Towards GPU-accelerated Direct and Iterative Solvers for Computational Wave Scattering in HELM

Auteur : Distrée, Florent
Promoteur(s) : Arnst, Maarten
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil physicien, à finalité approfondie
Année académique : 2024-2025
URI/URL : http://hdl.handle.net/2268.2/24701

Université de Liège - Faculté des Sciences Appliquées

Towards GPU-accelerated Direct and Iterative Solvers for Computational Wave Scattering in *HELM*

Author:
Distrée Florent

Supervisors:
Prof. Maarten Arnst
Romin Tomasetti

Thesis presented to obtain the degree of
Master of Science in Physics Engineering

August 21
— Academic Year 2024-2025 —

CONTENTS

REMERCIEMENTS

Un immense merci à mon promoteur, Maarten Arsnt, pour ses conseils avisés, sa patience et pour m'avoir maintenu sur le droit chemin tout au long de ce travail.

Je remercie également Romin Tomasseti pour son œil d'expert et sa perspicacité. Ses critiques, toujours justes, ont été précieuses pour donner à ce projet sa forme finale.

À mon cher compatriote Tom, avec qui le cliché des « meilleures années de sa vie » à l'université est devenu une réalité. Des sessions de travail intenses aux débriefings plus... informels, ce fut un plaisir de partager chaque moment de cette aventure avec toi.

À Kenza, mon ancre. Merci pour ton amour infini et ta compassion, surtout dans les moments où la motivation me fuyait. Ta patience a été mon plus grand soutien.

Et bien évidemment, à mes parents, sans qui rien de tout cela n'aurait été possible. Qui m'ont aidé à me relever et avancer à chaque obsctavle, qui m'ont conseiller et aviser dans les mometns clès. Merci pour tout.

ABSTRACT

This thesis contributes towards the development of GPU-accelerated direct and iterative solvers for computational wave scattering. This research was carried out in the context of experimental *Finite Element* code *HELM* developed in a computational stochastic research group.

The research extends the physical modeling capabilities of the *HELM* C++ framework by formalizing a scattered-field weak formulation with *Perfectly Matched Layers* to accurately model complex, heterogeneous media.

The core of the work investigates two primary solution strategies on modern GPU architectures.

First, a numerical performance evaluation of GPU-accelerated direct solvers is presented, demonstrating that the *NVIDIA cuDSS* library offers a performance advantage over both its predecessor, *cuSOLVER*, and CPU-based solvers like *KLU2*, achieving speedups of up to an order of magnitude on a single GPU.

Second, an iterative solver based on a *Multigrid Shifted Laplacian* preconditioner is explored. In particular, an implementation of this preconditioner in the *MueLu* package from *Trilinos* is used. Several contributions were made to the *MueLu* implementation to enable its use with general ordinal types, as well as to permit a nearly complete GPU-resident multigrid V-cycle for complex arithmetic.

A comparative scaling analysis reveals that while the GPU direct solver is faster for the tested 2D problems, the iterative multigrid method exhibits superior asymptotic complexity, suggesting it is a viable path for large-scale 3D and parallel problems. Key limitations, including the parallel scalability of the *Restricted Additive Schwarz* smoother and remaining host-bound computations, are identified. Future work could focus on integrating *Optimized Schwarz methods* and achieving a fully GPU-resident workflow by offloading the coarse-grid solve and matrix reordering.

CHAPTER 1

PROBLEM SETTING

1.1 Simulating Time-Harmonic Waves

The propagation of waves is an ubiquitous phenomenon, fundamental to our understanding of the physical world. From the light that illuminates our surroundings to the seismic tremors that reveal the Earth's structure, wave dynamics are described by a family of partial differential equations. A particularly important case arises when a system is excited by a source oscillating at a single, constant frequency. In such scenarios, the governing physics can be simplified into the time-harmonic wave equation, a powerful model that finds applications across a vast spectrum of scientific and engineering disciplines [1, 2].

