the ghost points with elements of the solution vector that are not on the boundary. To maintain the order of the schemes these ghost points require higher-order approximation by at least two due to the division of h_x^2, h_y^2, or h_z^2 presented in all compact schemes for Helmholtz equations. Therefore, the fourth-order approximations of the boundary conditions are given in this section for the second-order scheme of the Helmholtz equation.

The fourth-order accuracy for the left boundary in x-direction is given by

$$
\begin{aligned}
0 &= \frac{\partial}{\partial x} u_{1,j,l} - iku_{1,j,l} \\
&= \delta_x u_{1,j,l} - \frac{h_x^2}{6}\frac{\partial^2}{\partial x^2}\left(\frac{\partial}{\partial x} u_{1,j,l}\right) - iku_{1,j,l} + \mathcal{O}(h_x^4) \\
&= \delta_x u_{1,j,l} - \frac{h_x^2}{6}\frac{\partial^2}{\partial x^2}(iku_{1,j,l}) - iku_{1,j,l} + \mathcal{O}(h_x^4) \\
&= 6\delta_x u_{1,j,l} - h_x^2\delta_x^2(iku_{1,j,l}) - 6iku_{1,j,l} + \mathcal{O}(h_x^4) \\
&= 6h_x\delta_x u_{1,j,l} - ih_x k(h_x^2\delta_x^2 u_{1,j,l}) - 6ih_x k u_{1,j,l} + \mathcal{O}(h_x^5) \\
&= (3 - ih_x k)u_{2,j,l} - 4ih_x k u_{1,j,l} - (3 + ih_x k)u_{0,j,l} + \mathcal{O}(h_x^5)
\end{aligned}
$$

Thus $u_{0,j,l}$ can be expressed as

$$u_{0,j,l} = \frac{-4ih_xk}{3+ih_xk}u_{1,j,l} + \frac{3-ih_xk}{3+ih_xk}u_{2,j,l} + \mathcal{O}(h_x^4)$$

This is combined with the second-order discretization of the Helmholtz equation at the $(1,j,l)$-grid point to eliminate all $u_{0,j,l}$ terms. Similar derivation can be used to eliminate $u_{\iota,0,l}$ and $u_{\iota,j,0}$ terms. For the right boundary in x-direction

$$\begin{aligned}
0 &= \frac{\partial}{\partial x}u_{N_x,j,l} + iku_{N_x,j,l} \\
&= \delta_x u_{N_x,j,l} - \frac{h_x^2}{6}\frac{\partial^2}{\partial x^2}\left(\frac{\partial}{\partial x}u_{N_x,j,l}\right) + iku_{N_x,j,l} + \mathcal{O}(h_x^4) \\
&= \delta_x u_{N_x,j,l} - \frac{h_x^2}{6}\frac{\partial^2}{\partial x^2}(-iku_{N_x,j,l}) + iku_{N_x,j,l} + \mathcal{O}(h_x^4) \\
&= 6\delta_x u_{N_x,j,l} + h_x^2\delta_x^2(iku_{N_x,j,l}) + 6iku_{N_x,j,l} + \mathcal{O}(h_x^4) \\
&= 6h_x\delta_x u_{N_x,j,l} + ih_xk(h_x^2\delta_x^2 u_{N_x,j,l}) + 6ih_xku_{N_x,j,l} + \mathcal{O}(h_x^5) \\
&= (3+ih_xk)u_{N_x+1,j,l} + 4ih_xku_{N_x,j,l} - (3-ih_xk)u_{N_x-1,j,l} + \mathcal{O}(h_x^5)
\end{aligned}$$

Thus $u_{N_x+1,j,l}$ can be expressed as

$$u_{N_x+1,j,l} = \frac{-4ih_xk}{3+ih_xk}u_{N_x,j,l} + \frac{3-ih_xk}{3+ih_xk}u_{N_x-1,j,l} + \mathcal{O}(h_x^4)$$

The terms $u_{\iota,N_y+1,l}$ and u_{ι,j,N_z+1} are found similarly. This will complete the second-order discretization of the Helmholtz equation with the Sommerfield radiation boundary condition.

3.6.1 Higher-order approximation scheme for Sommerfield radiation boundary points

Both the fourth and sixth-order scheme for the Helmholtz equation has more additional ghost points compared to the second-order scheme for the Helmholtz equation such as $u_{0,j,0}$

and $u_{0,0,l}$. The values of $u_{0,j,0}$ can first be approximated by using $u_{0,j,1}$ and $u_{0,j,2}$. Then one can approximated $u_{0,j,1}$ with $u_{1,j,1}$ and $u_{2,j,1}$ (similarly for $u_{0,j,2}$ with $u_{1,j,2}$ and $u_{2,j,2}$). These points would require accuracy at least four higher than the compact scheme due to the division of h_x^2, h_y^2, and h_z^2 twice. It is worth mentioning that the eight ghost points such as $u_{0,0,0}$, $u_{N_x,0,0}$, $u_{0,N_y,0}$, $u_{N_x,N_y,0}$, $u_{0,0,N_z}$, $u_{N_x,0,N_z}$, u_{0,N_y,N_z}, and u_{N_x,N_y,N_z} are not divided by the step size in both fourth and sixth-order scheme. As such $u_{0,0,0}$ can be approximated by $u_{1,0,0}$ and $u_{2,0,0}$ which are subsequently approximated by $u_{1,1,0}$, $u_{1,2,0}$, $u_{2,1,0}$, and $u_{2,2,0}$ and lastly with $u_{1,1,1}$, $u_{1,1,2}$, $u_{1,2,1}$, $u_{1,2,2}$, $u_{2,1,1}$, $u_{2,1,2}$, $u_{2,2,1}$ and $u_{2,2,2}$. The higher-order approximations of the boundary conditions are given in this section for the fourth and sixth-order compact scheme of the Helmholtz equation.

In general, one may approximate the first-order derivative with respect to x by

$$\begin{aligned}\frac{u_{\iota+1,j,l}-u_{\iota-1,j,l}}{2h_x} &= \frac{\partial}{\partial x}u_{\iota,j,l}+\frac{h_x^2}{6}\frac{\partial^3}{\partial x^3}u_{\iota,j,l}+\frac{h_x^4}{120}\frac{\partial^5}{\partial x^5}u_{\iota,j,l}+\frac{h_x^6}{7!}\frac{\partial^7}{\partial x^7}u_{\iota,j,l}+\mathcal{O}(h_x^8)\\ &= \sum_{p=1}^{n}\frac{h_x^{2p-2}}{(2p-1)!}\frac{\partial^{2p-1}}{\partial x^{2p-1}}u_{\iota,j,l}+\mathcal{O}(h_x^{2p})\end{aligned}$$

Hence at the boundaries on x-direction, we have

$$\begin{aligned}u_{0,j,l} &= u_{2,j,l}-2\sum_{p=1}^{n}\frac{h_x^{2p-1}}{(2p-1)!}\frac{\partial^{2p-1}}{\partial x^{2p-1}}u_{1,j,l}+\mathcal{O}(h_x^{2p+1})\\ u_{N_x+1,j,l} &= u_{N_x-1,j,l}+2\sum_{p=1}^{n}\frac{h_x^{2p-1}}{(2p-1)!}\frac{\partial^{2p-1}}{\partial x^{2p-1}}u_{N_x,j,l}+\mathcal{O}(h_x^{2p+1})\end{aligned} \tag{3.28}$$

From the Sommerfield radiation boundary condition, one may derive

$$\frac{\partial^3}{\partial x^3}u_{1,j,l}=\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}u_{1,j,l}\right)\right)=(ik)^3u_{1,j,l}=-ik^3u_{1,j,l}$$

In general, we have $\frac{\partial^n}{\partial x^n}u_{1,j,l}=(ik)^nu_{1,j,l}$ and similarly $\frac{\partial^n}{\partial x^n}u_{N_x,j,l}=(-ik)^nu_{N_x,j,l}$. Substituting these expressions into Equation (3.28), the ninth-order accuracy scheme is given by

$$u_{0,j,l}=u_{2,j,l}-2i\left[h_xk-\frac{(h_xk)^3}{3!}+\frac{(h_xk)^5}{5!}-\frac{(h_xk)^7}{7!}\right]u_{1,j,l}+\mathcal{O}(h_x^9)$$

$$u_{N_x+1,j,l} = u_{N_x-1,j,l} - 2i\left[h_x k - \frac{(h_x k)^3}{3!} + \frac{(h_x k)^5}{5!} - \frac{(h_x k)^7}{7!}\right] u_{N_x,j,l} + \mathcal{O}(h_x^9)$$

and the eleventh-order accuracy scheme is given by

$$u_{0,j,l} = u_{2,j,l} - 2i\left[h_x k - \frac{(h_x k)^3}{3!} + \frac{(h_x k)^5}{5!} - \frac{(h_x k)^7}{7!} + \frac{(h_x k)^9}{9!}\right] u_{1,j,l} + \mathcal{O}(h_x^{11})$$

$$u_{N_x+1,j,l} = u_{N_x-1,j,l} - 2i\left[h_x k - \frac{(h_x k)^3}{3!} + \frac{(h_x k)^5}{5!} - \frac{(h_x k)^7}{7!} + \frac{(h_x k)^9}{9!}\right] u_{N_x,j,l} + \mathcal{O}(h_x^{11})$$

Overall, using the Maclaurin series for sine function, the accuracy scheme is given by

$$u_{0,j,l} = u_{2,j,l} - 2i\sum_{p=1}^{n} \frac{(h_x k)^{2p-1}}{(2p-1)!} u_{1,j,l} + \mathcal{O}(h_x^{2p+1}) = u_{2,j,l} - 2i\sin(h_x k) u_{1,j,l}$$

$$u_{N_x+1,j,l} = u_{N_x-1,j,l} - 2i\sum_{p=1}^{n} \frac{(h_x k)^{2p-1}}{(2p-1)!} u_{N_x,j,l} + \mathcal{O}(h_x^{2p+1}) = u_{N_x-1,j,l} - 2i\sin(h_x k) u_{N_x,j,l}$$

The approximation scheme for y- and z-directions can be derived similarly.

4 Direct FFT Solver

4.1 Introduction

In this chapter, an efficient parallel direct solver called Direct FFT Solver will be presented. This direct solver was developed by the authors and other researchers based on a combination of the separation of variables technique and FFT type method [37]. The direct FFT solver is designed to solve a system of linear equation

$$A\vec{x} = \vec{b}$$

with certain restrictions imposed on the matrix A. The idea behind this direct solver is to transform the coefficient matrix A into a tridiagonal matrix and solve the resulting matrix with LU decomposition. As such the matrix A is required to have certain symmetry to ensure it can be transformed into a tridiagonal or block tridiagonal matrix. In particular, to solve for the numerical solution of PDE, the coefficient of the stencil obtained from the discretization of the PDE are required to attain certain aspect of symmetry which will be shown shortly in the next section.

While the proposed numerical solver is restricted to a class of linear with specific restrictions, the direct solver has demonstrated highly accurate and scalable results for solving PDE that meets the criteria [37]. It would also be used as the preconditioner solver for the proposed iterative method.

4.2 Compact Stencil for Direct FFT Solver

The direct FFT solver is designed to solve PDE whose numerical scheme at every grid point (i, j, l) can be presented systematically as

$$\begin{aligned}\Sigma_{\nu=l-1}^{\nu=l+1} \big(a_\nu \left[\vec{u}_{i-1,j-1,\nu} + \vec{u}_{i-1,j+1,\nu} + \vec{u}_{i+1,j-1,\nu} + \vec{u}_{i+1,j+1,\nu}\right] \\ + b_\nu \left[\vec{u}_{i-1,j,\nu} + \vec{u}_{i+1,j,\nu}\right] + c_\nu \left[\vec{u}_{i,j-1,\nu} + \vec{u}_{i,j+1,\nu}\right] \\ + d_\nu \vec{u}_{i,j,\nu}\big) = \vec{f}_{i,j,l}. \end{aligned} \tag{4.1}$$

This equation corresponds to the $(i + j \cdot N_x + l \cdot N_x \cdot N_y)$-th row in the resulting linear system $A\vec{u} = \vec{f}$ where the vectors $\vec{u}, \vec{f} \in \mathbb{C}^{N_x \cdot N_y \cdot N_z}$ are such that $\vec{u}$ is the solution vector and $\vec{f}$ is right hand side vector. Figure 4.1 illustrates an example of a 27-point stencil that would satisfy Equation (4.1) for the three-dimensional case. A grid with the same color is required to have the same coefficient. Moreover, the values of these coefficients have to be constant at every $N_x \cdot N_y$ layer.

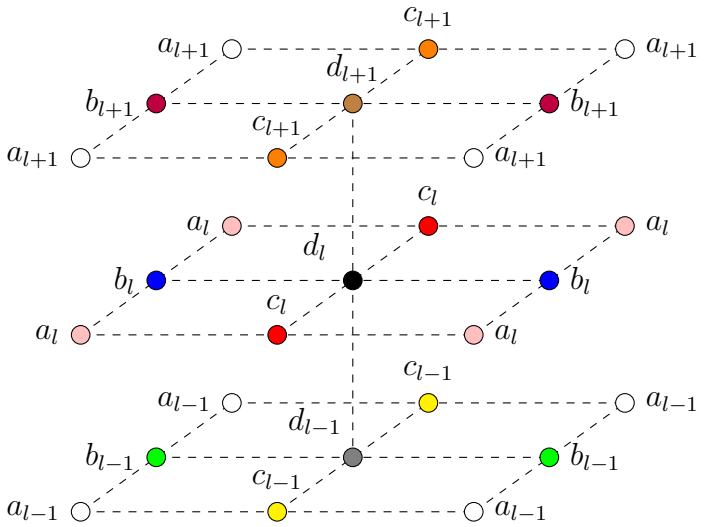

Figure 4.1: 27-point stencil operator for preconditioner system

In the case of the 3D Helmholtz equation, this symmetry can be achieved when the coefficient function k^2 depends only on the variable z. This also implies that the direct FFT solver is capable of solving any 3D Poisson equation directly. For the 3D convection-diffusion equation with constant coefficient, if two of the convection coefficients, namely α and β, would be zero then the resulting stencil from discretization would also satisfy these constraints.

The updated stencil coefficients for the specific 3D Helmholtz and 3D convection-diffusion equation that satisfy this restriction will be presented. Both sides of the equation for the second and fourth order scheme (3.5, 3.10) will be multiplied by h_z^2 as a way to reduce the computational cost by reducing the number of divisions. Denote the term $Rzx = h_z^2/h_x^2$ and $Rzy = h_z^2/h_y^2$ for the rest of this section. Additionally, the index notation would be replaced to match the label shown in Figure 4.1.

4.2.1 Second-order scheme for 3D Helmholtz equation

The second-order compact scheme for the modified equation can be obtained by replacing the coefficient function $k^2(x, y, z)$ with some coefficient function $k_0^2(z)$. Then the newly updated coefficient for the stencil can be presented as

$$d_l : h_z^2 k_l^2 - 2\left(Rzx + Rzy + 1\right); \quad b_l : Rzx; \quad c_l : Rzy; \quad d_{l\pm1} : 1;$$

$$a_l = a_{l\pm1} = b_{l\pm1} = c_{l\pm1} : 0$$

where $k_l^2 = k_0^2(z_l)$.

4.2.2 Fourth-order scheme for 3D Helmholtz equation

Similar to the second-order scheme, the coefficient of the stencil for the fourth-order scheme is obtained by replacing the coefficient function. In this case, we have

$$a_l : \frac{1}{12}\left(Rzx + Rzy\right);$$

$$b_l : \frac{h_z^2 k_l^2}{12} + \frac{2}{3}Rzx - \frac{1}{6}\left(Rzy + 1\right);$$

$$c_l : \frac{h_z^2 k_l^2}{12} + \frac{2}{3}Rzy - \frac{1}{6}\left(Rzx + 1\right);$$

$$d_l : \frac{h_z^2 k_l^2}{2} - \frac{4}{3}\left(Rzx + Rzy + 1\right);$$

$$a_{l\pm1} : 0\ ;$$

$$b_{l\pm1} : \frac{1}{12}\left(Rzx + 1\right);$$

$$c_{l\pm1} : \frac{1}{12}\left(Rzy + 1\right);$$

$$d_{l\pm1} : \frac{h_z^2 k_{l\pm1}^2}{12} + \frac{2}{3} - \frac{1}{6}\left(Rzx + Rzy\right)$$

4.2.3 Sixth-order scheme for 3D Helmholtz equation

Unlike the previous compact scheme, there are significant changes in the sixth-order compact scheme. Since the coefficient function k^2 now depends only in the z-direction, the derivatives of k^2 in x- and y-directions are now zero and Equation (3.24, 3.25) can be simplify as

$$\Delta_h U_{30} + \frac{h^2}{6}(\delta_x^2\delta_y^2 + \delta_x^2\delta_z^2 + \delta_y^2\delta_z^2)U_{15} + \frac{h^2}{10}(k^2)_z\delta_z U_6$$

$$+\frac{(k^2)_z h^4}{60}\left(\delta_{xxz}+\delta_{yyz}\right)u+\frac{h^4}{30}\delta_x^2\delta_y^2\delta_z^2 u+\frac{h^2}{20}[(k^2)_{zz}-k^4]u+k^2u$$

$$=\left(1-\frac{k^2h^2}{20}\right)f+\frac{h^4}{60}(k^2)_z\frac{\partial}{\partial z}f+\frac{h^2}{12}\Delta f+\frac{h^4}{360}\left(\frac{\partial^4}{\partial x^4}+\frac{\partial^4}{\partial y^4}+\frac{\partial^4}{\partial z^4}\right)f$$

$$+\frac{h^4}{90}\left(\frac{\partial^4}{\partial x^2\partial y^2}+\frac{\partial^4}{\partial x^2\partial z^2}+\frac{\partial^4}{\partial y^2\partial z^2}\right)f$$

where $(k^2)_{zz}=\frac{d^2}{dz^2}(k^2)$. Multiply both sides of equations by h^2 as before, the sixth-order scheme stencil coefficient can be simplified into

$$a_l:\frac{1}{10}+\frac{1}{90}h^2k_l^2;$$

$$b_l=c_l:\frac{7}{15}-\frac{1}{90}h^2k_l^2;$$

$$d_l:-\frac{64}{15}+\frac{14}{15}h^2k_l^2+\frac{h^4}{20}\left[(k^2)_{zz}-k_l^4\right];$$

$$a_{l\pm1}:\frac{1}{30};$$

$$b_{l\pm1}=c_{l\pm1}:\frac{1}{10}+\frac{1}{90}h^2k_{l\pm1}^2\pm\frac{h^3}{120}(k^2)_z;$$

$$d_{l\pm1}:\frac{7}{15}-\frac{1}{90}h^2k_{l\pm1}^2\pm\frac{h^3}{60}(k^2)_z\left(1+\frac{1}{2}h^2k_{l\pm1}^2\right)$$

The derivatives $(k^2)_z$ and $(k^2)_{zz}$ are always evaluated at the center point z_l.

4.2.4 Fourth-order scheme for 3D convection-diffusion equation with constant coefficients

Let $\alpha = \beta = 0$ for the 3D convection-diffusion equation. Then both sides of the equation are multiplied by h_z^2 and we have the following coefficients

$$a_l : \frac{1}{12}\left(Rzx + Rzy\right);$$

$$b_l : \frac{1}{6}\left(4Rzx - Rzy - 1\right);$$

$$c_l : \frac{1}{6}\left(4Rzy - Rzx - 1\right);$$

$$d_l : -\frac{4}{3}\left(Rzx + Rzy + 1\right) - \frac{h_z^2}{6}\gamma^2;$$

$$a_{l\pm1} : 0$$

$$b_{l\pm1} : \left(\frac{1}{12} \pm \frac{h_z\gamma}{24}\right)\left(1 + Rzx\right);$$

$$c_{l\pm1} : \left(\frac{1}{12} \pm \frac{h_z\gamma}{24}\right)\left(1 + Rzy\right);$$

$$d_{l\pm1} : \frac{1}{6}\left(4 - Rzx - Rzy\right) \pm \frac{h_z\gamma}{12}\left(4 - Rzx - Rzy \pm h_z\gamma\right);$$

Since $\alpha = \beta = 0$, one may modified the coefficients b_l, c_l, and d_l slightly as follows

$$b_l : \frac{1}{6}\left(4Rzx - Rzy - 1\right) + \frac{h_z^2}{12}\alpha^2;$$

$$c_l : \frac{1}{6}\left(4Rzy - Rzx - 1\right) + \frac{h_z^2}{12}\beta^2;$$

$$d_l : -\frac{4}{3}\left(Rzx + Rzy + 1\right) - \frac{h_z^2}{6}(\alpha^2 + \beta^2 + \gamma^2);$$

Adding these terms would modify the coefficient to be closer to the 3D convection-diffusion equations with nonzero convection coefficients in subsection 3.5.1, which is extremely important for designing the preconditioner matrix for the iterative method.

4.2.5 Fourth-order scheme for 3D convection-diffusion equation with variable coefficient in z-direction

In this case, we consider a similar approach to the 3D Helmholtz equation and modified the coefficient so that it depends only on the variable z, that is $\gamma(x, y, z)$ is replaced by $\gamma_0(z)$. The coefficient of the stencil is then given by

$$
\begin{aligned}
a_l &: 1; \\
b_l = c_l &: 2; \\
d_l &: -[24 + h^2\gamma_l^2 + h(\gamma_{l+1} - \gamma_{l-1})] \\
a_{l\pm1} &: 0 \\
b_{l\pm1} = c_{l\pm1} &: 1 \pm \frac{h}{2}\gamma_l; \\
d_{l+1} &: 2 - \frac{h}{4}(-2\gamma_l - 3\gamma_{l+1} + \gamma_{l-1}) + \frac{h^2}{8}\gamma_l(4\gamma_l + \gamma_{l+1} - \gamma_{l-1}); \\
d_{l-1} &: 2 + \frac{h}{4}(-2\gamma_l + \gamma_{l+1} - 3\gamma_{l-1}) + \frac{h^2}{8}\gamma_l(4\gamma_l - \gamma_{l+1} + \gamma_{l-1});
\end{aligned}
$$

where $\gamma_l = \gamma_0(z_l)$. It is worth noting that if γ is a constant function, then this scheme is exactly a multiple of the fourth-order scheme for the convection-diffusion equation with constant coefficients before the modification by a factor of 6, assuming the grid size is equal.

4.3 Direct FFT Solver

Now the numerical scheme (4.1) can be presented in block tridiagonal form as

$$\begin{aligned} C_1\vec{u}_1 + C_1^p\vec{u}_2 &= \vec{f}_1, \\ C_l^m\vec{u}_{l-1} + C_l\vec{u}_l + C_l^p\vec{u}_{l+1} &= \vec{f}_l; \qquad \text{for } l = 2, \ldots, N_z - 1, \\ C_{N_z}^m\vec{u}_{N_z-1} + C_{N_z}\vec{u}_{N_z} &= \vec{f}_{N_z}. \end{aligned} \tag{4.2}$$

Here, the vectors $\vec{u}_l$ and $\vec{f}_l$ are the parts of the solution vector $\vec{u}$ and the right hand side $\vec{f}$ of size $N_x \cdot N_y$, where $l = 1, ..., N_z$. The block tridiagonal matrices C_l^m, C_l, and C_l^p are defined by the coefficients in (4.1) based on the order of the scheme.

The goal for the next step is to diagonalize all the block tridiagonal matrices C_l^m, C_l, and C_l^p. This would transform the system into a block tridiagonal system that can be solved easily with a method such as LU decomposition.

4.3.1 Eigenvalues and eigenvectors

In this subsection, we will find the eigenvalues and eigenvectors (eigenpairs) for the matrices C_l^m, C_l, and C_l^p in Theorem 4.1 so that they can be diagonalized in the next subsection.

Theorem 4.1. *The $(N_x \cdot N_y) \times (N_x \cdot N_y)$ matrix of eigenvectors V of C_l^m, C_l, and C_l^p is defined by*

$$V = \left[\boldsymbol{v}^{1,1}\ \boldsymbol{v}^{2,1}\ \cdots\ \boldsymbol{v}^{N_x,1}\ \boldsymbol{v}^{1,2}\ \boldsymbol{v}^{2,2}\ \cdots\ \boldsymbol{v}^{N_x,N_y}\right]$$

where and $\boldsymbol{v}^{m,n}$ are a vector of size $(N_x \cdot N_y)$, $m = 1, \cdots, N_x$ and $n = 1, \cdots, N_y$ and its

components is given by

$$v_{i,j}^{m,n} = \frac{2}{\sqrt{(N_x+1)(N_y+1)}} \sin\left(\frac{\pi m i}{N_x+1}\right) \sin\left(\frac{\pi n j}{N_y+1}\right),$$

where $i = 1, \cdots, N_x$ *and* $j = 1, \cdots, N_y$. *The corresponding eigenvalues* $\lambda_\nu^{m,n}$, *for the matrices* $C_l^m, C_l,$ *and* C_l^p, *defined by*

$$\lambda_\nu^{m,n} = 4a_\nu \cos\left(\frac{m\pi}{N_x+1}\right) \cos\left(\frac{n\pi}{N_y+1}\right) + 2b_\nu \cos\left(\frac{m\pi}{N_x+1}\right) + 2c_\nu \cos\left(\frac{n\pi}{N_y+1}\right) + d_\nu,$$

where $\nu \in \{l-1, l, l+1\}$

Proof. Let $\mathbf{v}^{m,n}$ be as define in Theorem 4.1. First, consider the trigonometric addition theorem,

$$\frac{1}{2}\sin\left(\frac{\pi m(i-1)}{N_x+1}\right) + \frac{1}{2}\sin\left(\frac{\pi m(i+1)}{N_x+1}\right) = \cos\left(\frac{\pi m}{N_x+1}\right)\sin\left(\frac{\pi m i}{N_x+1}\right)$$

Observe that by multiplying both side of the equations with $2b_\nu \frac{2}{\sqrt{(N_x+1)(N_y+1)}} \sin\left(\frac{\pi n j}{N_y+1}\right)$ we would get

$$b_\nu\left(v_{i-1,j}^{m,n} + v_{i+1,j}^{m,n}\right) = 2b_\nu \cos\left(\frac{\pi m}{N_x+1}\right) v_{i,j}^{m,n} \tag{4.3}$$

Notice that the right-hand side of the equation minus the $v_{i,j}^{m,n}$ coincides with the second term in the definition of $\lambda_\nu^{m,n}$ in Theorem 4.1. Similarly replacing some index in the trigonometric addition theorem yield

$$\frac{1}{2}\sin\left(\frac{\pi n(j-1)}{N_y+1}\right) + \frac{1}{2}\sin\left(\frac{\pi n(j+1)}{N_y+1}\right) = \cos\left(\frac{\pi n}{N_y+1}\right)\sin\left(\frac{\pi n j}{N_y+1}\right)$$

Multiplying both side of the equations with $2c_\nu \frac{2}{\sqrt{(N_x+1)(N_y+1)}} \sin\left(\frac{\pi m i}{N_x+1}\right)$ instead resulted in

$$c_\nu\left(v_{i,j-1}^{m,n} + v_{i,j+1}^{m,n}\right) = 2c_\nu \cos\left(\frac{\pi n}{N_y+1}\right) v_{i,j}^{m,n} \tag{4.4}$$

Similar to before, the right-hand side of the equation minus $v_{i,j}^{m,n}$ coincide with the third term in the definition of $\lambda_\nu^{m,n}$ in Theorem 4.1. Additionally replacing the index j in (4.3)

with $j-1$ and $j+1$ then adding both of them give us

$$b_\nu \left(v_{i-1,j-1}^{m,n} + v_{i+1,j-1}^{m,n}\right) = 2b_\nu \cos\left(\frac{\pi m}{N_x+1}\right) v_{i,j-1}^{m,n}$$

$$b_\nu \left(v_{i-1,j+1}^{m,n} + v_{i+1,j+1}^{m,n}\right) = 2b_\nu \cos\left(\frac{\pi m}{N_x+1}\right) v_{i,j+1}^{m,n}$$

$$b_\nu \left(v_{i-1,j-1}^{m,n} + v_{i+1,j-1}^{m,n} + v_{i-1,j+1}^{m,n} + v_{i+1,j+1}^{m,n}\right) = 2b_\nu \cos\left(\frac{\pi m}{N_x+1}\right) \left(v_{i,j-1}^{m,n} + v_{i,j+1}^{m,n}\right)$$

Multiply both side of the equation by c_ν now then using the equality establish by (4.4) we have

$$\begin{aligned} b_\nu c_\nu \left(v_{i-1,j-1}^{m,n} + v_{i+1,j-1}^{m,n} + v_{i-1,j+1}^{m,n} + v_{i+1,j+1}^{m,n}\right) &= 2b_\nu \cos\left(\frac{\pi m}{N_x+1}\right) \left[c_\nu \left(v_{i,j-1}^{m,n} + v_{i,j+1}^{m,n}\right)\right] \\ &= 2b_\nu \cos\left(\frac{\pi m}{N_x+1}\right) \left[2c_\nu \cos\left(\frac{\pi n}{N_y+1}\right) v_{i,j}^{m,n}\right] \\ &= 4b_\nu c_\nu \cos\left(\frac{\pi m}{N_x+1}\right) \cos\left(\frac{\pi n}{N_y+1}\right) v_{i,j}^{m,n} \end{aligned}$$

Multiply both sides of the equation by $\frac{a_\nu}{b_\nu c_\nu}$ this would give us the first term in the definition of $\lambda_\nu^{m,n}$ in Theorem 4.1 minus $v_{i,j}^{m,n}$ on the right-hand side of the equation.

$$a_\nu \left(v_{i-1,j-1}^{m,n} + v_{i+1,j-1}^{m,n} + v_{i-1,j+1}^{m,n} + v_{i+1,j+1}^{m,n}\right) = 4a_\nu \cos\left(\frac{\pi m}{N_x+1}\right) \cos\left(\frac{\pi n}{N_y+1}\right) v_{i,j}^{m,n} \quad (4.5)$$

Now observe that adding all Equations (4.3), (4.4) and (4.5) along with the term $d_\nu v_{i,j}^{m,n}$ and we get

$$\begin{aligned} a_\nu (v_{i-1,j-1}^{m,n} + v_{i+1,j-1}^{m,n} + v_{i-1,j+1}^{m,n} + v_{i+1,j+1}^{m,n}) + b_\nu \left(v_{i-1,j}^{m,n} + v_{i+1,j}^{m,n}\right) + c_\nu \left(v_{i,j-1}^{m,n} + v_{i,j+1}^{m,n}\right) \\ + d_\nu v_{i,j}^{m,n} = \lambda_\nu^{m,n} v_{i,j}^{m,n} \end{aligned}$$

By definition the left-hand side of the equation is the same as $C_l^m \mathbf{v}^{m,n}$, $C_l \mathbf{v}^{m,n}$ and $C_l^p \mathbf{v}^{m,n}$.

□

4.3.2 Diagonalization

Before proceeding to the next subsection, it is necessary to show that the matrix V in Theorem 4.1 is an orthogonal matrix. First take note that the vector $\mathbf{v}^{m,n}$ is always an eigenvector of the matrices C_l^m, C_l, and C_l^p regardless of the value of a_ν, b_ν, c_ν, or d_ν may take.

Hence consider the special case where a_ν, b_ν, c_ν, or d_ν are real, i.e, for the case of the 3D Helmholtz equation with the second-order scheme and real coefficient $k_0^2(z)$; or even 0 for the case of Poisson equation. $\mathbf{v}^{m,n}$ would still be an eigenvector for the matrix C_1. But the matrix C_1 is now a real symmetric matrix and thus it will also be Hermitian. It follows that the vectors $\mathbf{v}^{m,n}$ are eigenvectors of a Hermitian matrix thus they formed an orthogonal basis in $\mathbb{C}^n$ by Theorem 3.29 in [25] which states the eigenvalues of a Hermitian $n \times n$ matrix are real, and the eigenvectors form an orthogonal basis in $\mathbb{C}^n$.

This meant that we can define $\Lambda_l^m = V^T C_l^m V$, $\Lambda_l = V^T C_l V$ and $\Lambda_l^p = V^T C_l^p V$ where Λ_l^m, Λ_l and Λ_l^p are the diagonal matrices of eigenvalues. It follows that, for $l = 2, \ldots, N_z - 1$,

$$C_l^m \vec{u}_{l-1} + C_l \vec{u}_l + C_l^p \vec{u}_{l+1} = \vec{f_l}$$

$$V^T C_l^m V V^T \vec{u}_{l-1} + V^T C_l V V^T \vec{u}_l + V^T C_l^p V V^T \vec{u}_{l+1} = V^T \vec{f_l}$$

$$\Lambda_l^m \vec{w}_{l-1} + \Lambda_l \vec{w}_l + \Lambda_l^p \vec{w}_{l+1} = \overline{F_l}$$

where $\vec{w}_l = V^T \vec{u}_l$ and $\overline{F_l} = V^T \vec{f_l}$. Overall system (4.2) can be rewritten as

$$\begin{aligned}
\Lambda_1 \vec{w}_1 + \Lambda_1^p \vec{w}_2 &= \overline{F}_1, \\
\Lambda_l^m \vec{w}_{l-1} + \Lambda_l \vec{w}_l + \Lambda_l^p \vec{w}_{l+1} &= \overline{F}_l; \qquad \text{for } l = 2, \ldots, N_z - 1, \\
\Lambda_{N_z}^m \vec{w}_{N_z-1} + \Lambda_{N_z} \vec{w}_{N_z} &= \overline{F}_{N_z}.
\end{aligned} \tag{4.6}$$

This resulted in a block tridiagonal system that can be solved using LU decomposition. The computations in this solution are independent with respect to both the x and y directions of the computational domain. Therefore, it can be parallelized in either direction.

4.3.3 Computing $\overline{F_l}$ with FFT

Prior to solving the block tridiagonal system in Equation (4.6), $\overline{F_l} = V^T \vec{f_l}$ must be found. The computational complexity of this matrix-vector multiplication alone would be $\mathcal{O}(N_x^2 \cdot N_y^2)$. Moreover the calculation is required for each $\vec{f_l}$ where $l = 1, \ldots, N_z$. Transforming the right-hand side in this manner is not ideal even on modern computers. However

Definition 4.3.1. The **discrete sine transform** of the vector $\vec{x} = [x_1 \ \ldots \ x_n]^T \in \mathbb{C}^n$ is given by $\text{DST}(\vec{x}) = [\hat{x}_1 \ \ldots \ \hat{x}_n]^T$ where

$$\hat{x}_k = \sum_{l=1}^{n} \sin\left(\frac{kl\pi}{n+1}\right) x_l$$

for $k \in \{1, \ldots, n\}$ [25].

Thus the matrix-vector multiplication $V^T \vec{f_l}$ is simply the discrete sine transform (DST) in both the x and y directions. The computational cost may be reduced even further with a fast Fourier transform (FFT). FFT is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). The subsequent definition utilizes $i = \sqrt{-1}$ and i is not an index.

Definition 4.3.2. The **discrete Fourier transform** of the vector $\vec{y} = [y_1 \ \ldots \ y_n]^T \in \mathbb{C}^n$ is given by $\hat{y} = [\hat{y}_1 \ \ldots \ \hat{y}_n]^T$ where

$$\hat{y}_k = \sum_{l=1}^{n} \exp\left\{\frac{-2\pi i(l-1)(k-1)}{n}\right\} y_l$$

$$= \sum_{l=1}^{n} \left[\cos\left(\frac{2\pi(l-1)(k-1)}{n} \right) - i \sin\left(\frac{2\pi(l-1)(k-1)}{n} \right) \right] y_l$$

for $k \in \{1, \ldots, n\}$ [25].

Lemma 4.1. *Let $\vec{x} \in \mathbb{C}^n$ and $\vec{y} \in \mathbb{C}^{2n+2}$. If $\vec{y} = \begin{bmatrix} 0 & \vec{x}^T & 0 & \cdots & 0 \end{bmatrix}^T$ then $\hat{x} = -Im\left(\hat{y}_{2:n+1}\right)$ where $\hat{x}$ is the discrete sine transform of $\vec{x}$, $\hat{y}$ is the discrete Fourier transform of $\vec{y}$ and $-Im\left(\hat{y}_{2:n+1}\right)$ is the negative of the complex part of the vector elements in $\hat{y}$ from the second entry to the $(n+1)$-st entry.*

Proof. Let $N = 2n + 2$, then $\frac{2}{N} = \frac{1}{n+1}$. Define $y_1 = y_{n+2} = y_{n+3} = \cdots = y_{2n+2} = 0$ and $y_{l+1} = x_l$ for $l = 1, \ldots, n$. Then for $k = 1, \ldots, n$,

$$-Im(\hat{y}_{k+1}) = -Im\left(\sum_{l=1}^{N} \left[\cos\left(\frac{2\pi(l-1)k}{N} \right) - i \sin\left(\frac{2\pi(l-1)k}{N} \right) \right] y_l \right)$$

$$= \sum_{l=1}^{N} \sin\left(\frac{2k(l-1)\pi}{N} \right) y_l$$

$$= \sum_{l=2}^{n+1} \sin\left(\frac{k(l-1)\pi}{n+1} \right) y_l$$

$$= \sum_{l=1}^{n} \sin\left(\frac{kl\pi}{n+1} \right) y_{l+1}$$

$$= \sum_{l=1}^{n} \sin\left(\frac{kl\pi}{n+1} \right) x_l$$

$$= \hat{x}_k$$

□

Lemma 4.1 shows that computing the DST of a vector can be accomplished by using the discrete Fourier transform on an extension of the vector. In particular, let FFT_x and FFT_y

be the DFTs in the x and y directions respectively. Then $\overline{F_l} = V^T \vec{f_l} = FFT_y(FFT_x(\mathbf{f}_l))$ where $\mathbf{f}_l$ is the appropriate extension of $\vec{f_l}$. Now the computational cost for finding $\overline{F_l}$ is reduced significantly from order $\mathcal{O}(N_x^2 \cdot N_y^2)$ to $\mathcal{O}(N_x \cdot N_y \log(N))$ where $N = \max\{N_x, N_y\}$.

4.3.4 Solving block tridiagonal system with LU decomposition

System (4.6) can be presented as

$$\begin{bmatrix} \Lambda_1 & \Lambda_1^p & & \\ \Lambda_2^m & \Lambda_2 & \ddots & \\ & \ddots & \ddots & \Lambda_{N_z-1}^p \\ & & \Lambda_{N_z}^m & \Lambda_{N_z} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{N_z} \end{bmatrix} = \begin{bmatrix} \overline{F}_1 \\ \overline{F}_2 \\ \vdots \\ \overline{F}_{N_z} \end{bmatrix} \tag{4.7}$$

Let Λ be the matrix in Equation (4.7). Then Λ is a block tridiagonal matrix where each block is a diagonal matrix. Before the solution to Equation (4.7) is computed, it is preferable to permute the matrix Λ.

This is done to improve the data locality which is a key to good performance on all modern CPUs and fine-grained architectures. Data locality refers to the process of moving computation to the node where that data resides, instead of vice versa. This help minimizes network congestion and improves computation throughput. In this instance, since each block in Λ are diagonal matrix or zero matrix, the data locality can be improved by placing the nonzero entries right next to each other.

Thus a permutation of Λ, say P, is considered where the permutation of Λ, denoted by $\tilde{\Lambda} = P(\Lambda)$, would be a tridiagonal matrix instead. For the proposed numerical solver, the creation of $\tilde{\Lambda}$ is considered as a preliminary step, that is, it is only necessary to compute the matrix once at the start of the algorithm and can be stored for repeated uses. As such it would not hinder the performance of the numerical algorithm. For the vector $\overline{F}$, one can

easily reorder this array by simply rearranging the index so that the array would loop in the z-direction first.

Figure 4.2 illustrates a simple example of reordering the array in which the array loop in the vertical, y-direction first for the two-dimensional case. The number shows the order in which the array is stored and x represents the horizontal direction while y represents the vertical direction. Algorithm 1 shows how to reorder the array for three-dimensional cases in the C programming language environment.

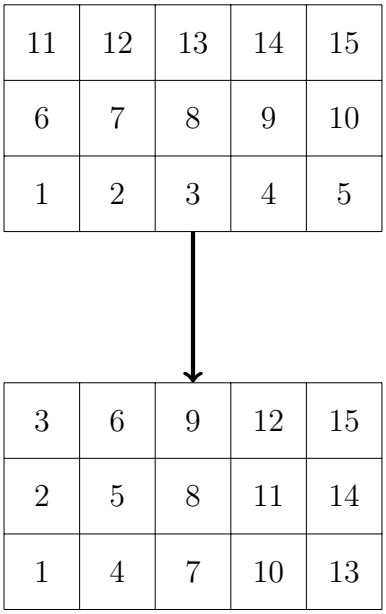

Figure 4.2: Reordering of array in 2D case

Algorithm 1 Reordering array for 3D cases in C

```
for l = 1, ..., N_z do
    for i = 1, ..., N_x · N_y do
        F̃ [l + i · N_z] = F̄ [i + l · (N_x · N_y)]
    end for
end for
```

Now $\tilde{\Lambda}$ is a tridiagonal matrix which can be illustrated as

$$\tilde{\Lambda} = \begin{bmatrix} A_1 & 0 & & \\ 0 & A_2 & \ddots & \\ & \ddots & \ddots & 0 \\ & & 0 & A_{N_x \cdot N_y} \end{bmatrix}$$

where A_i are tridiagonal matrices of size $N_z \times N_z$ with non zero entries on its main diagonal term for $i = 1, \ldots, N_x \cdot N_y$. Now the newly modified system can be solved with LU decomposition for each of the A_i. Each A_i can also be solved simultaneously as well making it highly scalable. The LU factorization of $A_i = L_i U_i$ is given explicitly as

$$A_i = \begin{bmatrix} a_{i,1} & a^p_{i,1} & & \\ a^m_{i,2} & a_{i,2} & \ddots & \\ & \ddots & \ddots & a^p_{i,N_z-1} \\ & & a^m_{i,N_z} & a_{i,N_z} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ l_{i,2} & 1 & & \\ & \ddots & \ddots & \\ & & l_{i,N_z} & 1 \end{bmatrix} \begin{bmatrix} u_{i,1} & a^p_{i,1} & & \\ & u_{i,2} & \ddots & \\ & & \ddots & a^p_{i,N_z-1} \\ & & & u_{i,N_z} \end{bmatrix}$$

where the formula for $u_{i,1}$, $u_{i,k}$ and $l_{i,k}$ where $k = 2, \ldots, N_z$ are given by

$$\tilde{u}_{i,1} = \frac{1}{u_{i,1}} = \frac{1}{a_{i,1}}$$

$$l_{i,k} = a^m_{i,k} \cdot \tilde{u}_{i,k-1}$$

$$\tilde{u}_{i,k} = \frac{1}{u_{i,k}} = \frac{1}{a_{i,k} - l_{i,k} \cdot a^p_{i,k-1}},$$

In practice, rather than saving the values of $u_{i,k}$, it is better to store the values of its reciprocal as a way to reduce the division required. Moreover, none of the terms in the main diagonal would be zero so $\tilde{u}_{i,k}$ is well-defined. After the LU factorization, solving system (4.6) is now equivalent to solving

$$\tilde{F}_i = A_i \tilde{w}_i = L_i U_i \tilde{w}_i$$

for $i = 1, \ldots, N_x \cdot N_y$. Here $\tilde{F}_i$ and $\tilde{w}_i$ are vector of size N_z that is parts of $\tilde{F} = P(\overline{F})$ and $\tilde{w} = P(\vec{w})$ respectively where P is the permutation on A mentioned earlier. The tridiagonal linear system can be solved by first finding $\vec{y} = [y_1 \ \cdots \ y_{N_z}]^T$ in $L_i \vec{y} = \tilde{F}_i$ and then solving

for $\tilde{w}_i$ in $U_i\tilde{w}_i = \vec{y}$. The formula to solve $L_i\vec{y} = \tilde{F}_i$ is given by

$$y_1 = \tilde{F}_{i,1}$$

$$y_k = \tilde{F}_{i,k} - l_{i,k}y_{k-1}; \quad \text{for } k = 2, \ldots, N_z,$$

and $U_i\tilde{w}_i = \vec{y}$ can be solve with

$$\tilde{w}_{i,N_z} = y_{N_z} \cdot \tilde{u}_{i,N_z}$$

$$\tilde{w}_{i,k} = \left(y_k - a^p_{i,k} \cdot \tilde{w}_{i,k+1}\right) \cdot \tilde{u}_{i,k}; \quad \text{for } k = N_z - 1, \ldots, 1,$$

Solving $\tilde{F}_i = L_iU_i\tilde{w}_i$ would required $\mathcal{O}(N_z)$ multiplication for each i. At the end of this process the order of the solution would be reverted to its original order, i.e., $\vec{w} = P^{-1}(\tilde{w})$. With all these steps, the direct FFT solver can solve the PDE by computing the solution vector $\vec{u}$ which is simply the reverse DFT of $\vec{w}$, that is $\vec{u}_l = V\vec{w}_l$ for $l = 1, \ldots, N_z$. This is equivalent to calculating $\vec{u}_l = V\vec{w}_l = FFT_y\left(FFT_x\left(\mathbf{w}_l\right)\right)$ where $\mathbf{w}_l$ is the appropriate extension of $\vec{w}_l$. The computational complexity of calculating $\vec{u}_l$ from $\vec{w}_l$ is $\mathcal{O}(N_x \cdot N_y \log(N))$ where $N = \max\{N_x, N_y\}$. These are the complete steps of the direct FFT solver.

4.4 Sequential Algorithm of Direct FFT Solver

Algorithm 2 details the sequential implementation of the direct FFT solver.

Algorithm 2 Sequential Implementation of Direct FFT Solver

1: Create FFTW plan
2: **for** $l = 1, \ldots, N_z$ **do**
3: 2D forward DST of the RHS in $x-, y-$direction
4: Reorder the RHS to improve data locality
5: **end for**
6: **for** $i = 1, \ldots, N_x \cdot N_y$ **do**
7: Solve the tridiagonal system $A_i w_i = f_i$ using LU decomposition
8: Reorder the solution vector back
9: **end for**
10: **for** $l = 1, \ldots, N_z$ **do**
11: 2D reverse DST of the solution w in $x-, y-$direction
12: **end for**

It is worth mentioning that the reordering of the array in Algorithm 1 is done at the same time when the array is stored after the computation of FFT and solving the tridiagonal system. So that there would be no additional cost required to perform said action. The FFT transform used in the direct FFT solver algorithm in Algorithm 2 is calculated using an open source C subroutine library developed at Massachusetts Institute of Technology (MIT), namely, FFTW [22]. This subroutine is currently considered the standard in FFT calculation and is available for free to the public. Additionally, Algorithm 2 can also be implemented in MATLAB as MATLAB already has a build-in fft function readily available.