However, the elegant simplicity of the continuous mathematical model lies the complexity of its application. For nearly all real-world scenario which involve intricate geometries, heterogeneous materials, or complex boundary interactions, finding an analytical solution is intractable. This necessitates a transition from the continuous domain of physics to the discrete world of computation. Numerical techniques, such as the *Finite Difference Method* (FDM) or, as used in this work, the *Finite Element Method* (FEM), are employed to discretize the problem [3, 4]. This process transforms the differential equation into a system of linear algebraic equations. While conceptually straightforward, this step presents its own formidable challenge: the resulting matrix is often large and sparse. The central focus of this thesis is therefore the development, analysis, and optimization of robust and efficient numerical methods for solving these large-scale linear systems [5].

The demand for such effective solutions is not merely academic; it is driven by urgent needs in technology and science. A particularly pressing challenge, and the primary motivation for this research, lies in nanoscale engineering. In the semiconductor industry, accurately modeling the diffraction of electromagnetic waves is critical for the *optical metrology* [6] of next-generation components like *Gate-All-Around Field-Effect Transistors* (GAAFETs). As device features shrink to the order of a few nanometers, a deep and predictive understanding of wave-material interactions is crucial for advancing fabrication processes.

This challenge, while acute at the nanoscale, is mirrored at the planetary scale. In geophysics, wave propagation is the primary tool for non-invasively probing the Earth's interior. Seismic events generate compressional (P-waves) and shear (S-waves), whose velocities are functions of the density and elasticity of subterranean layers. By modeling the reflection and refraction of these waves through a highly heterogeneous medium, geologists can construct detailed maps of the planet's crust, mantle, and core, providing insights into everything from resource exploration to plate tectonics [2].

Furthermore, the same fundamental principles of wave propagation govern the fields of optics and photonics. A camera lens, for instance, is a precisely engineered system designed to control light waves through refraction, focusing a sharp image onto a sensor. Concurrently, the camera's aperture introduces diffraction, a wave effect that fundamentally limits the ultimate resolution of any optical system. Designing advanced optical components, from microscope objectives to the lenses used in semiconductor lithography, relies heavily on the accurate simulation of these phenomena [7].

Across these diverse applications, a common thread emerges: the need to solve the Helmholtz equation, the specific form of the time-harmonic wave equation [8]. The development of scalable and efficient solvers for this equation is therefore a critical enabling technology, unlocking new possibilities in fields as disparate as electronics, seismology, and optics. This thesis contributes to this effort by focusing on the unique challenges presented by high-frequency electromagnetic wave simulation in the context of modern semiconductor manufacturing.

1.2 Application: Simulating Next-Generation Transistors

The primary motivation for this work is the semiconductor industry's transition to new transistor architectures, such as the *Gate-All-Around FET* (GAAFET), which is succeeding the mainstream FinFET technology [9]. This evolution, illustrated in Figure 1.1, is driven by the need for better electrostatic control in smaller, more complex geometries. As a nascent technology, GAAFETs are highly susceptible to nanoscale manufacturing variations, where minor imperfections can lead to unpredictable device performance.

To quantify the impact of this manufacturing uncertainty, statistical methods are employed. These techniques require vast datasets to generate reliable probabilistic models of device parameters. The physical method for gathering such data is optical metrology, a technique where the repeating array of transistors on a wafer is treated as a diffraction grating. By analyzing the scattered light pattern, key geometric properties can be inferred. However, this physical characterization is too slow to generate the thousands of data points required for statistical analysis.

Therefore, high-fidelity numerical simulation via the *Finite Element Method* (FEM) is one of the viable path to produce the large datasets needed. Simulating the optical metrology process requires solving the time-harmonic Maxwell equations, which directly leads to the large-scale Helmholtz problems central to this thesis.

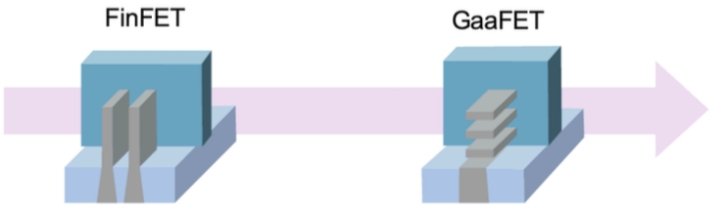


Figure 1.1: The evolution of transistor architectures from planar to FinFET and finally to Gate-All-Around (GAAFET), driven by the need for improved electrostatic control at smaller nodes. Figure extracted from [9].