5 Iterative Methods

5.1 Introduction

To find the numerical solution of a PDE, it is a common practice to convert the differential equation, which may be nonlinear, into a system of the linear equation as follows

$$A\vec{u} = \vec{f}$$

where A is the coefficient matrix arising from the discretization of the PDE as seen in Chapter 3, $\vec{u}$ is the unknown solution vector and f is the right-hand side vector that is given. Generally, the coefficient matrix is very large and sparse. It is also usually non-symmetric and non-positive definite when the coefficient function k^2 is not constant for the case of the Helmholtz equation. For the convection-diffusion equation, it will be non-symmetric and non-positive definite if the convection coefficients are nonzero.

The direct methods introduced in Chapter 4 would only be capable of solving PDE that satisfies certain constraints on its coefficient matrix after discretization. Other direct methods such as Gaussian elimination could apply the sparse-elimination method to exploit sparsity in the coefficient matrix and reduce computational requirements. Generally, direct methods work well for systems with thousands of degrees of freedom, but they require unfeasible computation resources for linear systems of much larger dimensions. The practical limit for feasibility is often that direct sparse methods are competitive for two-dimensional PDE problems but iterative methods are required in three-dimensional cases [10]. See Duff et al. [12] or George & Liu [1] for an overview of these ideas.

Iterative solution methods are considered the best choice in this type of situation. A feature of iterative methods is that they can take full advantage of the sparsity of the coefficient matrix. In particular, their storage requirements typically depend only on the number of non-zeros in the matrix. The aim then becomes to make convergence as fast as possible.

The proposed iterative solvers are based on the GMRES-type algorithm. In this chapter, an overview of classical iterative methods such as Jacobi and Gauss-Seidel methods will be provided, to introduce a modern class of Krylov subspace methods, Arnoldi iteration, and GMRES method in the following sections. A section will be dedicated to the discussion of the preconditioner which is often the heart of the iterative methods. This chapter will end with a section showing how the iterative algorithm will be implemented.

5.2 Classical Iterative Methods

The iterative method is a numerical method that uses an initial value to create a series of improving approximate solutions for a class of problems, in which the k-th approximation is derived from the previous ones.

Let $\vec{u}_k$ be the approximate solution obtained at k-th iteration and $\vec{u}_0$ be the initial guess. Also define the error and residual as $\vec{e}_k = \vec{u} - \vec{u}_k$ and $\vec{r}_k = \vec{f} - A\vec{u}_k$ respectively. Then the error equation is given by

$$A\vec{e}_k = A(\vec{u} - \vec{u}_k) = \vec{f} - A\vec{u}_k = \vec{r}_k$$

This equation can be solved for $\vec{e}_k$ to find the unknown solution $\vec{u} = A^{-1}\vec{r}_k + \vec{u}_k$.

In classical iterative methods, the coefficient matrix A is considered as the sum of a non-singular matrix A_p, which is called preconditioner, and $A_s = A - A_p$. So that the matrix

A^{-1} can be approximated using A_p^{-1}, which is easier to invert, and the next approximate solution is

$$\vec{u}_{k+1} = \vec{u}_k + A_p^{-1}\vec{r}_k \tag{5.1}$$

for $k = 0, 1, \ldots$

The Jacobi method defines $A_p = diag(A)$ as the diagonal part of A while the Gauss-Seidel method uses the strictly upper triangular part of A as its preconditioner A_p. In either case, the matrix A_p^{-1} is not necessarily a good approximation to the inverse of A. To obtain a better result, a non-stationary iterative method is introduced by adding a parameter $\alpha_k \in \mathbb{C}$ into Equation (5.1) as follows

$$\vec{u}_{k+1} = \vec{u}_k + \alpha_k A_p^{-1}\vec{r}_k \tag{5.2}$$

referred to as the non-stationary Richardson iteration. The parameter α_k is used to control the length of the vector $A_p^{-1}\vec{r}_k$ at each step. The optimal choice for the parameters α_k is still an area of active research but there is no need to retrieve them explicitly in modern Krylov subspace methods. For the non-stationary Richardson method, one can observe that the residuals are related by

$$\vec{r}_{k+1} = \vec{f} - A\vec{u}_{k+1} = \vec{f} - A(\vec{u}_k + \alpha_k A_p^{-1}\vec{r}_k) = \vec{r}_k - \alpha_k A A_p^{-1}\vec{r}_k \tag{5.3}$$

for all $k \geq 0$. Suppose that $A_p = I$ the identity matrix in Equation (5.3), then Equation (5.3) can be rewritten as

$$\vec{r}_{k+1} = (I - \alpha_k A)\vec{r}_k = \prod_{i=0}^{k}(I - \alpha_i A)\vec{r}_0 \tag{5.4}$$

Equation (5.4) shows that residual $\vec{r}_{k+1}$ can be expressed as product of a polynomial in A of degree k and its initial residual, $\vec{r}_0$. Substitute this expression into Equation (5.2) it follows

that any approximation $\vec{u}_{k+1}$ generated by the non-stationary Richardson method can be written as a combination of the initial guess $\vec{u}_0$ and a polynomial $p_k \in \mathbb{P}_k$ in A:

$$\vec{u}_{k+1} = \vec{u}_0 + p_k(A)\vec{r}_0$$

Thus the non-stationary Richardson method can reach any element in the affine space $\vec{u}_0 + \mathcal{K}_k$ where $\mathcal{K}_k = \text{span}\{\vec{r}_0,\ A\vec{r}_0,\ \cdots,\ A^{k-1}\vec{r}_0\}$ which is the Krylov subspace. Note that from the perspective of approximation theory, A^{-1} is approximated by a polynomial as $A^{-1} \approx p_k(A)$. In the next section, we will see that the inverse of A can be expressed exactly as a polynomial of the highest degree n so that the solution $\vec{u} = \vec{u}_0 + A^{-1}\vec{r}_0$ would lie within the affine space $\vec{u}_0 + \mathcal{K}_n$ for some $n \geq 0$.

5.3 Krylov Subspace Methods

Previously we have seen that classical iterative methods iterate within an affine space $\vec{u}_0 + \mathcal{K}_k$ where the latter space is a Krylov subspace. In this section, we will show that the exact solution to a linear system is an element in this subspace. This is a powerful result for iterative methods which use a Krylov subspace as their search space for the solution. Now the numerical solution to the PDE can be found in a fixed number of iterations if the Krylov subspace is enlarged at each iteration. Note also that a new basis vector for the Krylov subspace is simply found by a single matrix-vector product with the previous basis vector. Various iterative methods have been proposed which use this subspace as a search space for the solution. These Krylov subspace methods will be classified in this section, and the Arnoldi iteration will be discussed in the next section as a method for finding a numerically stable basis for the Krylov subspace.

Definition 5.3.1. A Krylov subspace denoted by $\mathcal{K}_m(A, \vec{f})$ is a subspace of $\mathbb{C}^n$ spanned by m vectors, induced by a matrix $A \in \mathbb{C}^{n\times n}$ and a vector $\vec{f} \in \mathbb{C}^n$ as

$$\mathcal{K}_m(A, \vec{f}) = \text{span}\{\vec{f},\ A\vec{f},\ \cdots,\ A^{m-1}\vec{f}\}$$

$\mathcal{K}_m$ is called the m-th order Krylov subspace.

As a consequence, every element $\vec{v} \in \mathcal{K}_m(A, \vec{f})$ of a Krylov subspace can be expressed as a polynomial $p_{m-1} \in \mathbb{P}_{m-1}$ as $\vec{v} = p_{m-1}(A)\vec{f}$. Krylov subspace is of particular interest because the solution $\vec{u}$ of the linear system $A\vec{u} = \vec{f}$ lies within a Krylov subspace as long as A is a non-singular matrix. To show this, we construct a polynomial $q(A)$ of the lowest degree for which $q(A) = 0$.

Definition 5.3.2. The minimal polynomial with respect to $A \in \mathbb{C}^{n\times n}$ is the unique, monic polynomial q over A with lowest degree such that

$$q(A) = \mathbf{0}$$

where $\mathbf{0}$ refers to the zero matrix.

Theorem 5.1 will show that the degree of the minimal polynomial does not exceed n which in turn guarantees that $A^{-1} \in \mathbb{P}_n(A)$. This implies the numerical solution to the PDE can be found within n iterations for a $n \times n$ matrix A with the Krylov subspace methods.

Theorem 5.1. *Let V be a vector space of dimension n over the field of either real numbers $\mathbb{R}$ or complex numbers $\mathbb{C}$. Let $A : V \to V$ be a linear transformation. Then the minimal polynomial $q(A)$ is of degree less than or equal to n.*

Proof. We prove the theorem by using mathematical induction on the dimension of V.

Suppose that the dimension of V is 1, that is $dim(V) = 1$. Then any nonzero vector from V is a constant multiple of a generating vector $\vec{x}$. Since A is a linear transformation, $A(\vec{x}) = k\vec{x}$ for some scalar k. Hence, $(kI - A)\vec{x} = \vec{0}$. Then $q(A) = kI - A = \mathbf{0}$ and degree of q is 1 which is equal to $dim(V)$. So it is true for $dim(V) = 1$.

Now, suppose that the theorem is true for all V of dimension $k-1$ where $1 < k \leq n$. Let $dim(V) = k$, and suppose that $\vec{x}$ is a nonzero vector in V. Then consider the $k+1$ vectors $\{\vec{x}, A\vec{x}, A^2\vec{x}, \ldots, A^k\vec{x}\}$, this will be linearly dependent set since there are more vectors than the dimension of the space, that is there exist scalars $\alpha_i, i = 0, 1, \ldots, k$, where not all them are zero, depending on $\vec{x}$ such that

$$\alpha_0\vec{x} + \alpha_1 A\vec{x} + \cdots + \alpha_k A^k\vec{x} = \vec{0}$$

Define $f(T) = \alpha_0 I + \alpha_1 T + \cdots + \alpha_k T^k$ where $T : V \to V$ is a linear transformation. Then the degree of f would be less than or equal to k and $f(A)\vec{x} = \vec{0}$. If $f(A)\vec{v} = \vec{0}$ for all $\vec{v} \in V$, then we are done as $f(A) = \mathbf{0}$ so the degree of the minimal polynomial would be less than the degree of f which is n.

Otherwise suppose $f(A)\vec{u} \neq \vec{0}$ for some $\vec{u} \in V$. Define $U = \{\vec{u} \in V \mid f(A)\vec{u} \neq \vec{0}\}$, then clearly $\vec{x}$ is not an element of U so U is a proper subspace of V, in particular we have $1 \leq dim(U) < dim(V)$. By the mathematical induction assumption there exist a polynomial g_1 of degree less than or equal to $dim(U) = l$ such that $g_1(A)\vec{u} = \vec{0}$ for all $\vec{u} \in U$. Now consider the quotient space V/U. Once again $dim(V/U) = k - l < k$ so by the mathematical induction there exist a polynomial g_2 of degree less than or equal to $k-l$ such that $g_2(A)\vec{w} = \vec{0}$ for all $\vec{w} \in V/U$.

Let $h(T) = g_1(T)g_2(T)$. Since g_1 and g_2 are polynomials they do in fact commute with each

other. Now for any $\vec{v} \in V$, $\vec{v} = \vec{u} + \vec{w}$ where $\vec{u} \in U$ and $\vec{w} \in V/U$. Furthermore

$$h(A)\vec{v} = h(A)(\vec{u} + \vec{w}) = g_2(A)g_1(A)\vec{u} + g_1(A)g_2(A)\vec{w} = \vec{0}$$

By definition the minimal polynomial q would have degree less than h which equal $deg(h) = deg(g_1) + deg(g_2) = k$. Thus the theorem holds in all cases and the proof is complete. □

A stronger version of Theorem 5.1 is given by the Cayley-Hamilton theorem [2] which states that for any square matrix A, there exists a polynomial p that annihilates A, that is $p(A) = \mathbf{0}$. This theorem also specifies that the characteristic polynomial p_A is an annihilator for A.

Lemma 5.1. *Let A be a non-singular $n \times n$ matrix. If $\vec{u}$ is the solution to the linear system $A\vec{u} = \vec{f}$, then $\vec{u} \in \mathcal{K}_m(A, \vec{f})$ for some $0 \leq m \leq n$.*

Proof. Suppose A has distinct eigenvalues $\{\lambda_1, \cdots, \lambda_d\}$ with $d \leq n$ where each eigenvalue λ_j has index m_j. Denote $m = \sum_{j=1}^{d} m_j$ the sum of the indexes, then

$$p_A(A) = \prod_{j=1}^{d} (A - \lambda_j I)^{m_j}$$

is the characteristic polynomial of A. By the Cayley-Hamilton theorem [2], when the polynomial is expended we have

$$\mathbf{0} = p_A(A) = \alpha_0 I + \alpha_1 A + \cdots + \alpha_m A^m$$

for all $\alpha_j \in \mathbb{C}$, it is clear that $\alpha_0 \neq 0$ since the eigenvalues are non-zero. By using $I = AA^{-1}$ and moving terms around, we have

$$\begin{aligned} -\alpha_0 A A^{-1} &= \alpha_1 A + \alpha_2 A^2 + \cdots + \alpha_m A^m \\ AA^{-1} &= -\frac{1}{\alpha_0}\left(\alpha_1 A + +\alpha_2 A^2 + \cdots + \alpha_m A^m\right) \\ A^{-1} &= -\frac{1}{\alpha_0}\left(\alpha_1 I + +\alpha_2 A + \cdots + \alpha_m A^{m-1}\right) \end{aligned}$$

So the inverse of A can be written as a polynomial of the highest degree $m-1$ that is

$$A^{-1} = -\frac{1}{\alpha_0}\sum_{j=0}^{m-1}\alpha_{j+1}A^j$$

Therefore, the solution $\vec{u} = A^{-1}\vec{f}$ is an element of the Krylov subspace $\mathcal{K}_m(A, \vec{b})$. □

From a numerical point of view, it is necessary to find a better basis for the Krylov subspace, as the basis vectors in Definition 5.3.1 is exactly the first iterates of the power method. Since the power method converges to the eigenvector corresponding to the eigenvalue of the largest magnitude if it exists, the basis vectors tend to become numerically dependent due to finite precision. To overcome this problem, Arnoldi's algorithm is often used to create an orthonormal basis for the Krylov subspace.

5.4 Arnoldi Iteration

The Arnoldi iteration is an eigenvalue algorithm used to explicitly constructs an orthonormal basis $\{\vec{v}_1, \cdots, \vec{v}_m\}$ for the Krylov subspace $\mathcal{K}_m(A, \vec{r}_0)$ and a Hessenberg matrix $H_m = (hij)$. In this method, the modified Gram–Schmidt process is used to produce a sequence of orthonormal vectors, $\{\vec{v}_1, \cdots, \vec{v}_m\}$ called the Arnoldi vectors, such that for every $m \geq 1$, the vectors $\{\vec{v}_1, \cdots, \vec{v}_m\}$ span the m-th order Krylov subspace, $\mathcal{K}_m$. The Arnoldi vectors form the matrix $V_m = [\vec{v}_1 \cdots \vec{v}_m]$. The algorithm begins with a normalized vector $\vec{v}_1$, and orthogonalizes each new basis vector $A\vec{v}_i$ to all previous vectors $\{\vec{v}_1, \cdots, \vec{v}_{i-1}\}$. The algorithm terminates after step j if $h_{j+1,j} = 0$. If the Arnoldi algorithm does not terminate at the m-th step, it can also be formulated in terms of the matrices V_{m+1} and H_m. Explicitly, the

algorithm is as shown in Algorithm 3

Algorithm 3 The Arnoldi algorithm

1: $\vec{v}_1 := \vec{r}_0/\|\vec{r}_0\|$
2: **for** $j = 1, \ldots, m$ **do**
3: **for** $i = 1, \ldots, j$ **do**
4: $h_{i,j} = (A\vec{v}_j, \vec{v}_i)$
5: **end for**
6: $\vec{w}_{j+1} = A\vec{v}_j - \sum_{i=1}^{j} h_{i,j}\vec{v}_i$
7: $h_{j+1,j} = \|\vec{w}_{j+1}\|_2$
8: **if** $h_{j+1,j} = 0$ **then**
9: **return**
10: **else**
11: $\vec{v}_{j+1} = \vec{w}_{j+1}/h_{j+1,j}$
12: **end if**
13: **end for**

5.5 GMRES

GMRES was developed by Saad & Schultz [32] is a Krylov subspace method that computes at the r-th step the best least squares solution $\vec{u}_r$ from the Krylov subspace $\mathcal{K}_r(A, \vec{r}_0)$. While the application of the classical iterative solvers was limited to either diagonally dominant or positive definite matrices, the GMRES method can be used for linear systems $A\vec{u} = \vec{f}$ with arbitrary (nonsingular) square matrices A. The essential ingredient in this general iterative solver is the Arnoldi iteration.

The main idea behind the GMRES method is to minimize the residual in Euclidean norm over all vectors of the m dimensional affine space $\vec{u}_0 + \mathcal{K}_m(A, \vec{r}_0)$, where $\vec{u}_0 \in \mathbb{C}^n$ is an initial guess and the latter space is the m-th order Krylov subspace induced by A and $\vec{r}_0$. This space is iteratively enlarged until a satisfactory approximate solution is found. Thus formally, at

every m-th iteration of GMRES the approximate solution $\vec{u}_m$ is

$$\vec{u}_m = \arg\min_{\vec{z}\in\vec{u}_0+\mathcal{K}_m(A,\vec{r}_0)}\|A\vec{z}-f\|_2$$

It is trivial that the Krylov subspace generated at each iteration contains the previous Krylov subspaces, i.e. $\mathcal{K}_m(A,\vec{r}_0)\subset\mathcal{K}_{m+1}(A,\vec{r}_0)$ for all $m\geq 1$. Therefore it holds that the sequence of norms of minimal residuals found by GMRES is non-increasing, this is referred to as the optimal property of GMRES.

To find such a minimizing vector $\vec{u}_m\in\vec{u}_0+\mathcal{K}_m(A,\vec{r}_0)$, write $\vec{u}_m=\vec{u}_0+\vec{y}$ for a $\vec{y}\in\mathcal{K}_m(A,\vec{r}_0)$. Let the matrix V_m be obtained by the Arnoldi iteration to span the Krylov subspace, then there exists a vector $\vec{z}\in\mathbb{C}^n$ such that $\vec{y}=V_m\vec{z}$.

One of the difficulties with the full orthogonalization method is that it becomes increasingly expensive as the step number m increases. When m increases, the number of vectors requiring storage increases like m and the number of multiplications like $\frac{1}{2}m^2N$ where N is the size of A. To remedy this difficulty, the GMRES algorithm is restarted after a fixed number, say k, of iterations. The resulting method is called GMRES(k) or Restarted GMRES. A drawback to this is that the iterative method may suffer from stagnation in convergence as the restarted subspace is often close to the earlier subspace.

The overall GMRES(k) algorithm is shown in Algorithm 4 where the initial guess $\vec{u}_0$ is the zero vector

Algorithm 4 GMRES(m)

1: Let $\vec{r}_0 = \vec{f}$, $\beta = \|\vec{r}_0\|_2$ and $\vec{v}_1 = \vec{r}_0/\beta$
2: **for** $j = 1, 2, \cdots, m$ **do**
3: Compute the vector $\vec{w}_j = A\vec{v}_j$
4: **for** $i = 1, 2, \cdots, j$ **do**
5: $h_{i,j} := (\vec{w}, \vec{v}_i)$ and $\vec{w} := \vec{w} - h_{i,j}\vec{v}_i$
6: **end for**
7: Compute $h_{j+1,j} = \|\vec{w}\|_2$ and $\vec{v}_{j+1} = \vec{w}/h_{j+1,j}$
8: **end for**
9: Define $V_m = [\vec{v}_1, \cdots, \vec{v}_m]$, $H_m = \{h_{i,j}\}_{1 \le i \le j+1; 1 \le j \le m}$
10: Form the approximate solution:
Compute $\vec{u}_m = V_m \vec{y}_m$ where $\vec{y}_m = argmin_{\vec{y}} \|\beta \vec{e}_1 - H_m \vec{y}\|$ and $\vec{e}_1 = [1, 0, \ldots, 0]^T$
11: If satisfied then stop else restart the algorithm by setting $\vec{r}_0 = \vec{f} - A\vec{u}_m$ go to 2

5.5.1 Convergence of GMRES

By its design, GMRES iteration will always converge in at most n steps for a system with $n \times n$ matrix A. Moreover, convergence is monotonic since $\|\vec{r}_{m+1}\| \le \|\vec{r}_m\|$. This latter fact is true since $\vec{r}_{m+1}$ is minimized over $\mathcal{K}_{m+1}(A, \vec{r}_0)$, and we have $\mathcal{K}_m(A, \vec{r}_0) \subset \mathcal{K}_{m+1}(A, \vec{r}_0)$. Thus, minimization over a larger subspace will allow us to achieve a smaller residual norm. For practical computations, the only case of interest is when the algorithm converges (to within a specified tolerance) in k iterations where $k < n$. Usually, the relative residual is used to control the convergence, i.e., we check whether

$$\frac{\|\vec{r}_k\|}{\|\vec{r}_0\|} \le \text{ tol}$$

where tol represents the specific tolerance which is usually around 10^{-6}.

5.6 Preconditioner

Preconditioning techniques are effective for accelerating the convergence of the GMRES algorithm. The general principle of preconditioning is to construct a matrix, say A_p, that approximates the coefficient matrix A but for which it requires little work to apply the action of the inverse of A_p, that is, to compute $A_p^{-1}\vec{x}$. Thus one may consider solving

$$A_p^{-1}A\vec{u} = A_p^{-1}\vec{f}$$

instead of $A\vec{u} = \vec{f}$; they clearly have the same solution. This is also called the left preconditioning. Now the GMRES in Algorithm 4 would be modified as follow to match this modified system as seen in Algorithm 5 with the initial guess $\vec{u}_0 = \vec{0}$.

Algorithm 5 GMRES(m) with Left Preconditioning

1: Find $\vec{r}_0 = A_p^{-1}\vec{f}$ by solving $A_p\vec{r}_0 = \vec{f}$ with preconditioner solver
2: Compute $\beta = \|\vec{r}_0\|_2$ and $\vec{v}_1 = \vec{r}_0/\beta$
3: **for** $j = 1, 2, \cdots, m$ **do**
4: Compute the matrix-vector multiplication $\vec{g} = A\vec{v}_j$
5: Compute the vector $\vec{w} = A_p^{-1}\vec{g}$ by solving $A_p\vec{w} = \vec{g}$ with preconditioner solver
6: **for** $i = 1, 2, \cdots, j$ **do**
7: $h_{i,j} := (\vec{w}, \vec{v}_i)$ and $\vec{w} := \vec{w} - h_{i,j}\vec{v}_i$
8: **end for**
9: Compute $h_{j+1,j} = \|\vec{w}\|_2$ and $\vec{v}_{j+1} = \vec{w}/h_{j+1,j}$
10: **end for**
11: Define $V_m = [\vec{v}_1, \cdots, \vec{v}_m]$, $H_m = \{h_{i,j}\}_{1\le i\le j+1; 1\le j\le m}$
12: Form the approximate solution:
Compute $\vec{u}_m = V_m\vec{y}_m$ where $\vec{y}_m = argmin_{\vec{y}}\|\beta\vec{e}_1 - H_m\vec{y}\|$ and $\vec{e}_1 = [1, 0, \ldots, 0]^T$
13: If satisfied then stop else set $\vec{r}_0 = A_p^{-1}(\vec{f} - A\vec{u}_m)$ go to 2

If A_p is a good approximation of A, then it is expected that the iterative algorithm would converge at a rapid rate. In fact, for very large problems, preconditioning is necessary to make computation feasible. Effective preconditioning is often at the heart of efficient solution algorithms, in particular when solving three-dimensional PDE problems.

In this section we will discuss two different types of preconditioning techniques considered in this dissertation:

(i) approximating the numerical solution of the original PDE with a similar PDE that can be solved directly with the direct FFT solver, for example in the case of Helmholtz equation, one may replace the coefficient function $k^2(x, y, z)$ with some coefficient function $k_0^2(z)$

(ii) approximating the higher order approximation scheme by using the lower order approximation scheme as a preconditioner

5.6.1 Helmholtz equation

The basic principle of preconditioning is to create an approximation to the original system where the preconditioner system can be solved efficiently with a preconditioner solver. In the proposed iterative method, the preconditioner system is modeled as a variation of the original PDE problem that could be solved directly with the direct FFT solver introduced in Chapter 4.

Hence, for the case of the Helmholtz equation, we consider the numerical solution of the Helmholtz equation with the same right-hand side function $f(x, y, z)$ and computational domain as in (2.1) but with a different coefficient function k_0^2 that depends only on the variable z, that is

$$\Delta u_h(x, y, z) + k_0^2(z) u_h(x, y, z) = f(x, y, z) \tag{5.5}$$

where $u_h(x, y, z)$ is the numerical solution of Equation (5.5) that can be computed with direct FFT solver. Here are some examples and reasonable options for the function $k_0^2(z)$.

General examples

Consider the case where $k^2(x,y,z)$ is a separable function that is $k^2(x,y,z) = g(x,y)h(z)$ for some functions g and h. Let $k_0^2(z) = h(z)$ and $\mathbf{x} = (x,y,z)$. Then

$$\Delta u(\mathbf{x}) + k^2(\mathbf{x})u(\mathbf{x}) = \Delta u_h(\mathbf{x}) + k_0^2(z)u(\mathbf{x})$$

$$\Delta u(\mathbf{x}) + k^2(\mathbf{x})u(\mathbf{x}) - k_0^2(z)u(\mathbf{x}) + k_0^2(z)u(\mathbf{x}) = \Delta u_h(\mathbf{x}) + k_0^2(z)u(\mathbf{x})$$

$$\Delta[u(\mathbf{x}) - u_h(\mathbf{x})] + k_0^2(z)[u(\mathbf{x}) - u_h(\mathbf{x})] = -[k^2(\mathbf{x}) - k_0^2(z)]u(\mathbf{x})$$

and

$$\|k^2(\mathbf{x}) - k_0^2(z)\| = \|g(x,y) - 1\|\|k_0^2(z)\|$$

As such, the preconditioner system is expected to have better performance when the value of $\|g(x,y) - 1\|$ is small. Similar approach can be taken if $k^2(\mathbf{x}) = g(x,y) + h(z)$ then let $k_0^2(z) = h(z)$ and we have $\|k^2(\mathbf{x}) - k_0^2(z)\| = \|g(x,y)\|$.

Subsurface inclusion model problem

For the main targeted application of the proposed numerical method, the coefficient function $k_0^2(z)$ in the preconditioner system is constructed by considering the case when the function $k^2(x,y,z)$ has no inclusions, that is,

$$k_0^2(z) = \begin{cases} 439.2, & \text{if } 0 < z < 0.5, \\ 1273 + 31i, , & \text{if } 0.5 \leq z < 1. \end{cases}$$

With this preconditioner system, one can observe that the size of the inclusion would determine the difference between $k_0^2(z)$ to $k^2(x,y,z)$ and consequently the performance of the

preconditioner. However, in practice, the size of the inclusion would be small in comparison to the computational domain, so excellent performance can be expected from the iterative methods with this preconditioner.

It is worth noting that the only requirement for the choice of k_0^2 is that it depends on one and only one spatial variable, that is the function k_0^2 could be $k_0^2(x)$ or $k_0^2(y)$ whichever would be a better approximation. In the former case define the function $g(x, y, z) = f(z, y, x)$. Once this condition is satisfied, then Equation (5.5) could be solved directly with the direct FFT solver.

5.6.2 Convection-diffusion equation

Convection-diffusion equation with constant coefficients

For the convection-diffusion equation with constant coefficients, the numerical solution of a variant of Equation (2.10) is considered where $\alpha = \beta = 0$ as seen in subsection 4.2.4. Let $\mathbf{x} = (x, y, z)$ then the modified convection-diffusion equation is given by

$$\Delta u_h(\mathbf{x}) + \gamma \frac{\partial}{\partial z} u_h(\mathbf{x}) = f(\mathbf{x}) \tag{5.6}$$

with the same right hand side function f and computational domain as Equation (2.10). In this setup, we have

$$\Delta[u(\mathbf{x}) - u_h(\mathbf{x})] + \gamma \frac{\partial}{\partial z}[u(\mathbf{x}) - u_h(\mathbf{x})] = -\left(\alpha \frac{\partial}{\partial x} u(\mathbf{x}) + \beta \frac{\partial}{\partial y} u(\mathbf{x})\right)$$

If the values of α and β are small and negligible, the iterative method is expected to converge at a faster rate.

Convection-diffusion equation with variable coefficient

For the case with variable convection coefficient only in z-direction, the same approach as in subsection 5.6.1 is taken where the problem is modified into a similar problem that can be solved with the direct FFT solver. In this instance, the modified problem is given by

$$\Delta u_h(\mathbf{x}) + \gamma_0(z)\frac{\partial}{\partial z}u_h(\mathbf{x}) = f(\mathbf{x})$$

This modified convection-diffusion equation would be solvable by the direct FFT solver as seen in subsection 4.2.5. on a uniform grid. Similar to subsection 5.6.1, one can show that the difference between the numerical solution of the original problem and the modified problem is bounded by $\|\gamma(\mathbf{x}) - \gamma_0(z)\|\|u(\mathbf{x})\|$.

All the examples above show that the solution obtained from the modified problem, u_h, will be a good approximation to solution u of the original PDE provided that the modified PDE closely resembled the original PDE.

5.6.3 Lower-order preconditioner scheme

This approach uses a strategy that combines the high-order compact approximation scheme and lower-order approximation preconditioner. Specifically, the matrix A is derived from a high-order compact scheme, while the preconditioner matrix A_p is derived from a lower-order compact scheme. This strategy was shown in [35] where a sixth-order explicit finite difference scheme is used with a second-order approximation preconditioner. This approach would be practical in scenarios where an efficient lower approximation solver already exists. Additionally, this approach higlights another advantage of the iterative method over the direct method. In particular, it is possible for a compact scheme to achieve the higher order

accuracy via the iterative methods.

In this dissertation, the proposed iterative method would implement the higher-order compact scheme with a lower-order approximation scheme for the preconditioner. Chapter 7 will present the numerical result for the accuracy and computational cost.

5.7 Implementation

5.7.1 PETSc

PETSc is a modular set of libraries, data structures, and routines developed and maintained by Argonne National Laboratory and a thriving community around the world [26]. It is designed to provide a parallel component environment for the implementation of parallel solvers for PDE based on MPI standards. PETSc libraries offer a variety of support for different numerical formulations, including finite element [31], finite volume [21], or finite difference methods. The wide array of fundamental tools for scientific computing makes PETSc an ideal environment for modeling scientific applications as well as for rapid algorithm design and prototyping. As of today, PETSc has been regarded as an efficient numerical simulation environment more than a sophisticated set of software tools.

Designed to be used by large-scale scientific applications, PETSc makes it possible to use GPUs, threads, and MPI parallelism in the same model for different effects, further optimizing code performance. In the proposed iterative algorithm, PETSc will be used for parallel data distribution model and MPI communications. Through PETSc, the proposed numerical scheme is expected to perform computation on large numbers of cores and computational

grid sizes. Additionally, PETSc also supports OpenMP and GPU acceleration, which will be useful for application optimization [24]. The strategies used to parallelize the proposed numerical scheme using the PETSc libraries will be presented in Chapter 6, with the result that the model attains a reasonable speed up and scale in a parallel environment.

PETSc Implementation

In this paper, PETSc will be used as a programming platform for the implementation of the proposed numerical scheme in combination with the preconditioner strategies discussed in Section 5.6. PETSc libraries contain KSP, a package of solvers for linear systems, which includes direct/iterative methods and several preconditioners. Among them is the KSPGMRES method that implements the default GMRES with a restart method and supports left-preconditioning. To implement the proposed iterative algorithm two modifications are made. First, the preconditioner in KSPGMRES is replaced with the preconditioner system of our own. Second the matrix-vector multiplication step, that is line 4 of Algorithm 5, is replaced with our version of matrix-vector multiplication to implement the high-order compact scheme presented in Chapter 3. Algorithm 6 shows how KSPGMRES was implemented in the proposed numerical solver.

Algorithm 6 KSPGMRES implementation

```
1: MatShellSetOperation(A,MATOP_MULT,(void(*)(void))my_Mat_Vec_Mult);
   //set up our own matrix-vector multiplication
2: KSPCreate(comm,&ksp);
3: KSPSetOperators(ksp,A,A_p);
   //set the system operator, the last input is the preconditioner
4: KSPSetType(ksp, KSPGMRES);
5: KSPSetInitialGuessNonzero(ksp,PETSC_TRUE);
6: KSPGMRESSetRestart(ksp, 100);
   //set the GMRES to restart every 100 iterations
7: KSPGetPC(ksp,&pc);
   //get the preconditioner object
8: KSPSetTolerances(ksp,tol,PETSC_DEFAULT,PETSC_DEFAULT,100);
9: PCSetType(pc,PCSHELL);
   //set the preconditioner object
10: PCShellSetApply(pc,mySolver);
   //applying our own preconditioner solver
11: PCShellSetContext(pc,(void*)&sys);
12: KSPSetPCSide(ksp,PC_LEFT);
   //set gmres with left preconditioning
```

For the proposed iterative algorithm, it is very convenient to use PETSc for verifying the performance of the numerical algorithm. Through the PETSc platform, there is no need to consider many details of implementation for the Krylov iteration itself such as the Arnoldi iteration. Users can instead focus on their preconditioner solver and the matrix-vector multiplication portion of Algorithm 5.

5.7.2 MATLAB

For numerical test problems on small grid sizes, the proposed iterative algorithm can also be implemented in MATLAB. MATLAB also has built-in functions such as gmres readily available for the sequential implementation of the iterative algorithm. Similar to PETSc implementation, the user may just pass in the functions for matrix-vector multiplication

and the preconditioner and left other tasks completed by MATLAB itself. This allows for ease in implementation for testing the sequential algorithm of the iterative solver on a local machine.

6 Parallelization

6.1 Introduction

In sequential computing, all the computations run one after another without overlapping, whereas, in parallel computing, computations are broken down into multiple, smaller, independent, often similar parts to be executed by multiple processors simultaneously. The results of each calculation are combined upon completion as part of an overall algorithm. The primary goal of parallel computing is to reduce computational time by fully capitalizing on all available computation power.

The sequential algorithm for the direct FFT solver presented in Section 4.4 is very expensive when the matrix A_p is large. Therefore, a standard implementation often turns out to be unsatisfactory, especially for real-time computations. To obtain high performance, we need to exploit the hierarchical parallelism of novel architectures such as multi-core, a cluster of multi-processors, and GPUs.

In general, an algorithm can be split into a parallelizable part and another which is not. Let S be the execution time to compute the part of the application that cannot be parallelized and B be the sequential time to compute the parallelizable part. The idea of parallelization would be to archive an execution time of $S+(B/p)$ on p processors. However, some overheads may occur and factors such as data latency or memory bandwidth would make the theoretical time slightly different. To achieve the best result for parallelization, it is crucial to divide the work or tasks of an application as evenly as possible among the processes.

This chapter aims to provide a general perspective on parallelization and the parallel toolbox that was used during the development of the proposed numerical algorithms. A detailed explanation of the parallel implementation of the developed model under various parallel programming environments will be presented in this chapter and the results of the implementation will be shown in the following chapter.

6.1.1 Types of parallel programming

With the rise of multi-processor systems, so does the variety of parallel programming. At large, these parallel programming techniques can be easily categorized based on which type of memory is used.

It is crucial to distinguish which type of memory is used on parallel computers as the reduction of both latency and bus contention are often determinant factors to their performance. The latency defines the time from the request to data access to the answer. Bus contention refers to the fact that more than one device attempt to simultaneously access memory.

There are two main categories of memory systems namely the shared memory system and the distributed memory system. A shared memory system relies on a centralized memory shared between all processors. In a shared memory system, a set of processors is linked to a memory system through an interconnection network, and a processor has access to each memory location without any software support. The interconnect can either connect all the processors to the main memory, or each processor has its local memory and has access to other memory locations.

On the other hand, distributed memory is located at different locations where each processor

does not have direct hardware access to this memory bank. In the distributed memory system, each processor or group of processors is paired with its private memory paired with an interconnection network. Distinct processors at various locations may explicitly communicate by sending messages to exchange data. Some well-known examples include clusters, composed of multiple nodes with their own local and private memory. Nodes are then linked by interconnection networks such as Ethernet or InfiniBand (IB).

For the implementation of the proposed numerical algorithm, OpenMP will be used as a reference to shared memory parallel programming while MPI would be the choice for distributed memory-based parallel programming.

6.2 OpenMP

6.2.1 Overview

OpenMP is the well-known standard in shared memory parallel programming. OpenMP is an implementation of multithreading, a method of parallelization whereby the system divides a task among a set number of threads. In OpenMP, threads refer to independent processing units that are capable of performing computations concurrently where every thread has access to the same memory.

Thanks to its shared memory architecture, OpenMP is an easy parallelization tool to implement but it is also restrictive. Programs using strictly OpenMP can only run on a single computer with shared memory. On a large multi-node cluster this typically restricts the parallel execution to just a single node with only 16 to 32 processors. More importantly,

OpenMP restricts the algorithm to a single node limiting its usage to the amount of random access memory (RAM) available on that node. In many situations, the computations that require vast amounts of RAM that are simply unavailable for OpenMP applications.

Despite its shortcoming, OpenMP has some promising features. OpenMP has a tool called runtime environment which automatically balances and allocates the workload to different threads. It is relatively simple to implement and offers an excellent speed-up in the execution of structured blocks.

In OpenMP, compiler directives are used to specify the computations that should be executed in parallel. This will create a team of threads and every thread will execute the parallelized section of code (or parallel region) independently. At the end of a parallel region, there is an implied barrier that forces all threads to wait until the computation inside the region has been completed. The C compiler directives are formed exclusively with #pragma omp *directive-specification*.

This key feature of OpenMP helps avoid the need to rewrite the entire program from scratch.

6.2.2 Implementation

Preconditioner direct FFT solver

OpenMP implementation of the direct solver discussed in Chapter 5 is accomplished primarily by dividing a for-loop iteration in Algorithm 2 among separate processing units. This can be easily implemented by simply adding the compiler directives before the for-loops. This key feature of implementation helps avoid the need to rewrite the entire program from scratch. Algorithm 7 shows how OpenMP was implemented for the preconditioner direct

solver portion of the proposed iterative method.

Algorithm 7 OpenMP Implementation of Direct FFT Solver

```
1: #pragma omp parallel {
2: #pragma omp critical
3: Create FFTW plan
4: #pragma omp for
5: for l = 1, ..., N_z do
6:    2D forward DST of the RHS in x−, y−direction
7:    Reorder the RHS to improve data locality
8: end for
9: #pragma omp for
10: for i = 1, ..., N_x · N_y do
11:    Solve the tridiagonal system A_i w_i = f_i using LU decomposition
12:    Reorder the solution vector back
13: end for
14: #pragma omp for
15: for l = 1, ..., N_z do
16:    2D reverse DST of the solution w in x−, y−direction
17: end for
18: }
```

The compiler directives #pragma omp parallel creates a parallel region that encapsulates the entirety of direct FFT Solver as seen in Algorithm 7. This is because creating a parallel region can be quite costly, especially for the iterative method. However an FFTW plan is a necessary component that sets up the calculation of the FFT and the creation of these plans is unfortunately not thread-safe, that is this process cannot be parallelized. So they must be created within a critical region within the parallel section of the code. This meant that this section is executed by a single thread at a time.

Matrix-vector multiplication

Algorithm 8 shows how OpenMP was implemented to parallelize the matrix-vector multiplication portion of the proposed iterative method, that is line 4 of Algorithm 5.

Algorithm 8 OpenMP Implementation of Matrix-Vector Multiplication

```
1: #pragma omp parallel for collapse(3)
2: for l = 1, ..., N_z; j = 1, ..., N_y; i = 1, ..., N_x do
3:     g_{i,j,l} = A v_{i,j,l}
4: end for
```

where A is the coefficient matrix that can be expressed as a 27-point stencil obtained from the discretization of the original PDE in Chapter 3. One can observe that the computation of each value of the result vector $\vec{g}$ is the scalar product between a row of A and the vector $\vec{v}$. This implies that each scalar product is independent and can be executed in any order and parallel. Thus the collapse clause was suggested to improve OpenMP performances. This will allows all three loops to be distributed evenly over all threads.

6.2.3 Conclusion

Overall, the simple structure and implementation of OpenMP implementation would suggest a promising result in terms of scalability. This would indicate that the OpenMP implementation is perfect for a small to medium-sized grid problem. The performance of OpenMP implementation will be shown and analyzed in the next chapter.

6.3 MPI

6.3.1 Overview

While OpenMP provides a very simple and efficient implementation for parallel programming, for a large enough computational grid, the memory required to allocate the necessary arrays can easily overrun the random access memory (RAM) available on a single node.

The natural solution to this problem is to distribute the working arrays and computational tasks between the nodes of a cluster. Several application programming interfaces (API) were developed for this, but the de facto standard for communication among processes today is the MPI.

In this section, a process (or MPI process) will refer to either an individual processor or a compute node of a cluster as both can be used in MPI. Each process in MPI is assigned a unique positive integer value starting with zero, called a rank. The rank is used to determine which calculations need to be executed in that process.

MPI enables a numerical solver on a large computational grid by dividing the computational domain across multiple nodes on a cluster. This effectively removes the memory limitation due to the use of a single node. MPI process no longer has access to the same memory. This meant that communication between processors is necessary for computation that requires memory from other processors. Communication here refers to the act of transferring data from one process to another. The size of the data required and the number of communication needed would impose a challenge to the MPI implementation.

6.3.2 Implementation

Unlike OpenMP, a program utilizing MPI does not have specific sections of parallelization. Each process runs the entire program and only communicates with one another when explicitly specified by the programmer. Therefore, the program must be modified to only perform the appropriate calculations and communicate the information back and forth as efficiently as possible.

The proposed numerical algorithm is very well suited for this type of parallelization. This is because each step of the algorithm can be carried out with a selected portion of the computational domain. To implement MPI into the numerical algorithm it is necessary to modify the sequential algorithm so that each node is allocated the minimum required memory on each.

The computational domain for the numerical algorithm is primarily divided as evenly as possible along the z-direction with some exceptions. Figure 6.1 illustrates how the computational domain is divided among three processes in the z-direction. Each process is associated with one color.

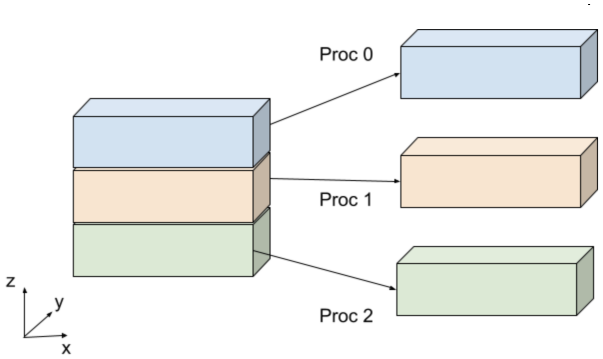

Figure 6.1: Computational domain divided among 3 processes for MPI implementation

The exceptions mentioned earlier were matrices and arrays required to solve the block tridiagonal system in Equation (4.6). Unlike other portions of the numerical solver, solving Equation (4.6) is dependent on the z-direction but independent in the x- and y-directions. As such all matrices and vectors used specifically for solving Equation (4.6) would be divided in the y-direction instead. There is no overlapping in terms of matrices as those matrices are only used exclusively for solving Equation (4.6). As for the right-hand side array, f, and the solution of the system w in Algorithm 2, duplication of these arrays with the right size

and data is required.

For the remaining section of this chapter, we shall denote NP as the total number of MPI processes used and $N_z(rank)$ be the grid size in z-direction obtained after dividing the computational domain for each process identified by its rank. The value of $N_z(rank)$ can be computed by

$$N_z(rank) = N_z/NP + (1 \text{ if } rank < (N_z \bmod NP) \text{ else } 0) \tag{6.1}$$

$N_y(rank)$ would be defined similarly replacing N_z in Equation 6.1 with the grid size in y-direction, N_y instead.

Preconditioner direct FFT solver

Algorithm 9 shows how the direct FFT solver was implemented with MPI.

Algorithm 9 MPI implementation of Direct FFT Solver

```
1: Create FFTW plan
2: for l = 1, ..., N_z(rank) do
3:    2D forward DST of the RHS in x−, y−direction
4:    Reorder the RHS to improve data locality
5: end for
6: Scatter the data via MPI to the appropriate process
7: for i = 1, ..., N_x · N_y(rank) do
8:    Solve the tridiagonal system A_i w_i = f_i using LU decomposition
9:    Reorder the solution vector back
10: end for
11: Scatter the data via MPI to the appropriate process
12: for l = 1, ..., N_z(rank) do
13:    2D reverse DST of the solution w in x−, y−direction
14: end for
```

The MPI implementation is almost identical to the sequential algorithm of the direct FFT solver except for changes to the size of the loop to reflect the changes in the computational domain. Additionally, there is a need for communication before and after solving the

tridiagonal system which is the main challenge of MPI implementation.

Data transfer in preconditioner direct FFT solver

Since the calculation of FFT and the solution of the tridiagonal system have a different dependency on the spatial directions, the MPI processes must communicate and transfer the required information to the appropriate processes. For the best performance out of MPI implementation, both the amount of communication and the size of the data transfer should be kept at their bare minimum.

Assuming the use of three processes for simplicity, Figure 6.2 illustrates the transfer of data between three processes in Algorithm 9.

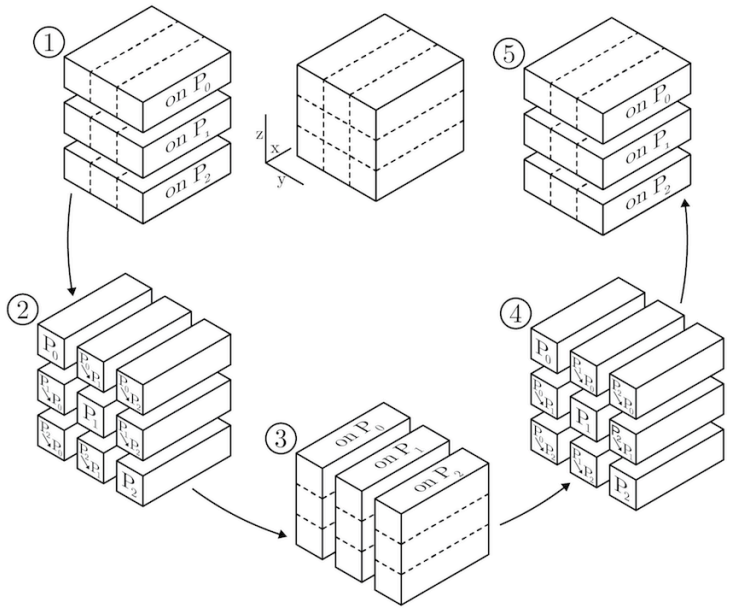

Figure 6.2: Data transfer between forward DST, solving tridiagonal systems and reverse DST steps

The first step shows how the domain is divided as evenly as possible among the three processes in the vertical, z-direction. This is where the forward transform is computed since the

calculations do not depend on the variable z. Once the FFT is computed, a certain portion of the computational domain is required to be sent to different processes as the tridiagonal solver is now dependent on the z-direction and independent in the y-direction.

To keep the size of the data transfer to each process at the bare minimum required, the second step shows how the domain on each process is further divided into smaller domains. Then each of the sections is sent only to the appropriate MPI process that requires it with commands such MPI_Scatterv.

All individual sections of the domain are then assembled on their respective process in the third step. Now the computational domain is divided in the y-direction. The solution to the tridiagonal system is then calculated. Once the solution vector $\vec{w}$ is obtained, the entire data transfer step is reversed. So that the computational domain can be divided in the z-direction again for the computation of the reverse transform.

Matrix-vector multiplication

Unlike OpenMP implementation, the vector $\vec{v}$ used for the matrix-vector multiplication step has been divided in the z-direction over separate nodes. This meant that to compute the matrix-vector multiplication communication between processes is required. Fortunately, the proposed numerical algorithm uses a compact scheme to formulate the coefficient matrix A. This meant that A is a block tridiagonal matrix and to calculate the matrix-vector multiplication it would be sufficient to have a specific portion of the data. In particular, for the case of three-dimensional problems, each process would only require the data of size $N_x \cdot N_y$ at the boundary layer of their neighboring processes.

Figure 6.3 illustrates the distributed matrix-vector multiplication for a problem of grid size

3^3 over three processes. All data with the same color are stored on the same processor. Each block would represent a 9×9 matrix and the block without any label are the zero matrices. Observe that for the first processor to complete matrix-vector multiplication it would only require the block label "u4" from the second processor. While the second processor would require the block label "u3" from the first processor and the block label "u7" from the third processor. Lastly, the third processor would only require the block label "u6" from the second processor.

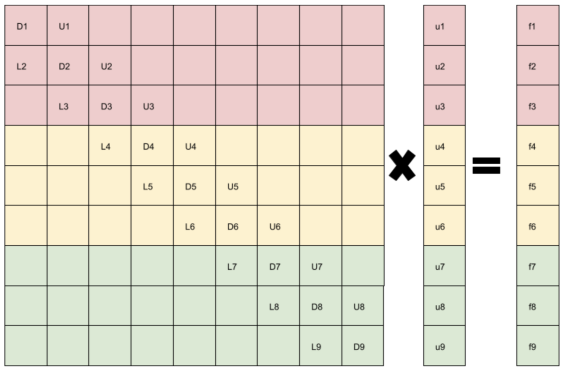

Figure 6.3: Distributed matrix-vector multiplication over three processors

Data transfer in matrix-vector multiplication

To optimize the performance of MPI implementations, each processor would only send the required data needed to the corresponding processor. The size of the vector $\vec{v}$ would also be increase to from $N_x \times N_y \times N_z(rank)$ to $N_x \times N_y \times (N_z(rank) + 2)$ which is denoted by $\hat{v}$. Some extra buffer arrays (tempSend and tempReceive) may be introduced as a temporary placeholder to assist with the data transfer.

Figure 6.4 shows the first step of the communication between three processors. In general, every processor beside the highest rank processor would duplicate the required data (last

$N_x \cdot N_y$ layer of the vector) and send the data to a processor with a rank one higher than itself.

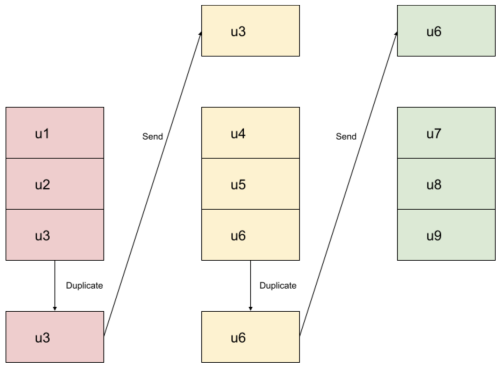

Figure 6.4: Data transfer in matrix-vector multiplication part 1

Figure 6.5 shows the next step for the communication between three processors. In this step, every processor beside the processor rank 0 would duplicate the necessary data (first $N_x \cdot N_y$ layer of the vector) and send the data to the processor with rank one lower than itself.

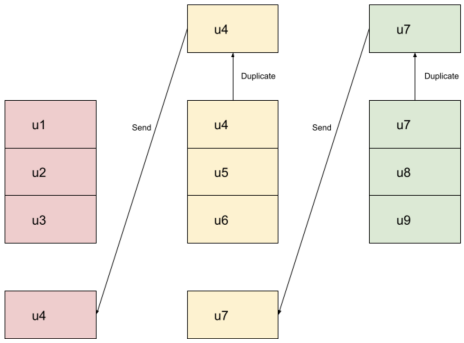

Figure 6.5: Data transfer in matrix-vector multiplication part 2

Then all the data are collected and organized into the extended array $\hat{v}$ as in Figure 6.6. The block label with 0 is the zero matrices so that the size of the array would be consistent across all processes. Now every process has access to all data required to perform matrix-vector multiplication in parallel.

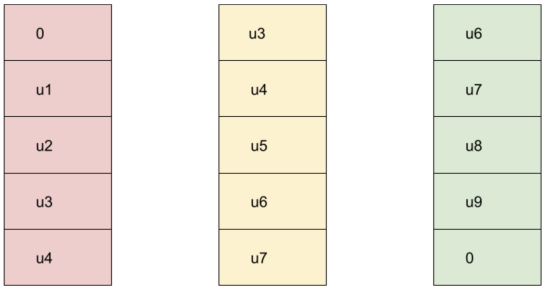

Figure 6.6: The data is stored in $\vec{v}$ for distributed matrix-vector multiplication over 3 processors. All data with the same color are on the same processor.

Algorithm 10 shows how MPI was implemented to parallelize the matrix-vector multiplication step of GMRES.

Algorithm 10 MPI Implementation of Matrix-Vector Multiplication

1: Duplicate the data at the highest level in $z-$ direction to tempSend
2: **for** $i = 0, \ldots, NP$ **do**
3: MPISend(tempSend, $rank + 1$);
4: MPIRecv(tempRecv, $rank - 1$);
5: **end for**
6: Store and organized the data to the extended array $\vec{v}$
7: Duplicate the data at the lowest level in $z-$ direction to tempSend
8: **for** $i = 0, \ldots, NP$ **do**
9: MPISend(tempSend, $rank - 1$);
10: MPIRecv(tempRecv, $rank + 1$);
11: **end for**
12: Store and organized the data to the extended array $\vec{v}$
13: **for** $l = 1, \ldots, N_z(rank); j = 1, \ldots, N_y; i = 1, \ldots, N_x$ **do**
14: Compute matrix-vector multiplication $\vec{g}_{i,j,l} = A\vec{v}_{i,j,l}$
15: **end for**

6.3.3 Conclusion

In conclusion, the implementation of MPI is harder than that of OpenMP implementation due to the necessity to manually have the processes communicate with each other. Nonetheless, thanks to its distributed memory architecture, the numerical solver is now capable of

solving problems on extremely large grid sizes.

6.4 Hybrid OpenMP-MPI

6.4.1 Overview

This section discusses an approach based on a combination of OpenMP and MPI implementation called the Hybrid OpenMP-MPI (or Hybrid OpenMP/MPI). From a user's perspective, the most convenient approach to any parallel programming is to ignore the hybrid approach and use a pure message-passing programming model. This is an effective approach since most MPI library developers have taken advantage of the shared memory within a node and optimized the intra-node communication [11].

However, the MPI implementation discussed in the previous section has an unexpected limitation on the preconditioner solver for the iterative solver. Since this algorithm uses $2D$ FFT, the algorithm only parallelizes in just one direction to avoid the additional need of sharing data across processes. That is, the operations are divided only in the z-direction in both the forward and reverse transforms and only in the y-direction for the tridiagonal solver. Since parallelization with MPI is accomplished through dividing the grid size, specifically N_y and N_z in the solver, the number of MPI processes that can be used will be limited by $N = \min(N_y, N_z)$. This would imply that the proposed algorithm can only achieve a maximum of N-times speed up even if there may be more than N processes available.

This is a potential issue when running on extremely large grid size problems. The issue becomes more evident upon the realization that as the grid size of the problems increases

cubically so does the memory and number of processes required, but the limitation N would only increase linearly. For example, suppose that each supercomputer has a total of 32 processes and it would require a minimum of two supercomputers to compute a 1000^3 size problem. Then to compute a 10000^3 size problem, we would require a minimum of 2000 supercomputers. Despite having access to all 2000 supercomputers with a total of $2000 \times 32 = 64000$ processes, the MPI implementation in Algorithm 9 is only capable of using 10000 MPI processes due to this limitation. In short, in this example, the MPI implementation could only take advantage of 15.63% of the given resources despite having additional resources available.

This leads to the development of hybrid implementation and how it would help remove this limitation. As a disclaimer, one can resolve this limitation in MPI by further parallelization in the y-direction when performing 2D FFT and in the x-direction for the tridiagonal solver at the cost of extensive communication time which is not ideal.

6.4.2 Implementation

Consider a cluster with p nodes each with q cores. To use the full power of a cluster, that is to utilize all $p \cdot q$ available cores, a combination of both the OpenMP and MPI tools into a hybrid program is required. In this approach, each node can represent an MPI process while the cores are used for the OpenMP threads. The computational domain would first be divided among the p MPI processes, then OpenMP allows all q threads to have the same access to all memory within the same MPI process. At this moment this approach has yet to remove the limitation imposed by N but it is worth to be considered.

This approach has an advantage over using the strictly MPI approach since it reduces the amount of communication between MPI processes by reducing the amount of MPI processes used. It is worth experimenting with and comparing the performance between this hybrid approach and MPI implementation. This will be the first approach to the hybrid OpenMP-MPI also referred to as Hybrid I in this dissertation.

To implement hybrid OpenMP-MPI, for the intended purpose of removing the limitation imposed by the grid, the idea is to use each parallel tool for different loops or sections of the algorithm. This approach is referred to as Hybrid II in this dissertation.

Preconditioner direct FFT solver

Algorithm 11 shows the implementation of Hybrid I. This is a simple direct OpenMP implementation applied to the MPI implementation shown in Algorithm 9.

Algorithm 11 Hybrid I implementation of Direct FFT Solver

```
#pragma omp parallel {
#pragma omp critical
Create FFTW plan
#pragma omp for
for l = 1, ..., N_z(rank) do
    2D forward DST of the RHS in x−, y−direction
    Reorder the RHS to improve data locality
end for
Scatter the data via MPI to the appropriate process
#pragma omp for
for i = 1, ..., N_x · N_y(rank) do
    Solve the tridiagonal system A_i w_i = f_i using LU decomposition
    Reorder the solution vector back
end for
Scatter the data via MPI to the appropriate process
#pragma omp for
for l = 1, ..., N_z(rank) do
    2D reverse DST of the solution w in x−, y−direction
end for
}
```

For the Hybrid II approach, MPI and OpenMP must be used to parallelize different sections of the algorithm. In particular, for the FFT transforms, MPI has been used to divide the computational domain in the z-direction, so OpenMP threads have to parallelize the $2D$ DST. This can be accomplished by using FFTW multi-threading [22]. As for the tridiagonal solver, this step is already independent of both x and y variable and the for-loops has been combined in all previous implementation. This implies that this step can be parallelized by $N_x \times N_y$ processes which are already above the limitation so there is no need for the extra procedure. The implementation of this modified approach is outlined in Algorithm 12.

Algorithm 12 Hybrid II implementation of Direct FFT Solver

1: Create a multi-threaded FFTW plan
2: **for** $l = 1, \ldots, N_z(rank)$ **do**
3: 2D forward multi-threading DST of the RHS in $x-, y-$direction
4: Reorder the RHS to improve data locality
5: **end for**
6: Scatter the data via MPI to the appropriate process
7: #pragma omp parallel for
8: **for** $i = 1, \ldots, N_x \cdot N_y(rank)$ **do**
9: Solve the tridiagonal system $A_i w_i = f_i$ using LU decomposition
10: Reorder the solution vector back
11: **end for**
12: Scatter the data via MPI to the appropriate process
13: **for** $l = 1, \ldots, N_z(rank)$ **do**
14: 2D reverse multi-threading DST of the solution w in $x-, y-$direction
15: **end for**

Matrix-vector multiplication

Since the limitation of the grid only exists on the preconditioner direct solver of the proposed numerical algorithm. There will only be one hybrid OpenMP-MPI implementation for the matrix-vector multiplication. Algorithm 13 shows the hybrid implementation of the matrix-vector multiplication. Due to the simplicity of the matrix-vector multiplication, there are not many changes besides applying the OpenMP implementation to the for-loop in line 13 of Algorithm 10.

Algorithm 13 Hybrid OpenMP-MPI Implementation of Matrix-Vector Multiplication

1: Duplicate the data at the highest level in $z-$ direction to tempSend
2: **for** $i = 0, \ldots, NP$ **do**
3: MPISend(tempSend, $rank + 1$);
4: MPIRecv(tempRecv, $rank - 1$);
5: **end for**
6: Store and organized the data to the extended array $\vec{v}$
7: Duplicate the data at the lowest level in $z-$ direction to tempSend
8: **for** $i = 0, \ldots, NP$ **do**
9: MPISend(tempSend, $rank - 1$);
10: MPIRecv(tempRecv, $rank + 1$);
11: **end for**
12: Store and organized the data to the extended array $\vec{v}$
13: #pragma omp parallel for collapse(3)
14: **for** $l = 1, \ldots, N_z(rank); j = 1, \ldots, N_y; i = 1, \ldots, N_x$ **do**
15: Compute matrix-vector multiplication $\vec{g}_{i,j,l} = A\vec{v}_{i,j,l}$
16: **end for**

6.4.3 Conclusion

The author and co-researchers are interested to compare the performance of strictly MPI implementation and Hybrid I implementation on a problem size not limited by N. Numerical experiments would be conducted in the next chapter to present the results. The Hybrid II implementation is expected to deliver excellent results on large grid sizes by being able fully to utilize all available resources. However, it is worth mentioning that the FFTW's multi-threading support used in Hybrid II implementation does not necessarily translate into performance gains since there is an overhead required for synchronization that may outweigh the computational parallelism [22]. Therefore, one can only benefit from threads if the problem has a sufficiently large grid size. When this method was tested in OpenMP environments on medium to small grid problems, the overhead drastically increased. Nonetheless, for the hybrid implementation of the solver on relatively large grids, Hybrid II implementation has

some significant advantages. The approach alleviates a critical restriction on the number of MPI processes used by the direct FFT solver in the distributed memory environment. So, in the hybrid algorithm for large grids (test problem with grid larger than 1000^3), the Hybrid II implementation was preferable while Hybrid I implementation is applied on medium to small grid problems (test problem with grid around or less than 500^3).

7 Numerical Results

This chapter will present the results of numerical experiments which will demonstrate the quality of the proposed parallel numerical algorithm. The proposed algorithm is implemented in the C programming language unless stated otherwise. A majority of the numerical experiments were conducted on a standard Alienware desktop with an Intel Core i7, 2.8 GHz processor, and 16 GB of RAM. The result of computation on a large grid size problem, however, was obtained from a run on two different supercomputer clusters, these include the Falconviz clusters at Idaho National Lab (INL) and the Cori cluster at Lawrence Berkeley National Laboratory (LBNL). The Falconviz compute nodes contain dual Intel Xeon E5-2695 v4 2.10 GHz processors with 18 cores each and a total of 128 GB of memory. The Cori supercomputer consists of Haswell nodes where each node has two sockets, each socket is populated with a 16-core of Intel Xeon 2.3 GHz processor for a total of 32 physical cores with 128 GB DDR4 2133 MHz memory.

7.1 Direct FFT Solver

First, we demonstrate the efficiency of the developed direct solvers in the case of a three-dimensional test problem. In the section, we will be using the following measures related to the computed approximate solution u_h and analytic solution u of the test problems.

- L_2**-res** (the relative L_2 residual) is $\|Au_h - f\|_2$
- L_2**-err** (the relative L_2 error) is $\|u - u_h\|_2/\|u\|_2$

- **max-err** denotes the maximal component-wise error: $\|u - u_h\|_\infty$

For an Nth-order convergence rate, the errors would decrease approximately by a factor of 2^N when the grid size is double. The **rate of convergence** for the second, fourth, and sixth order scheme is calculated by $\ln\left(\frac{|err_{m,N}|}{|err_{2m,N}|}\right) / \ln(2)$ where $err_{m,N}$ is the max-err obtained from PDE problem with Nth-order scheme on a grid of m^3. Hence for an Nth-order compact scheme, the rate of convergence is expected to be N.

7.1.1 Solution of Helmholtz equation

To illustrate the quality of the developed direct methods, the numerical experiment is done on the test problems recently published in [5]. In [5], the authors considered the solution of the 3D Helmholtz problem using iterative block-parallel CARP-CG method [4]. We shall compute the solution to the same problem by applying the direct FFT solver presented in Chapter 4.

Test Problem 1

For the first test problem, the coefficient k is defined by

$$k(z) = a - b\sin(cz) \qquad \text{with } a > b \geq 0$$

Since the coefficient only depends on one spatial variable, the direct FFT solver can be used to obtain the approximate solution of the Helmholtz equation with the following analytic solution

$$u(x, y, z) = \sin(\beta x)\sin(\gamma y)e^{-k(z)/c}, \text{ where } \beta^2 + \gamma^2 = a^2 + b^2$$

define over the domain $\Omega = [0, \pi] \times [0, \pi] \times [0, \pi]$. The number of grid points in all directions will be equal to include a comparison for the sixth-order scheme. For this test problem, the right-hand side is given by

$$f(x, y, z) = \Delta u(x, y, z) + k^2(z)u(x, y, z) = -b(2a + c)\sin(\beta x)\sin(\gamma y)\sin(cz)e^{-k(z)/c}$$

All numerical experiments in this section will be performed using this test problem with the following parameter: $a = 10, b = 9, c = 10, \beta = 10$, and $\gamma = 9$ unless stated otherwise.

Convergence of the numerical scheme

To demonstrate the convergence of the proposed numerical scheme, a series of tests was conducted on test problems with various grid sizes, that is $125^3, 250^3$, and 500^3. For this test, all implementations (sequential and parallel) can give consistent results on all grid sizes considered with different sets of processing units. Table 7.1 shows the convergence of the second-, fourth- and sixth-order implementations, respectively.

Table 7.1: Test for convergence for direct FFT solver on Test Problem 1

Grid	Scheme	max-err	L_2-err	L_2-res	Rate of Convergence
125^3	2nd	5.7570466e-03	6.4986713e-03	4.7269292e-13	-
	4th	3.4493268e-05	3.5925614e-05	3.6301725e-13	-
	6th	2.1875397e-06	1.9909214e-06	3.1581209e-13	-
250^3	2nd	1.4853854e-03	1.6510028e-03	2.5846930e-12	1.95
	4th	2.1782070e-06	2.2582699e-06	1.9857221e-12	3.99
	6th	3.4942928e-08	3.1643311e-08	2.0541112e-12	5.97
500^3	2nd	3.7448165e-04	4.1516358e-04	6.5883688e-12	1.99
	4th	1.3726414e-07	1.4187594e-07	5.0832056e-12	3.99
	6th	5.5211108e-10	4.9939925e-10	5.2147803e-12	5.98

The presented outcomes of the numerical experiments confirm the declared rate of conver-

gence for each corresponding approximate solution.

Comparison with existing iterative solver

Table 7.2 presents a comparison of various solvers: the first two rows show the result of the iterative solver used in [5], with results from runs on a Supermicro cluster consisting of 12 nodes. Each node contained two Intel Xeon E5520 2.27 GHz quad processors that shared 8 GB of memory.

The remaining rows present the results of the second-order direct solver considered in our previous publications [36, 34], and fourth and sixth-order solvers presented in Chapter 5. Rows three to five give results for the Xeon X5690 server with a 3.47 GHz processor and 144 GB of memory. While the last three rows exhibit the results achieved on an iMac PC with Intel Quad-Core i7 2.93 GHz processor and 16 GB 2133 MHz LPDDR3 memory.

The first column represents the hardware used in the numerical experiment. The second and third columns indicate the order of approximation of the solver and the type of the solver (direct or iterative). In the fourth column, the number of grid points required to achieve the indicated relative accuracy (L_2**-err** < 0.001). The fifth column shows the number of iterations until the convergence of the iterative solver, where in the case of the direct solvers we put 1. The last column displays the CPU time required for each test run.

Table 7.2 shows that the direct FFT solver on the X5690 and iMac i7 was approximately 46 and 36 times faster than the iterative solver respectively in the second-order case. With the sixth-order scheme, the direct FFT solver was approximately 18 and 13 times faster on the X5690 and iMac i7 than the iterative solver, respectively. This is an expected result since the iterative CARP-CG method was designed to solve the general 3D Helmholtz equation

Table 7.2: Comparison of direct FFT solver and other iterative solvers on Test Problem 1

Machine	Scheme	Type	N	# Iterations	Time(s)
Supermicro	2nd	iterative	333	1970	703
Supermicro	6th	iterative	45	350	1.01
X5690	2nd	direct	353	1	15.18
X5690	4th	direct	62	1	0.078
X5690	6th	direct	50	1	0.055
iMac i7	2nd	direct	353	1	19.8
iMac i7	4th	direct	62	1	0.097
iMac i7	6th	direct	50	1	0.08

instead of the problems with specific restrictions considered in this test problem. Table 7.2 also indicates to reach the desired accuracy on the approximation scheme the direct solver requires more slightly larger grid size. This is not a detriment to the direct solver as due to the optimality condition of the FFT method, sometimes it is advantageous to consider a slightly larger number of grid points which has more factors of 2 in its prime factorization. The CPU time for the fourth-order scheme on the 64^3 grid was 0.07 sec on X5690.

7.1.2 Solution of convection-diffusion equation

To demonstrate the robustness of our approach, this subsection presents the result for computing the numerical solution of the 3D convection-diffusion equation (5.6) using the direct FFT solver.

Test Problem 2a

For the next test problem, we consider the 3D convection-diffusion equation with the following analytic solution.

$$u(x,y,z) = \sin\left(\frac{\pi x}{\sqrt{2}}\right)\sin\left(\frac{\pi y}{\sqrt{2}}\right)e^{-\gamma z/2}\frac{2e^{\gamma/2}\sinh(\sigma z)+\sinh(\sigma(1-z))}{\sinh\sigma}$$

where $\sigma = \sqrt{\pi^2+\gamma^2/4}$ define over the domain $\Omega = [0,\sqrt{2}]\times[0,\sqrt{2}]\times[0,1]$. Let $\gamma = -100$ and the right-hand side is given by

$$f(x,y,z) = \Delta u(x,y,z) + \gamma\frac{\partial}{\partial z}u(x,y,z) = 0$$

It is worth mentioning that the step size $h_x = h_y = \sqrt{2}/(N+1)$ and $h_z = 1/(N+1)$ for Test Problem 2a are not equal. The direct FFT solver will be using the stencil shown in subsection 4.2.4 for Test Problem 2a.

Convergence of the numerical scheme

The fourth-order convergence of the approximate solution to the analytic solution on a sequence of grids is presented in Table 7.3. As expected the rate of convergence of the maximum error verified the accuracy of the numerical method.

Table 7.3: Test for convergence for direct FFT solver on Test Problem 2a

Grid	max-err	L_2-err	L_2-res	Rate of Convergence
64^3	3.2612907e-03	4.6813690e-04	1.5435312e-15	-
128^3	2.0579387e-04	2.9792890e-05	4.6094565e-15	3.99
256^3	1.2939970e-05	1.8507601e-06	9.9002420e-15	3.99
512^3	8.1975702e-07	1.1559163e-07	3.2599029e-14	3.98

Test Problem 2b

For the next test problem on the convection-diffusion equation, we consider the 3D convection-diffusion equation with $\alpha = \beta = 0$ and $\gamma(z) = Re\cos(z)$ where $Re = 100$ is the Reynolds number. The analytic solution is given by

$$u(x, y, z) = \sin(x)\sin(y)\sin(z)$$

define over the domain $\Omega = [0, 1] \times [0, 1] \times [0, 1]$. In this test problem, the right-hand side would be

$$\begin{aligned} f(x, y, z) &= \Delta u(x, y, z) + \gamma(z)\frac{\partial}{\partial z}u(x, y, z) \\ &= -3\sin(x)\sin(y)\sin(z) + \gamma(z)\sin(x)\sin(y)\cos(z) \end{aligned}$$

$\gamma(z)$ in this test problem is now a function of z so the direct FFT solver will be using the coefficient shown in subsection 4.2.5.

Convergence of the numerical scheme

The fourth-order convergence of the approximate solution to the analytic solution on a sequence of grids is presented in Table 7.4. Similar to the previous test, the result from the numerical runs verified the accuracy of the numerical method.

Table 7.4: Test for convergence for direct FFT solver on Test Problem 2b

Grid	max-err	L_2-err	L_2-res	Rate of Convergence
64^3	6.31161e-08	7.58421e-15	1.48463e-07	-
128^3	4.07262e-09	1.40755e-14	9.43177e-09	3.95
256^3	2.58605e-10	1.46549e-14	5.94238e-10	3.98

7.1.3 Scalability of direct FFT solver

Subsection 7.1.1 has displayed the strength of the developed direct FFT solver over one of the most well-known general iterative methods. In this subsection, we focus on the scalability properties of the developed algorithms and their limitations. The parallel implementation of the direct FFT solver introduced in Chapter 6 will be used to compute approximate solutions to the same test problem shown in the previous subsection. For the remaining of Section 7.1, the numerical experiment would be run with Test Problem 1 from subsection 7.1.1 with the following parameter: $a = 10, b = 9, c = 10, \beta = 10$ and $\gamma = 9$ and the sixth-order scheme, unless stated otherwise.

First, a comparison between the OpenMP and MPI implementation was made with a numerical test on the grid of 512^3. The results from both implementations were obtained from a run on a single Haswell node on Cori. The sets of OpenMP threads and MPI processes were also chosen to be the same and the results of this comparison are presented in Figure 7.1.

Figure 7.1 shows the solution time required for both of the parallel implementations demonstrates near linear scalability. The solution wall time (in seconds) decreases by a factor of close to 2 as the number of processes doubles. This numerical test also supported the idea that OpenMP has slightly better performance than MPI on a single node.

However, the limitation of OpenMP is revealed while attempting to run the test problem on a grid size of 1024^3. The machines tested, including a single Haswell node on Cori, were unable to run this experiment due to a lack of memory. The experiment was repeated with the MPI implementation on one, two, and four Haswell nodes on Cori. The attempts with both one

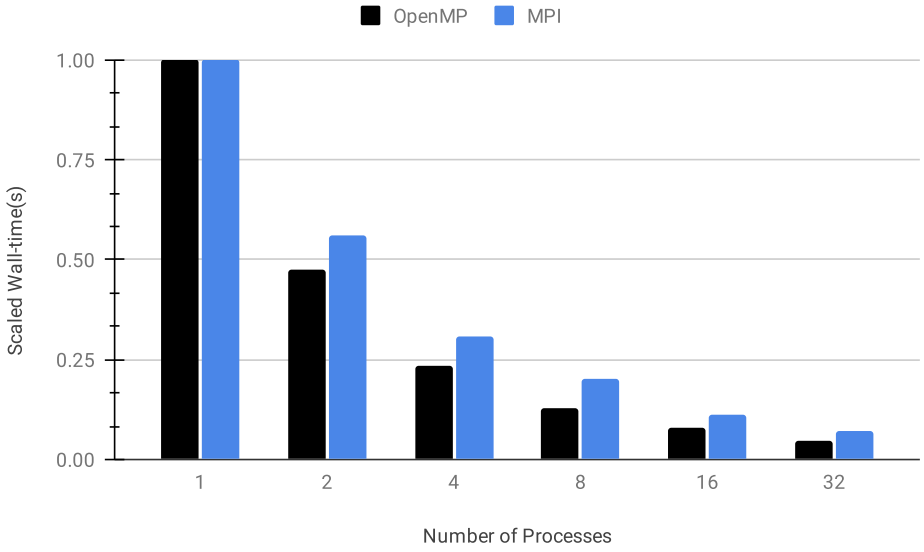

Figure 7.1: Scalability of OpenMP vs MPI implementation of direct FFT solver

and two nodes failed, again due to a lack of memory. However, the program was able to run successfully with four Haswell nodes. This highlights the power of MPI implementation.

An experiment was conducted to further investigate the performance and limitations of the MPI implementation. The setup, communication, and computation times were recorded as the number of MPI processes used increased. Computation time here refers to the time taken for all the MPI processes to complete the forward and inverse 2D DST, and tridiagonal solver steps. The communication time measures the longest time taken for the MPI processes to scatter the data to the appropriate processes and assign data to a local array, while the setup time gives the time required to prepare the parallel environment. The total time is the sum of both of these run times. This test is run on a grid size of 512^3 on Cori as well as on Falconviz and the results are displayed in Figure 7.2 and 7.3. The time shown in both figures is measured in seconds (s) and then apply the natural logarithmic.

One can observe that the computation time decreases almost linearly with a slope close to

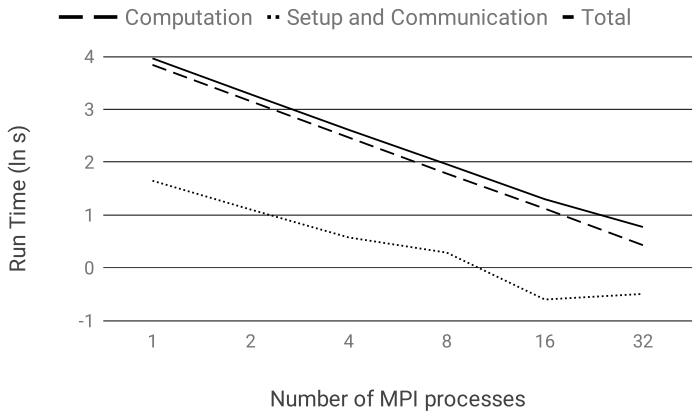

Figure 7.2: MPI implementation performance on Cori

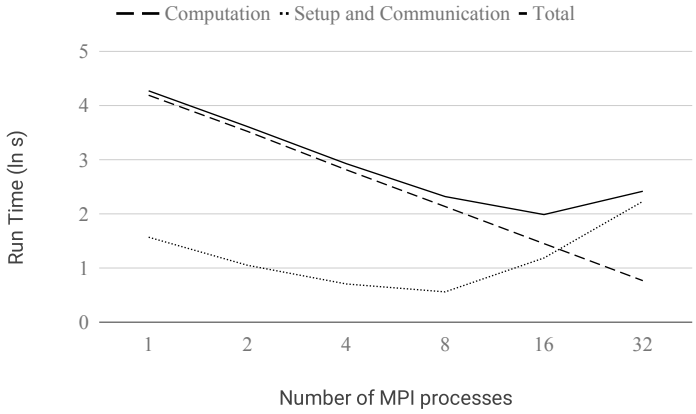

Figure 7.3: MPI implementation performance on Falconviz

-0.5 in both figures. It is worth noting that the setup and communication time improved significantly from 8 MPI processes to 16 MPI processes on Cori. The author suspects that the increase in performance could be attributed to the computation domain being divided small enough to fit into the cache. Thus making the data transfer process faster. This trend improvement in computation time is observed throughout all numerical tests for the hybrid implementation of the direct and iterative solvers on Cori on the same grid.

On the other hand, the communication time would show a trend of decreasing and then

started to increase past a certain threshold (16 for Cori and 8 for Falconviz). Past the threshold, the setup and communication time is seen to be increasing and even exceeding the computation time (as seen with the case on Falconviz). This numerical test has successfully demonstrated the weakness of MPI implementation when the increase in the number of MPI processes causes the overhead to increase and outweighs the computational parallelism.

To overcome this hurdle from MPI implementation, the hybrid OpenMP-MPI approach is considered. A sequence of test runs was performed to verify the hypothesis that once the number of MPI processes reaches a certain threshold, reducing the number of MPI processes used while maintaining the number of physical processors utilized will improve the computation time over MPI. Table 7.5 shows the computation times in seconds for Hybrid I implementation of direct FFT solver on a grid of 512^3 on Cori. Similarly, Table 7.6 shows the computation times for Hybrid II implementation instead of on the same grid. The nodes in the table indicate the number of Haswell nodes the solver is using which also serves as the number of MPI processes used. It is worth reiterating that each Haswell node has 32 physical cores available which serve as the number of OpenMP threads. Here, the number of OpenMP threads changes horizontally, and the MPI processes change vertically.

Table 7.5: Hybrid I implementation of direct FFT solver on Test Problem 1

Nodes	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads
1	39.61	24.56	15.04	10.78	8.88	8.07
2	19.90	11.90	7.19	4.97	4.00	3.44
4	10.31	6.11	3.66	2.60	2.28	1.90
8	5.30	3.34	2.09	1.52	1.30	1.22
16	2.42	1.48	0.85	0.55	0.47	0.41
32	1.81	1.35	1.06	0.86	0.77	0.86

Table 7.6: Hybrid II implementation of direct FFT solver on Test Problem 1

Nodes	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads
1	39.61	21.73	12.93	8.83	6.84	5.62
2	19.87	10.94	6.48	4.56	3.48	2.92
4	9.99	5.66	3.48	2.52	2.10	1.82
8	5.27	3.11	1.99	1.55	1.37	1.27
16	2.49	1.42	0.85	0.64	0.58	0.58
32	1.85	1.33	1.09	0.93	0.76	0.80

In general, both of the hybrid implementations produce similar results to each other on medium size grids. It is worth noting that the result of using 16 nodes and 2 threads appears to have a slight edge over using 32 nodes and 1 thread (which is comparable to a strictly MPI implementation) in both hybrid implementations. This small edge in performance could be observed on the last two rows when comparing the result diagonally down and to the left in both Table 7.5 and 7.6. This could support the hypothesis mentioned earlier that once the number of MPI processes reaches a certain threshold then reducing the number of MPI processes used while maintaining the number of physical processors utilized will improve the computation time over MPI. This hypothesis is also supported by Figure 7.2 which shows that the computation time begins to increase when users increase the amount of MPI processes used from 16 to 32.

Nonetheless, the true strength of the hybrid approach lies in a problem with relatively large grids. To highlight the strength of hybrid OpenMP-MPI implementation, further experiments were conducted on a sequence of larger grids ranging from 512^3 up to 4096^3. The results of these experiments were obtained from a run on Cori as well. The performance of the MPI and Hybrid II implementation of direct FFT solver is shown in Table 7.7. The

first column represents the number of grid points used in the experiment. The second column indicates the type of method used for the parallel implementation. The third and fourth columns represent the number of Haswell nodes used and the total number of processors used by the numerical solver. For the MPI implementation, the total number of processes is equivalent to the number of MPI processes used, while for Hybrid II implementation, the number of nodes represents the number of MPI processes used and each node has 32 cores which serve as the number of OpenMP threads used. The CPU time required for each test run is recorded in seconds and presented in the last column.

Table 7.7: Comparison of MPI and Hybrid II implementation performance

N	Method	Nodes	# Processes	Time(s)
512	MPI	1	32	2.830525
	Hybrid II	1	32	7.793963
1024	MPI	4	128	8.759851
	Hybrid II	4	128	16.911352
2048	MPI	32	1024	40.465395
	Hybrid II	32	1024	19.417831
4096	MPI	256	4096*	445.803343
	Hybrid II	256	8192	27.522366

* Despite having 8192 processors available the implementation limits the use to 4096

One may observe that while running the numerical solver on a medium grid size the MPI implementation performed better than the hybrid implementation. This is likely due to the overhead required by OpenMP, in particular of those multi-threading FFTW, as discussed in sub-subsection 6.4.2. However, the Hybrid II implementation was able to outperform the MPI implementation by almost a factor of 2 when working with a grid size of 2048^3. This could be an indication of how the setup and communication time will become a bottleneck

for MPI implementation alluded to by Figure 7.2. The result on the grid size of 4096^3 then demonstrates another limitation of the MPI implementation for the proposed algorithm. As mentioned in Section 6.4 the MPI implementation is only able to utilize at most 4096 processors which is the grid size in z-direction. The Hybrid II implementation, however, significantly alleviates this restriction and can fully utilize all available resources almost doubling the number of processes used in MPI implementation. As a result, the Hybrid II implementation significantly outperforms the strictly MPI implementation by 16 times.

All in all, while restricted to specific types of problems, the direct FFT solver managed to represent highly accurate and scalable methods for the solution of the considered problems. These features make the direct FFT solver desirable as the preconditioner solver for the proposed iterative method.

7.2 Iterative GMRES-FFT Solver

In this section, numerical experiments for the proposed iterative method will be presented. Let u_k represent the computed approximate solution after the k-th iteration of the iterative solver and u_0 represent the initial guess of the iterative solver. The proposed iterative solver set the $u_0 = 0$ as the initial guess and computes $\vec{r}_0$ by solving $\vec{r}_0 = A_p^{-1} f$ with the direct FFT solver where A_p represents the preconditioner matrix.

This section will be using the following measurements to analyze the performance of the iterative method

- the relative residual is $\|Au_n - f\|_2 / \|\vec{r}_0\|_2$
- **max-err** denotes the maximal component-wise error: $\|u - u_n\|_\infty$

where the iterative method stops at the n-th iteration and u is the analytic solution of the test problem (if any). The rate of convergence is calculated by $\ln\left(\frac{|err_{m,N}|}{|err_{2m,N}|}\right)/\ln(2)$ where $err_{m,N}$ is the max-err obtained from PDE problem with Nth-order scheme on a grid of m^3 where an analytic solution is known as in Section 7.1.

The first two subsections will focus on test problems based on the 3D Helmholtz problem and the 3D convection-diffusion problem. Test problems that are not solvable with the direct FFT solver will be presented. The third subsection will present the application of the developed iterative method for the forward problem of the subsurface scattering problem. Each of the subsections would also include a sub-subsection to discuss the preconditioning system used. The next section would present the performance of the iterative method when a lower-order scheme preconditioner is used. The scalability of the proposed iterative method would be displayed via the performance of the parallel algorithm at the end of the section.

7.2.1 Solution of Helmholtz equation

In this subsection, we shall consider the 3D Helmholtz equation where the coefficient function k^2 is a multivariate function dependent on the variables x, y, and z. Hence the matrix from the discretization of this Helmholtz equation would fail the restriction for direct FFT solver introduced in Chapter 4. This meant that the direct solver could not be used to find the numerical solution for the Helmholtz equation. All numerical experiments in this subsection were conducted on a standard Alienware desktop described earlier in this chapter.

Test Problem 3

Define the coefficient k^2 as

$$k^2(x, y, z) = -2\left(\frac{1}{x^2} + \frac{1}{y^2} + \frac{1}{z^2}\right)$$

Then we shall considered the 3D Helmholtz equation (2.1) with the following analytic solution

$$u(x, y, z) = \frac{1}{xyz}$$

defined over the domain $\Omega = [0.5, 2.5] \times [0.5, 2.5] \times [0.5, 2.5]$. The numerical experiments will be running with an equal number of grid points in all directions to include a comparison with the sixth-order scheme. For this particular test problem, the right-hand side function would be the zero function, that is $f(x, y, z) = \Delta u(x, y, z) + k^2(x, y, z)u(x, y, z) = 0$.

Preconditioning system

As mentioned in Section 5.6, the preconditioner plays a huge role in the performances of the iterative method. We shall consider the preconditioning strategy discussed in subsection 5.6.1 by approximating the coefficient $k^2(x, y, z)$ in Test Problem 3 with $k_0^2(z) = -2/z^2$. Then the modified Helmholtz equation for the preconditioning system

$$\Delta u(x, y, z) + k_0^2(z)u(x, y, z) = 0$$

could be solved with the direct FFT solver. The iterative process is set to stop once the relative residual falls below 10^{-12}.

Convergence of numerical scheme

To demonstrate the convergence of the proposed iterative method, a series of numerical tests were conducted on Test Problem 3 with various grid sizes, that is $32^3, 64^3, 128^3$, and 256^3.

The results of this series of numerical tests are presented in Table 7.8. The first column of Table 7.8 displays the grid size $N = N_x = N_y = N_z$ of the grids used for the numerical experiments. Column two of the table would display the order of the scheme used and column three presents the number of iterations required before the desired error is obtained. The fourth column reports the maximal component-wise error of the initial guess and the analytic solution. The max-err is then reported in the fifth column. Also shown is the sequential CPU time (in seconds) for the iterative algorithm to complete in the sixth column. Finally, the last column of the table shows the rate of convergence.

Table 7.8: Test for convergence on 3D Helmholtz problem with variable coefficient function

N	Scheme	Iteration	$\|u_1 - u\|_\infty$	max-err	Time(s)	Rate of Convergence
32	2nd	10	0.437	4.12e-03	0.05	-
	4th	10	0.433	4.71e-05	0.06	-
	6th	10	0.436	5.62e-07	0.08	-
64	2nd	10	0.439	1.06e-03	0.42	1.96
	4th	10	0.438	3.05e-06	0.51	3.95
	6th	10	0.439	9.03e-09	0.70	5.96
128	2nd	10	0.439	2.67e-04	3.99	1.99
	4th	10	0.439	1.91e-07	4.60	4.00
	6th	10	0.440	1.42e-10	6.11	5.99
256	2nd	10	0.440	6.69e-05	35.09	2.00
	4th	10	0.440	1.20e-08	40.46	3.99
	6th	10	0.440	2.50e-12	53.09	5.83

Recall that $u_1 = u_0 + A_p^{-1}\vec{r}_0 = A_p^{-1}\vec{r}_0$ is in fact the solution to the system $A_p x = f$ obtained with the direct FFT solver. Hence the fourth column of Table 7.8 is the undeniable proof that the direct FFT solver fails to find a numerical solution for Test Problem 3. On the other hand, the fifth column shows that through the iterative method, one may obtain a

significantly better approximation to the analytic solution. Furthermore, the last column of Table (7.8) displays strong evidence that when the grid size is doubled, the max-err obtained is reduced by approximately 2^n where n denotes the order of scheme used. This confirmed the declared rate of convergence for each corresponding approximate scheme.

It is worth mentioning that while all three schemes require the same number of iterations, the max-err would suggest that the sixth-order scheme has a significantly better result. In particular, one can observe that the second order scheme with a grid of 256^3 has an error that is about 120 times larger than that of the sixth order scheme with a mesh of 32^3 (i.e. 1/512 of the number of grid points of the second order scheme). On the other hand, the sixth-order accurate scheme with a grid of 64^3 solves the Helmholtz equation with accuracy approximately the same as the fourth-order accurate scheme on a mesh of 256^3. This meant that in terms of computational time, the sixth-order scheme is capable of obtaining a solution of the same accuracy at about 58 times faster than that of the fourth-order scheme.

In conclusion, these numerical experiments demonstrate the convergence properties of the proposed GMRES-FFT-type iterative method for the general 3D Helmholtz equation. It also displayed the advantages of using a sixth-order accurate scheme over the other two schemes.

7.2.2 Solution of convection-diffusion equation

In this subsection, we shall consider the 3D convection-diffusion problem but with non-zero convection coefficients α and β instead. The matrix from the discretization of this problem is expected to fail the restriction for the direct FFT solver. The results of the numerical experiments in this subsection are obtained from runs on the standard Alienware desktop

described earlier.

Test Problem 4a

The accuracy of the finite difference scheme is first tested. To illustrate this, we used the same numerical experiments in [23] where the following 3D convection-diffusion equation with constant convection coefficients is consider

$$\alpha = -Re\cos(A)\cos(B)$$

$$\beta = -Re\cos(A)\sin(B)$$

$$\gamma = -Re\sin(A)$$

$$u(x, y, z) = \sin(\pi x)\sin(\pi y)\sin(\pi z)$$

with $A = 35^\circ$ and $B = 45^\circ$ define over $\Omega = [0, 1] \times [0, 1] \times [0, 1]$. This numerical test is repeated for small to moderate values of the Reynolds number ($\mathrm{Re} \leq 10^3$) and various step sizes as in [23].

Preconditioning system

As discussed in subsection 5.6.2, the preconditioning system considers approximating the original test problem with the following 3D convection-diffusion equation

$$\Delta u(x, y, z) - \gamma \frac{\partial}{\partial z} u(x, y, z) = f(x, y, z)$$

whose discretization is shown in subsection 4.2.4. Direct FFT solver can solve this system directly and the iterative process is set to stop once the relative residual is below 10^{-10}. The result of the numerical experiments was obtained from runs on a standard Macbook Pro laptop with a 2.3 GHz Quad-Core Intel Core i5. The numerical algorithm was implemented

in MATLAB.

Convergence of numerical scheme

Table 7.9 presents the results for the test of fourth-order convergence of the approximate solution to the analytic solution. The first column shows the value of the Reynolds number used for the numerical test. The step size of the experiment, $h = 1/(N+1)$ and max-err is reported on the second and third column of the table respectively. The last column of the table displayed the rate of convergence.

Table 7.9: Test for convergence of 3D convection-diffusion equation with constant coefficients

Re	h	max-err	Rate of Convergence
1	1/16	1.5014e-05	-
	1/32	9.4463e-07	4.00
	1/64	5.9003e-08	4.00
10	1/16	5.7296e-05	-
	1/32	3.5854e-06	4.00
	1/64	2.2409e-07	4.00
100	1/16	1.5857e-03	-
	1/32	1.0279e-04	3.95
	1/64	6.4460e-06	4.00
1000	1/16	9.0426e-03	-
	1/32	1.1538e-03	2.97
	1/64	8.3612e-05	3.79
	1/128	5.3502e-06	3.97

Overall the result shown in Table 7.9 were nearly identical to the result shown in [23]. Numerical tests with $Re = 1, 10$, and 100 have proven the fourth-order convergence rate for the proposed iterative method. The fourth-order scheme performed relatively poorly at $Re = 1000$ on a small grid. However, as the grid is increased to allow more grid points

inside the computational domain, the accuracy of the fourth-order compact scheme increases rapidly.

Comparison with existing numerical solver

Table 7.10 presents the comparisons between the proposed iterative solvers and the numerical results from [23]. The authors of [23] compute the numerical solution for Test Problem 4a using the four-color Gauss-Seidel and backward relaxation technique with the best results. The numerical experiment results were obtained from runs on a C-90 supercomputer at the Pittsburg Supercomputing Center. The C-90 is a vector machine with 16 processors.
The method used in the experiments is shown in the first row of the table. The iterations column displays the number of iterations required for the iterative methods to converge while the CPU time shows the computational time measured in seconds. The numerical result for $h = 1/32$ and $h = 1/64$ with various values of the Reynolds number, $Re = 1, 10, 100$, and 1000 is presented.

Table 7.10: Comparison of 3D convection-diffusion solvers on Test Problem 4a

Re	Four-Color Gauss-Seidel		Iterative GMRES-FFT	
	Iterations	CPU time (s)	Iterations	CPU time (s)
for $h = 1/32$				
1	8	1.02	7	0.20
10	10	1.14	17	0.36
100	18	1.64	39	0.81
1000	61	4.19	47	1.09
for $h = 1/64$				
1	8	6.86	7	0.73
10	10	7.62	16	1.47
100	19	10.47	46	5.15
1000	60	22.97	65	6.23

Table 7.10 reveals that the iterative GMRES-FFT method requires more iterations to con-

verge. However, the computational time taken for the iterative GMRES-FFT method is consistently faster than the four-color Gauss-Seidel method used in [23] for Test Problem 4a. It is also worth noting that if the coefficient b_l, c_l, and d_l were not modified as in Section 4.2.4, the iterative GMRES-FFT solver would require a total of 375 iterations and 22.60 seconds to compute the numerical solution of Test Problem 4a with $Re = 1000$ and step size $h = 1/64$.

Test Problem 4b

To further illustrate the quality of the developed iterative methods, we compared the proposed solver with another numerical solver but for the convection-diffusion equation with a large Reynolds number instead. The numerical experiment in [33] is conducted.

In [33], the 3D convection-diffusion equation with constant convection coefficients is considered. The test problem has a very large Reynolds number, ($Re = 10^7$) and $\alpha = \beta = \gamma = Re$. In this case, traditional iterative and multigrid methods would either fail to converge or has a slow convergence rate. The analytic solution is given by

$$u(x, y, z) = cos(4x + 6y + 8z)$$

defined on $\Omega = [0, 1] \times [0, 1] \times [0, 1]$.

The authors in [33] considered the solution of this test problem by the multigrid method with their developed fourth-order scheme (FOC) using either plane relaxation smoother or point relaxation smoother. The stopping criteria they used for the operator-based interpolation and the V-Cycle on the $2h$ and h grid steps were 10^{-10}. The result of their test was obtained from running on one processor of an IBM HS21 blade cluster at the University of Kentucky.

The processor has 2 GB local memory and runs at 2.0 GHz. We shall compute the solution of the same problem by applying the iterative method with the preconditioner system that assumes $\alpha = \beta = 0$ as in Test Problem 4a. The iteration method is set to stop once the relative residual falls below 10^{-10}. The numerical results were obtained from runs on the standard Alienware desktop. Also, we demonstrate that even the sequential variant of the developed iterative method is significantly faster than the multigrid method used in [33]. But this could be expected since the multigrid method used in [33] was designed to solve general 3D convection-diffusion equations with variable convection coefficients instead of problems with constant convection coefficients like this particular test problem.

The numerical results are listed in Table 7.11. The first column displays the step size, $h = 1/(N+1)$, used for the numerical experiments. Column two shows the type of numerical solvers used. Here FOC with the point or plane relaxation were the methods used in [33] and the iterative GMRES-FFT solver is the proposed iterative method in this dissertation. The max-err and CPU time (in seconds) for the algorithm are reported in the last two columns respectively.

Table 7.11 has demonstrated the efficiency of the proposed iterative method. The proposed iterative solver computes a similar approximation solution in a shorter amount of time for a 3D convection-diffusion equation with constant convection coefficients and a large Reynolds number.

Test Problem 4c

Test problems 4a and 4b have demonstrated the excellent performance of the iterative solver for computing numerical solutions of 3D convection-diffusion equations with constant con-

Table 7.11: Comparison of 3D convection-diffusion solvers on Test Problem 4b

Step size	Method	# iter	max-err	Time (s)
1/8	FOC with point relaxation	52	6.10e-02	0.009
	FOC with plane relaxation	8	6.10e-02	0.017
	Iterative GMRES-FFT solver	25	6.89e-02	0.109
1/16	FOC with point relaxation	164	1.41e-02	0.262
	FOC with plane relaxation	17	1.41e-02	0.320
	Iterative GMRES-FFT solver	38	1.99e-02	0.290
1/32	FOC with point relaxation	500	3.36e-03	7.621
	FOC with plane relaxation	26	3.36e-03	4.852
	Iterative GMRES-FFT solver	52	5.45e-03	1.012
1/64	FOC with point relaxation	Does not converge	-	-
	FOC with plane relaxation	46	8.21e-04	70.073
	Iterative GMRES-FFT solver	69	1.41e-03	6.395

vection coefficients. In this sub-subsection, we present the result of the iterative solver on a 3D convection-diffusion equation with the variable coefficient on z-direction as follows

$$\alpha(x,y,z) = 0$$

$$\beta(x,y,z) = 0$$

$$\gamma(x,y,z) = Re\sin(x)\sin(y)\sin(z)$$

$$u(x,y,z) = xyz(1-x)(1-y)(1-z)$$

defined on $\Omega = [0,1]\times[0,1]\times[0,1]$ and $Re = 10$. The right-hand side is given by

$$\begin{aligned} f(x,y,z) &= \Delta u(x,y,z) + \gamma(x,y,z)\frac{\partial}{\partial z}u(x,y,z) \\ &= -2(yz(y-1)(z-1) + xz(x-1)(z-1) + xy(x-1)(y-1)) \\ &\quad -\gamma(x,y,z)xy(x-1)(y-1)(2z-1) \end{aligned}$$

Preconditioning system

As discussed in subsection 5.6.2, the preconditioning system considers approximating the original test problem with the following 3D convection-diffusion equation

$$\Delta u(x,y,z) - \gamma_0(z)\frac{\partial}{\partial z}u(x,y,z) = f(x,y,z)$$

where $\gamma_0(z) = Re\sin(z)$ for this test problem. Then this preconditioner system can be solved by the direct FFT solver and the iterative process is set to stop once the relative residual is below 10^{-12}.

Convergence of numerical scheme

Table 7.12 presents the results for the test of fourth-order convergence of the approximate solution to the analytic solution for Test Problem 4c. The first column shows the grid size of the grids used for the numerical experiments. Column two of the table shows the number of iterations required until the convergence of the iterative solver. The third and fourth columns report the max-err and the sequential CPU time (in seconds) for the algorithm to complete. The rate of convergence is displayed in the last column of the table.

Table 7.12: Test for convergence of 3D convection-diffusion equation with variable coefficient

N	Iteration	max-err	Time(s)	Rate of Convergence
32	15	5.008e-10	1.687	-
64	15	3.336e-11	11.898	3.91
128	15	2.151e-12	90.094	3.96

The numerical experiments stop at grid 128^3 since the computed solution obtained was already very close to the analytical solution as evidenced by the max-err at 128^3 grid which

is at 10^{-12}. All in all, the results show promising performance for the iterative solver for the convection-diffusion equation with variable coefficient in the z-direction.

7.2.3 Subsurface inclusion model problem

In this subsection, we consider the mathematical model for subsurface imaging such as landmine detection that was introduced in subsection 2.1.1. In this model, mine-like targets are considered to be hidden within the ground represented by $\{0.5 \leq z \leq 1.0\}$ and $\{0 \leq z < 0.5\}$ representing the air. The frequency $\omega = 1$ GHz was chosen here so that the value of the coefficient k^2 is given by Table 2.1. In particular, the coefficient function k^2 is defined as

$$k^2(x, y, z) = \begin{cases} 439.2, & \text{in air,} \\ 1273 + 31i, & \text{in soil and outside inclusions,} \\ 1256 + 2.26i, & \text{inside the inclusions.} \end{cases} \tag{7.1}$$

Here inclusions refer to parts of the computational domain that would correspond to the position of the mine-like target.

Preconditioning system

In this subsection, we will consider the preconditioning strategy discussed in subsection 5.6.1. where the preconditioner matrix is created by approximating $k^2(x, y, z)$ with $k_0^2(z)$ given by

$$k_0^2(z) = \begin{cases} 439.2, & \text{for } 0 \leq z < 0.5, \\ 1273 + 31i, & \text{for } z \geq 0.5, \end{cases} \tag{7.2}$$

The numerical algorithm for these numerical experiments is run on the standard Alienware desktop unless stated otherwise. The iterative method is set to stop once the relative residual is below 10^{-6}.

The real part of the solution vector obtained from the iterative method is then stored and drawn using MATLAB libraries. The plot from the approximated solution vector is compared

to an actual illustration of the computation domain with inclusion. The residual after each iteration is also stored to analyze the rate of convergence.

Single mine-like target inclusion in ground

In the first experiment for the model problem, we consider the case where circular mine-like targets were embedded within the ground. The mine-like target is modeled as a sphere with center $(0.7, 0.7, 0.7)$ with a radius of $r = 0.1$. Specifically the coefficient function k^2 can be written as

$$k^2(x,y,z) = \begin{cases} 439.2, & \text{for } 0 \le z < 0.5, \\ 1273 + 31i, & \text{for } (x-0.7)^2 + (y-0.7)^2 + (z-0.7)^2 > 0.01 \text{ and } z > 0.5, \\ 1256 + 2.26i, & \text{for } (x-0.7)^2 + (y-0.7)^2 + (z-0.7)^2 \le 0.01, \end{cases} \tag{7.3}$$

The first two rows in Equation (7.3) correspond to k_0^2 mentioned earlier, and the third one is due to a mine-like target. Figure 7.4 illustrates the coefficient function k^2 for this setup while Figure 7.5 shows a color plot of the real part of the approximated numerical solution U over the computational domain. Both figures display the slices of the computational domain at $x = 0.2$ and $y = 0.7$ and the approximate solution u_h is obtained by using the sixth-order scheme on a computational grid of size 117^3. It took a total of 18 iterations for the iterative method to converge for this model problem with the sixth-order scheme.

Figure 7.5 proves that the proposed numerical method can recognize the existence of the mine-like target and is capable of estimating the general location of the mine even on a small grid of 117^3.

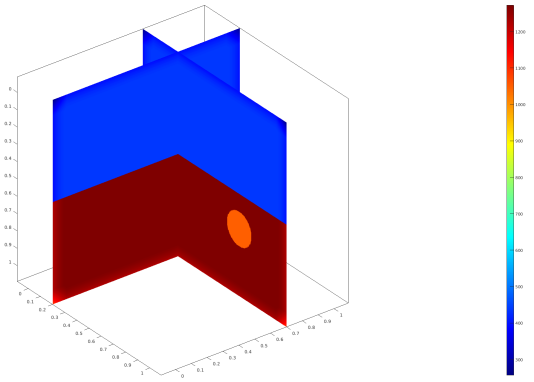

Figure 7.4: Color plot of the coefficient $k^2(x, y, z)$ with one circular inclusions

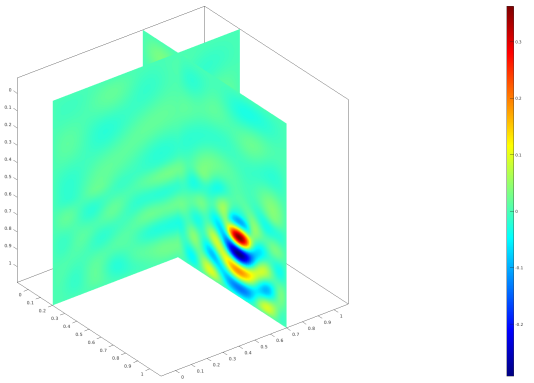

Figure 7.5: A color plot of the real part of the computed solution for subsurface with one inclusion

Multiple mine-like targets inclusion in ground

The next numerical test involved the case where multiple, in this case, two, mine-like targets of different sizes and shapes embedded inside the ground. Similar to the previous test, the values of the coefficients k^2 are given by Table 2.1. However, in addition to the circular mine target in the previous test another mine-like target will be embedded within the ground. The new mine-like target will be modeled as a cube centered at $(0.2, 0.1, 0.85)$ with its length set

as 0.1.

The coefficient k^2 is illustrated in Figure 7.6 while Figure 7.7 shows a color plot of the real part of the approximate numerical solution u_h over the computational domain using a computational grid of size 117^3. Both figures display the slices of the computational domain at $x = 0.2$ and $y = 0.7$ and the approximate solution U is obtained by using the sixth-order scheme. It took a total of 19 iterations for the iterative method to converge for this model problem with the sixth-order scheme.

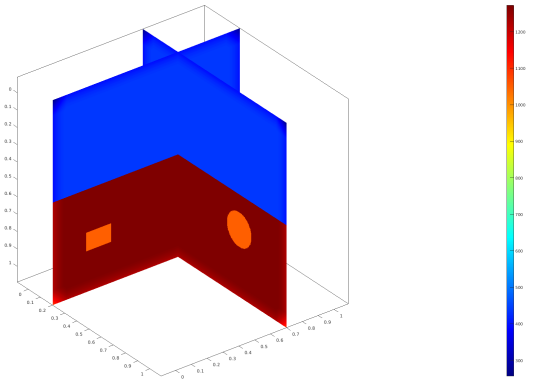

Figure 7.6: Color plot of the coefficient $k^2(x, y, z)$ with two inclusions

This numerical experiment proves that the proposed numerical method can clearly identify and distinguish multiple mine-like targets and even provide an estimation for the general location of the mine-like targets.

Lastly, Figure 7.8 displays and compares the scaled convergence history of the relative residual on each iteration in the preconditioned GMRES method based on the second, fourth, and sixth-order compact approximation scheme on the computation grid of 117^3. The logarithmic scale is used to represent the convergence history.

The sixth-order scheme appears to have the best convergence rate for this modeling problem

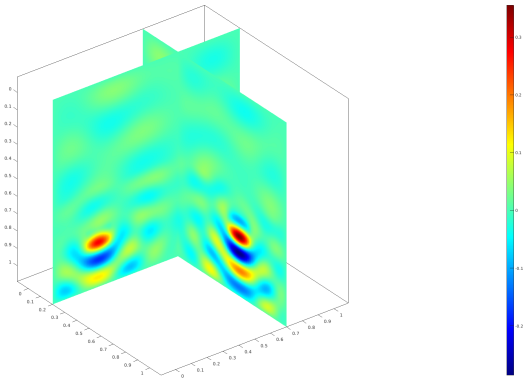

Figure 7.7: A color plot of the real part of the computed solution for the subsurface with two inclusions

as shown in Figure 7.8. All in all, these numerical experiments were able to successfully prove that the proposed iterative solvers are capable of producing images that display the location of the targets embedded inside the ground. This is important to the main targeted application for subsurface imaging as now the iterative solver can be used to generate multiple images for various types of inclusion for the inverse problem of the subsurface scattering problem.

7.2.4 Low-order preconditioners

In this subsection, we shall test the performance of the iterative method based on a high-order scheme with a low-order preconditioner. Specifically, the matrix used in the matrix-vector multiplication step in GMRES will be based on high order scheme, that is fourth or sixth order while the preconditioner system would be solving the preconditioner system with a lower order scheme. Thus a fourth-order scheme iterative method would use a second-order scheme preconditioner and a sixth-order iterative method could use either a fourth or second-

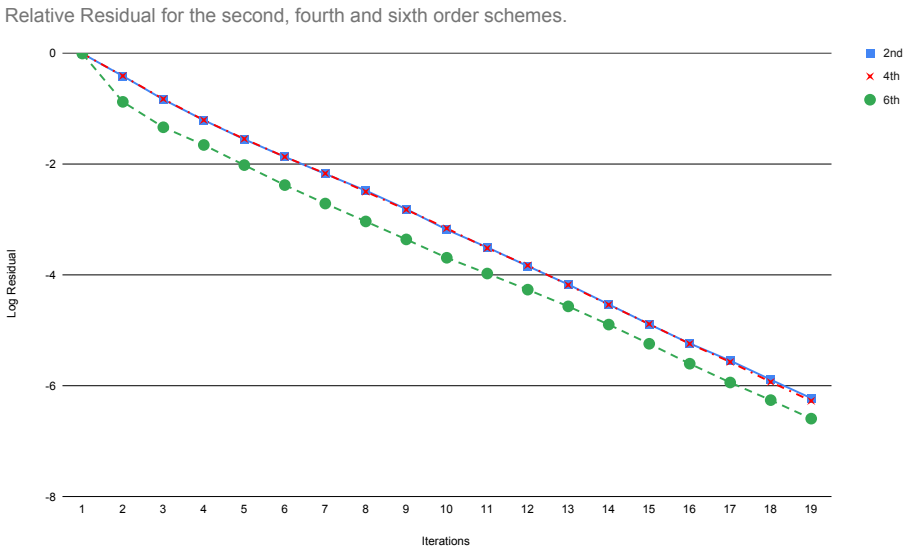

Figure 7.8: Convergence history of the preconditioned GMRES method

order preconditioner.

Numerical experiments with Test Problem 3 and the subsurface inclusion model problem with one inclusion are repeated. The numerical experiment with test Problem 4a will also be repeated to demonstrate how a higher-order accuracy solution can be obtained with the iterative method. The same preconditioning system in previous test problems would be applied but with lower order preconditioning scheme instead.

All results for the numerical experiments in this subsection are obtained by a run on a standard Macbook Pro laptop with a 2.3 GHz Quad-Core Intel Core i5. Moreover, the numerical algorithm is also implemented via MATLAB.

Test Problem 3 with low-order preconditioning

Table 7.13 shows the result of fourth-order iterative solver with second and fourth-order preconditioner, while the result of sixth-order iterative solver with second, fourth, and sixth-

order preconditioner is displayed in Table 7.14. The first column of both tables displays the grid size used. The second column shows order of the preconditioning system. The number of iterations until the iterative method converges is reported in the third column and the final two columns report the max-err and total computation time (in seconds) respectively. Similar to before, the iterative method is set to stop once the relative residual falls below 10^{-12}.

Table 7.13: Comparison of 4th order iterative solver with various order preconditioner on Test Problem 3

N	Preconditioner Scheme	Iteration	max-err	Time(s)
32	2nd	16	5.225e-05	0.226
	4th	10	5.225e-05	0.202
64	2nd	16	3.575e-06	1.247
	4th	10	3.575e-06	0.864
128	2nd	16	2.314e-07	11.167
	4th	10	2.314e-07	6.834
256	2nd	15	1.470e-08	153.718
	4th	10	1.470e-08	96.870

The results show that except for the sixth-order scheme with the second-order preconditioning system showing signs of deteriorating performance at the end, every other combination considered can compute similar results to the iterative solver using a matching order preconditioner. Additionally, this numerical test shows a consistent result to support the use of a sixth-order iterative scheme with a fourth-order preconditioner which can produce comparable accuracy with the sixth-order preconditioner in less computation time.

Table 7.14: Comparison of 6th order iterative solver with various order preconditioner on Test Problem 3

N	Preconditioner Scheme	Iteration	max-err	Time(s)
32	2nd	15	4.693e-07	0.242
	4th	10	4.693e-07	0.147
	6th	10	4.693e-07	0.171
64	2nd	15	8.244e-09	1.488
	4th	10	8.243e-09	0.990
	6th	10	8.243e-09	1.016
128	2nd	15	1.496e-10	13.339
	4th	10	1.361e-10	9.193
	6th	10	1.361e-10	9.845
256	2nd	14	7.844e-10	185.004
	4th	10	3.890e-12	133.160
	6th	10	2.504e-12	136.730

Subsurface inclusion model problem with low-order preconditioning

Table 7.15 displays the performance of the iterative solver with matching and lower order scheme preconditioner on a grid of 117^3. The first column shows the order of the iterative scheme. The ordering scheme of the preconditioner is shown in the second column. The number of iterations required for the iterative method to converge, the relative residual, and the computation time is taken (in seconds) are displayed on the last three columns respectively.

The results from Table 7.15 would suggest that the order of scheme used by the preconditioning system does not have a huge impact on the performance of the proposed iterative method for the application of subsurface imaging.

Table 7.15: Comparison of iterative solver with lower order preconditioner on subsurface inclusion model problem

Iterative Scheme	Preconditioner Scheme	Iteration	L_2-res	Time(s)
4th	2nd	19	4.89e-07	24.239
	4th	18	5.79e-07	23.828
6th	2nd	19	4.83e-07	31.204
	4th	18	5.74e-07	30.250
	6th	18	5.77e-07	30.035

Test Problem 4a for higher accuracy with low-order preconditioning

In this subsubsection, we explore the possibility of obtaining higher-order accuracy for the numerical solution of Test Problem 4a. Wang and Zhang [33] use an operator-based interpolation scheme combined with an extrapolation technique to approximate a sixth-order accurate numerical solution. To the best of the author's knowledge, there is no other explicit sixth-order compact scheme for the convection-diffusion equation.

Given the analytic solution for Test Problem 4a, a sixth-order compact scheme can be obtained by modeling Test Problem 4a after the Helmholtz equation. Specifically defined the coefficient function $k^2(x, y, z)$ as

$$\begin{aligned} k^2(x,y,z) &= \alpha\frac{u_x}{u} + \beta\frac{u_y}{u} + \gamma\frac{u_z}{u} \\ &= \alpha\pi\cot(\pi x) + \beta\pi\cot(\pi y) + \gamma\pi\cot(\pi z) \end{aligned}$$

where $u_x = \frac{\partial}{\partial x}u$ and similarly for u_y and y_z on $\Omega = [\epsilon, 1-\epsilon] \times [\epsilon, 1-\epsilon] \times [\epsilon, 1-\epsilon]$ where $\epsilon = 10^{-300}$. The computational domain is changed because k^2 is undefined when the variables are 0 or 1. The coefficient matrix for Test Problem 4a can be derived from the sixth-order approximation compact scheme in Section 3.4. The preconditioner matrix is still derived from

the fourth-order scheme for the 3D convection-diffusion equation with constant coefficients in subsection 4.2.4. It is worth mentioning that the magnitude of k^2 in Test Problem 4a is incredibly large near the boundary of the computational domain. Hence, the resulting matrix would be ill-conditioned and even fail to converge for medium to large Reynolds number ($Re \geq 100$).

A series of numerical tests were conducted on Test Problem 4a with various grid sizes, that is $8^3, 16^3$, 32^3, and 64^3 and the Reynolds number, $Re = 1$ and 10. The iterative process is set to stop once the relative residual is below 10^{-10}. The results of this series of numerical tests are presented in Table 7.16 and Table 7.17. The first column of the table displays the grid size $N = N_x = N_y = N_z$ used for the numerical experiments. Column two of the table would display the order of the scheme used and column three presents the number of iterations required before the desired error is obtained. The fourth column reports the max-err. Also shown is the sequential CPU time (in seconds) for the iterative algorithm to complete in the sixth column. Finally, the last column of the table shows the rate of convergence, the rate of convergence for the explicit scheme is not shown as the method would always converge to the desired accuracy or stop as GMRES hit the maximum number of iterations.

Table 7.16 and Table 7.17 show the promising result as the iterative method can approximate an almost sixth-order accurate solution for Test Problem 4a with $Re = 1$ and 10 in the same computational time as the fourth-order scheme.

Incidentally, from the trigonometry angle addition and subtraction theorems, we have

$$\sin(\pi x + \pi h) = \sin(\pi x)\cos(\pi h) + \cos(\pi x)\sin(\pi h)$$

$$\sin(\pi x - \pi h) = \sin(\pi x)\cos(\pi h) - \cos(\pi x)\sin(\pi h)$$

Table 7.16: Rate of convergence for Test Problem 4a with $Re = 1$ for various order schemes

N	Scheme	Iteration	max-err	Time(s)	Rate of Convergence
8	4th	7	1.51e-04	0.12	-
	6th	7	8.75e-06	0.10	-
16	4th	7	1.19e-05	0.18	3.67
	6th	7	1.95e-07	0.23	5.49
32	4th	7	8.34e-07	0.28	3.83
	6th	7	3.65e-09	0.23	5.74
64	4th	7	5.55e-08	0.73	3.91
	6th	7	6.45e-11	1.02	5.82

Table 7.17: Rate of convergence for Test Problem 4a with $Re = 10$ for various order schemes

N	Scheme	Iteration	max-err	Time(s)	Rate of Convergence
8	4th	15	5.69e-04	0.11	-
	6th	20	3.31e-05	0.11	-
16	4th	16	4.46e-05	0.18	3.67
	6th	20	1.26e-06	0.14	4.71
32	4th	17	3.16e-06	0.32	3.82
	6th	20	3.42e-08	0.38	5.20
64	4th	16	2.10e-07	1.73	3.91
	6th	20	7.77e-10	2.69	5.46

where $h = h_x = h_y = h_z$ is the grid step size. Thus

$$\cos(\pi x) = \frac{\sin(\pi x + \pi h) - \sin(\pi x - \pi h)}{2\sin(\pi h)}$$

Similar results can be obtained in the y- and z-directions. Let $C = \pi/(2\sin(\pi h))$, Test Problem 4a can be discretized explicitly as 7-point stencil as follows

$$-3\pi^2 u_{i,j,k} + \alpha C(u_{i+1,j,k} - u_{i-1,j,k}) + \beta C(u_{i,j+1,k} - u_{i,j-1,k}) + \gamma C(u_{i,j,k+1} - u_{i,j,k-1}) = f_{i,j,k}.$$

Using this explicit scheme, the iterative method can compute the numerical solution for Test

Problem 4a for any Reynolds number at the cost of a higher number of iterations. Table 7.18 presents the numerical results of running Test Problem 4a with the explicit scheme for the Reynolds number, $Re = 10^3$, 10^5, and 10^7 on a grid of 7^3. The first column displays the value of the Re used while the second column reports the max-err. The number of iterations required before the relative residual is below 10^{-15} is shown in the third column. Lastly, the computational time taken for the iterative algorithm to complete is in the fourth column.

Table 7.18: Numerical results for Test Problem 4a with explicit scheme on a grid of 7^3

Re	max-err	Iterations	Time(s)
1000	1.4323e-13	306	0.54
100000	6.1441e-11	319	0.62
10000000	1.3583e-09	318	0.54

Overall the numerical tests in this subsection have shown that one may attempt to approximate a higher-order scheme with a preconditioner of lower order especially when the order was close enough. Numerical experiments on Test Problem 4a also demonstrated the strength of the iterative method that enables a fourth-order compact scheme to compute the numerical solution with higher accuracy. This could open the door for more applications for the direct FFT solver as a potential option as a preconditioner solver for an iterative method based on a higher-order scheme in the future.

7.2.5 Scalability of iterative GMRES-FFT solver

In this subsection, the numerical results of each parallel implementation will be presented. The performance will be evaluated and analyzed in depth. All numerical experiment results

obtained in this section were conducted on the standard Alienware desktop unless stated otherwise.

The numerical experiments on the subsurface inclusion test problem presented in subsection 7.2.3 with a single mine-like target embedded in the ground are repeated. The numerical experiment is performed on a grid of size 256^3 with the sixth-order scheme. For this particular series of tests, the iterative method is set to stop once the relative residual is less than 10^{-6}. The iterative GMRES-FFT solver was able to successfully converge after a total of 27 iterations for all types of implementations (sequential or parallel) regardless of the number of processes used.

The overall run time, measured in seconds, of the whole iterative method as well as individual parts or steps of the method, were also recorded to analyze its performance. The run time was rounded off to the nearest two decimal places and presented in the following table. The average time for each iteration is calculated by taking the ratio between the overall run time and the number of iterations it took, in this test that would be 27.

The result of the OpenMP implementation of the numerical algorithm with various numbers of OpenMP threads is presented in Table 7.19 for comparison. The first row in Table 7.19 shows the average run time for the preconditioner direct solver discussed previously for all 27 iterations discussed previously. It also shows the timing for each step inside the preconditioner algorithm. The second row displays the time it took for the algorithm to perform the matrix-vector multiplication while the third row presents the estimated time for other steps that were left entirely up to PETSc such as the Arnoldi method, the timing of which is calculated by subtracting the time for preconditioner direct solver and matrix-vector multiplication from the average time for each iteration. The second column of Table

7.19 represents running the algorithm sequentially and the remaining column in Table 7.19 represents the results obtained from OpenMP implementation using two and four OpenMP threads respectively.

Table 7.19: Performance of OpenMP implementation on a grid of 256^3

Step	Sequential	2 threads	4 threads
Preconditioner Direct Solver			
Forward FFT	2.95	1.56	0.81
LU Tridiagonal Solver	0.38	0.22	0.11
Inverse FFT	3.03	1.66	0.83
Total	6.38	3.46	1.77
Matrix-Vector Multiplication	2.11	1.29	0.75
Other	2.30	2.23	2.33
Average time for each iteration	10.79	6.98	4.85
Overall	291.41	188.41	130.82

Similar to its performance in direct FFT solver, the OpenMP implementation was able to successfully reduce the run time for both the preconditioner and matrix-vector multiplication step almost linearly by a factor of two as the number of threads doubled. However, the time taken for the iterative methods to complete other steps within the GMRES method such as the Arnoldi method appears to be unchanged despite the increase in the number of OpenMP threads used. Unfortunately, this is to be expected as PETSc by default is not thread-safe [26] and hence it does not support OpenMP implementation unless the code is manually modified as we did for the preconditioner solver and matrix-vector multiplication. Table 7.19 also supports this viewpoint as it shows that the only step OpenMP implementation managed to parallelize were steps that were manually modified.

Hence GMRES implementation that relies on PETSc libraries such as the proposed iterative

method will not be able to perform optimally with OpenMP implementation. Moreover looking at the performance of using four OpenMP threads, one can observe that this step has overtaken the previously most computational heavy preconditioner direct solver step. For the OpenMP implementation, these would become a big bottleneck and heavily restrict its performance.

On the other hand, the result of MPI implementation of the iterative algorithm with various numbers of MPI processors is presented in Table 7.20 for comparison. Now the second column of Table 7.20 represents running the algorithm sequentially and the remaining column in Table 7.20 represents the results obtained from MPI implementation using two and four MPI processors respectively. A new row is included within the preconditioner step to report the communication time between processors.

Table 7.20: Performance of the MPI implementation on a grid size of 256^3

Step	Sequential	2 processors	4 processors
Preconditioner Direct Solver			
Forward FFT	2.95	1.48	0.78
LU Tridiagonal Solver	0.38	0.23	0.12
Inverse FFT	3.03	1.48	0.78
Communication	-	0.22	0.13
Total	6.38	3.42	1.83
Matrix-Vector Multiplication	2.11	1.19	0.66
Other	2.30	1.31	0.75
Average time for each iteration	10.79	5.92	3.24
Overall	291.41	159.88	87.58

Contrasting to the OpenMP performance, one can observe that the MPI implementation has a significantly better performance. It is reasonable to assume this could be because

PETSc does have better support for MPI implementation. This hypothesis is supported by the observation that the run time in the row label "Other" from Table 7.20 has shown a visible reduction as the number of MPI processors doubled. Thus MPI implementation can archive a significantly better performance overall.

Table 7.21 shows the run time of different parallel implementations for each step in the iterative method. The second column of the table displays the performance of running the algorithm sequentially. The third and fourth columns of the table show the performance of running the algorithm using four OpenMP threads and four MPI processors respectively. Hybrid I refers to the hybrid implementation shown in Algorithm 11 and Hybrid II refers to the hybrid implementation shown in Algorithm 12. The result for each respective hybrid implementation is presented in the fifth and sixth columns of Table 7.21. Both the hybrid implementation uses exactly two OpenMP threads and two MPI processors.

Table 7.21: Performance of various implementations on a grid size of 256^3

Step	Sequential	OpenMP	MPI	Hybrid I	Hybrid II
Preconditioner Direct Solver					
Forward FFT	2.95	0.81	0.78	0.78	0.88
LU Tridiagonal Solver	0.38	0.11	0.12	0.13	0.13
Inverse FFT	3.03	0.83	0.78	0.79	0.89
Communication	0.00	0.00	0.13	0.17	0.18
Total	6.38	1.77	1.83	1.90	2.09
Matrix-Vector Multiplication	2.11	0.75	0.66	0.63	0.63
Other	2.30	2.33	0.75	1.33	1.33
Average time for each iteration	10.79	4.85	3.24	3.86	4.06
Overall	291.41	130.82	87.58	104.11	109.53

The numerical test shown in Table 7.21 suggested that both hybrid OpenMP-MPI implemen-

tations have almost similar performance when running on a medium size problem. Hybrid II implementation also took the longest time to compute both forward and inverse FFT does show the increase in overhead for using multithread FFTW. The overall performance for both hybrid implementations is better than the OpenMP implementation but falls behind the performance of the MPI implementation. This is to be expected based on results from previous numerical tests and could be attributed to PETSc support for strictly MPI implementation.

Nonetheless, numerical experiments on direct FFT solver in subsection 7.1.3 have proved that the true strength of hybrid OpenMP-MPI implementations is shown when running on larger grid size problems. As such numerical experiments were performed on a grid of 1024^3. The results of the test were obtained from a run on the Cori supercomputer as there is not enough memory to run the algorithm on a local machine. It would require a minimum of 8 Haswell nodes to perform a run on a grid of 1024^3. The iterative process is set to stop when the relative residual is below 10^{-6}. It took a total of 22 iterations for the fourth-order scheme to stop while the sixth-order scheme requires 26 iterations.

Table 7.22 and 7.23 shows the computations times measured in seconds for the Hybrid II implementation of the iterative solver on a grid of 1024^3 on Cori with fourth and sixth order scheme respectively. Similar to the previous table, the nodes in the table indicate the number of Haswell nodes the solver is using which also serves as the number of MPI processes used. It is worth reiterating that each Haswell node has 32 physical cores available which serve as the number of OpenMP threads. Here, the number of OpenMP threads changes horizontally, and the MPI processes change vertically.

One can observe from Table 7.22 and 7.23 that the computation times displayed almost linear

Table 7.22: Hybrid II implementation of iterative solver with 4th order scheme

Nodes	1 Thread	4 Threads	16 Threads	32 Threads
8	1609.37	633.04	364.98	315.03
16	805.82	319.73	184.62	168.63
32	391.51	159.76	86.33	73.37

Table 7.23: Hybrid II implementation of iterative solver with 6th order scheme

Nodes	1 Thread	4 Threads	16 Threads	32 Threads
8	2022.59	777.82	461.24	401.76
16	991.24	401.28	284.56	259.48
32	489.56	196.32	112.66	94.43

scalability as the number of MPI processes increased. However, the scalability observed when OpenMP threads increased did not perform optimally. Nonetheless, the presented results are consistent with other numerical experiments on the iterative solver.

Lastly, the numerical experiments for the Hybrid II implementation with a sixth-order scheme were repeated but with a grid of 2089^3 instead. The test was performed using LBNL's supercomputer Cori using a total of 64 Haswell nodes. The run would take 31 iterations to reach the desired tolerance of 10^{-6} with a total computational time of 1283 seconds which is equivalent to 21 minutes and 23 seconds. Unfortunately, due to the limited resources allocated to the author and co-researchers, the run was only able to complete once before running out of node hours and memory. This is the limit of the numerical experiment performed on the proposed iterative solver.

In conclusion, this dissertation has presented an iterative scalable GMRES-FFT-type algorithm for a class of compact numerical approximations on a rectangular grid. The developed

algorithms represent highly accurate and scalable methods for the solution of the considered problems. The advantages of the iterative method over the direct method are shown by the wide variety of test problems it manages to compute whereby the direct method would fail. Specifically, the iterative solver is now capable of solving general 3D Helmholtz equation and 3D convection-diffusion equation with a constant convection coefficient. The parallel implementation also enables the developed method to solve these problems with a considerably large size in a short amount of time.

8 Conclusion and Future Work

The recent development of multi-core technologies on modern desktop computers makes parallelization of the proposed numerical approaches a priority in algorithmic research. This dissertation presented an efficient parallel generalized GMRES-FFT-type iterative algorithm with near-optimal complexity for PDE such as the Helmholtz equation on a rectangular grid. The iterative solvers utilize high-order approximations to provide high-resolution solutions for the PDE. Moreover, the robustness of the solver is demonstrated by finding the numerical solution for 3D convection-diffusion equation. Numerical results from various test problems based on the Helmholtz and convection-diffusion equation were able to verify the effectiveness of the proposed iterative approach. Above all, the iterative solver has proven its capability to find the solution for the forward problem of propagation of high-frequency GPR signals using PML boundary conditions over regions with small inclusions. Test problems with realistic parameter ranges were used to verify the performance of the proposed iterative method.

Chapter 6 has shown how one may convert the sequential algorithm to a parallel algorithm with relative ease. This is also supported by numerical experiments in Chapter 7 that demonstrated the efficiency of the OpenMP, MPI, and hybrid OpenMP-MPI implementations of the proposed parallel algorithms. This includes the vast improvement in computation time and the ability to compute solutions for problems with large grid sizes.

8.1 Future Work

The performance of the OpenMP implementation on the iterative solver is less than ideal. An improvement to the hybrid approach can be considered by replacing the OpenMP with other parallel toolboxes. In particular, one may attempt GPU-enabled GMRES algorithms that were designed to use NVIDIA's Compute Unified Device Architecture (CUDA). All of the Krylov methods in PETSc except KSPIBCGS run on the GPU systems from NVIDIA using CUDA, and AMD and Intel using OpenCL/ViennaCL and HIP [26]. While PETSc still supports strictly MPI implementation, a hybrid approach with the aforementioned technique may be considered.

The development of a fast and accurate numerical method for the solution of the forward problem is just a necessary preliminary step to the solution of the inverse problem. The proposed iterative solver could be used to compute solutions to the forward problem with multiple variations of the inclusion such as different locations, sizes, and shapes of the inclusion. These solutions generate images at the surface that can be used to train a convolutional neural network (CNN) to accurately identify the location, shape, and size of the inclusion.

Bibliography

[1] A.George and J.W.Liu, *Computer Solution of Large Sparse Positive Definite* Prentice Hall, New Jersey, 1981.

[2] *Cayley-Hamilton theorem* Encyclopedia of Mathematics. URL: `http://encyclopediaofmath.org/index.php?title=Cayley%E2%80%93Hamilton_theorem&oldid=22272`

[3] D.Colton, and R.Kress, *Inverse Acoustic and Electromagnetic Scattering Theory* 2nd Edition, Springer, 1998.

[4] D.Gordon, and R.Gordon, *Solution methods for linear systems with large off-diagonal elements and discontinuous coefficients* Comp. Model. Eng. Sci. 53 (2009) 23–45.

[5] E.Turkel, D.Gordon, R.Gordon, and S.Tsynkov, *Compact 2D and 3D Sixth Order Schemes for the Helmholtz Equation with Variable Wave Number*. Journal of Computational Physics 232 (2013) 272-287.

[6] G.J.Hicks, *Arbitrary source and receiver positioning in finite difference schemes using Kaiser windowed sinc functions*. Geophysics 67 (2002) 156–166.

[7] G.Reid, *Landmine detection using semi-supervised learning* Electronic Theses and Dissertations, University of Louisville, 2018, Paper 3132. `https://doi.org/10.18297/etd/3132`

[8] H. Elman, and D.O'Leary, *Efficient iterative solution of the three- dimensional Helmholtz equation* J. Comput. Phys. 142 (1998) 163–181.

[9] H.Elman, and D.O'Leary, *Eigenanalysis of some preconditioned Helmholtz problems* Numer. Math. 83 (1999) 231–257.

[10] H.Elman, S.David, and W.Andy, *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics, 2nd edn* Oxford University Press, 2005

[11] H.Jin, D.Jespersen, P.Mehrotra, R.Biswas, L.Huang, and B.Chapman, *High performance computing using MPI and OpenMP on multi-core parallel systems* Parallel Computing. 37. 562-575. https://doi.org/10.1016/j.parco.2011.02.002

[12] I.Duff, A.Erisman, and J.Reid, *Direct Methods for Sparse Matrices, 2nd edn* Oxford University Press, 2002

[13] I.Giannakis, *Realistic numerical modelling of Ground Penetrating Radar for landmine detection* 2016

[14] I.G.Graham, and S.Sauter, *Stability and finite element error analysis for the Helmholtz equation with variable coefficients* Math. Comp. 89 (2020) 105–138.

[15] I.Singer, and E.Turkel, *A perfectly matched layer for the Helmholtz equation in a semi-infinite strip.* J. Comput. Phys. 201 (2004) 439–465.

[16] J.Kalita, A.Dass, and D.Dalal, *A transformation-free hoc scheme for steady convection-diffusion on non-uniform grids.* Int. J. Numer. Meth. Fluids 44 (2004) 33–53.

[17] J.Toivanen, and M.Wolfmayr, *A fast fourier transform based direct solver for the Helmholtz problem* Numer. Linear Algebra Appl. 27 (e2283) (2020) 272–287.

[18] J. Zhang, *Multigrid acceleration techniques and applications to the numerical solution of Partial Differential Equations* Ph.D. Thesis, The George Washington University, Washington, DC, 1997.

[19] J.Zhang, *An explicit fourth-order compact finite difference scheme for three dimensional convection–diffusion equation* Commun. Numer. Meth. Eng. 14 (1998) 263–280.

[20] J. Zhang *Preconditioned iterative methods and finite difference schemes for convection–diffusion* Appl. Math. Comput., 109 (2000), pp. 11-30.

[21] L.Liu, R.Li, G.Yang, B.Wang, L.Li, and Y.Pu, *Improving parallel performance of a finite-difference AGCM on modern high-performance computers* J. Atmos. Ocean. Technol. 2014, 31, 2157–2168.

[22] M.Frigo, and S.Johnson, *FFTW Manual* Massachusetts Institute of Technology, January 2003).

[23] M.Gupta, and J. hang, *High Accuracy Multigrid Solution of the 3D Convection-Diffusion Equation* Applied Mathematics and Computation. 1998, 113. 249-274. https://doi.org/10.1016/S0096-3003(99)00085-5

[24] M.Valera, M.P.Thomas, M.Garcia, and J.E.Castillo, *Parallel Implementation of a PETSc-Based Framework for the General Curvilinear Coastal Ocean Model* J. Mar. Sci. Eng. 2019, 7, 185. https://doi.org/10.3390/jmse7060185

[25] R.Kress *Numerical Analysis* Springer, (1998).

[26] S.Balay, S.Abhyankar, M.Adams, J.Brown, P.Brune, K.Buschelman, L.Dalcin, A.Dener, V.Eijkhout, W.Gropp, D.Karpeyev, D.Kaushik, M.Knepley, D.May, L.Curfman McInnes, R.Mills, T.Munson, K.Rupp, P.Sanan, B.Smith, S.Zampini, H.Zhang, and H.Zhang *PETSc Users Manual- Revision 2.3.2* ARGONNE NATIONAL LABORATORY, September 1, 2006.

[27] S.Lele, *Compact finite difference schemes with spectral-like resolution.* Journal of Computational Physics 103 (1992) 16-42.

[28] S.V.Pakantar, *Numerical Heat Transfer and Fluid Flow* McGraw-Hill, New York (1980).

[29] T.M.Shih, *Numerical Heat Transfer* Hemisphere, Washington, DC (1984).

[30] U.Ananthakrishnaiah, R.Manohar, and J.W.Stephenson, *High-order methods for elliptic equation with variable coefficients* Numer. Meth. Partial Differen. Eqns. 3 (1987) 219–227.

[31] W.Wang, T.Fischer, B.Zehner, N.Böttcher, U.J.Görke, and O.Kolditz, *A parallel finite element method for two-phase flow processes in porous media: OpenGeoSys with PETSc* Environ. Earth Sci 73, 2269–2285 (2015). https://doi.org/10.1007/s12665-014-3576-z

[32] Y.Saad, and M.H.Schultz, *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems* Siam Journal on Scientific and Statistical Computing, 1986, 7, 856-869.

[33] Y.Wang, and J.Zhang, *Fast and Robust Sixth Order Multigrid Computation for 3D Convection Diffusion Equation* J Comput Appl Math. 2010 Oct 15;234(12):3496-3506. https://doi.org/10.1016/j.cam.2010.05.022

[34] Y.A.Gryazin, *High order approximation compact schemes for forward subsurface scattering problems* in: Proceedings of the SPIE 9077, Radar Sensor Technology XVIII Conference, 2014, pp. 1–9. https://doi.org/10.1117/12.2050189

[35] Y.Gryazin, *Preconditioned Krylov Subspace Methods for Sixth Order Compact Approximations of the Helmholtz Equation* ISRN Computational Mathematics, 2014, https://doi.org/10.1155/2014/745849

[36] Y.Gryazin, M.Klibanov, and T.Lucas, *Gmres computation of high frequency electrical field propagation in land mine detection* J. Comput. Phys. 158 (2000) 98–115.

[37] Y.Gryazin, R.Gonzales, and Y.T.Lee, *Parallel FFT algorithms for high-order approximations on three-dimensional compact stencils* Parallel Computing, Volume 103, 2021, 102757, ISSN 0167-8191, https://doi.org/10.1016/j.parco.2021.102757