1.3 The *HELM*-Performance Simulation Framework

To perform the intensive simulations required by this research, this work utilizes the *HELM* finite element framework, developed by *M. Arnst* and *R. Tomasetti*. *HELM* is a C++ application specifically

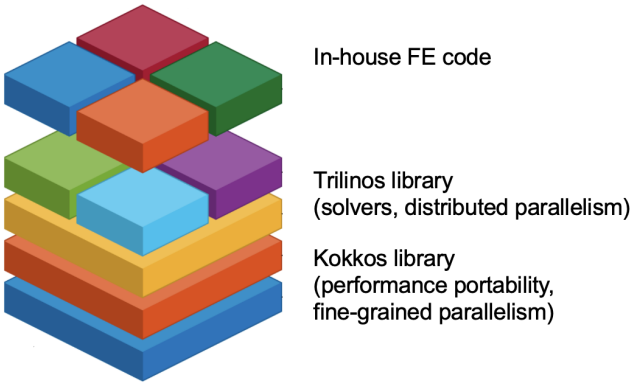


Figure 1.2: Illustration of the library hierachy on which *HELM* is built.

designed to orchestrate the assembly and solution of the large-scale linear systems that arise from FEM formulations of wave problems.

At its core, the framework is architected upon the *Trilinos* library ecosystem, a comprehensive suite of C++ packages dedicated to creating scalable, parallel solutions for scientific and engineering problems [10]. This layered software architecture, visualized in Figure 1.2, is what enables *HELM* to achieve high performance on diverse and modern computing hardware. The key components of this stack are:

- ***Kokkos*:** At the lowest level is the *Kokkos* ecosystem, which serves as the performance portability layer [11]. *Kokkos* provides a programming model that allows developers to write a single piece of code that can be compiled to run efficiently on CPUs, *NVIDIA* GPUs, and other accelerators. It manages on-node parallelism (e.g., threads) and data layout, ensuring that computations are optimized for the specific architecture being used. Computationally intensive kernels, such as the Sparse Matrix-Vector product (SpMV), are handled by the *Kokkos-Kernels* sub-package.

- ***Tpetra*:** Central to *Trilinos* distributed computing capabilities is the *Tpetra* package. *Tpetra*provides the concrete implementation of distributed linear algebra objects. It defines the fundamental data structures, such as the sparse matrix (`Tpetra::CrsMatrix`) and dense multivectors (`Tpetra::MultiVector`), that are partitioned across multiple MPI processes. Crucially, these *Tpetra* objects are built on top of *Kokkos*, meaning a single `Tpetra::CrsMatrix` is both distributed across a cluster via MPI and capable of performing its on-node computations on a GPU via *Kokkos*. It acts as the essential bridge between the distributed-memory world of MPI and the shared-memory parallelism of a single node.

- ***Trilinos*Solvers (*Belos*, *Muelu*, *Amesos2*):** The higher-level numerical algorithm packages within *Trilinos* operate on the distributed data structures provided by *Tpetra*. For this work, we use the *Belos* package for Krylov iterative solvers, the *MueLu* package for algebraic multigrid preconditioning, and the *Amesos2* package for direct solver interfaces. These libraries provide the mathematical machinery for solving the linear system.

- ***HELM*:** At the top of the stack, *HELM* is responsible for the physics-based tasks. It defines the geometry, discretizes the problem using the *Finite Element Method*, and performs the assembly process, which involves populating the entries of a distributed *Tpetra* matrix. Once the system is assembled, HELM hands it off to the *Trilinos* solver stack for the solution phase.

This modular, layered design is what enables *HELM* to tackle computationally demanding simulations, providing a path to scalable performance.

1.4 Challenges and Work Plan

At the start of this thesis, the applications available in *HELM* included Poisson, beam and non-linear icesheet problems. Relevant to wave propagation, applications included a one-dimensional homogeneous wave and a two-dimensional solution for the diffraction of an electromagnetic wave by a cylinder problem (Mie scattering). *HELM* contains an extensive suite of tests. For many applications, these tests verify the numerical solution by comparing it with an analytical solution. Notably, a discrepancy remained in the Mie test case in which the FEM solver result did not match the analytical solution.

While the *HELM* library provides a robust foundation, further development was required in key areas to meet the challenges of this project.

- **Extending Physical Modeling of Wave Scattering in *HELM***: Two main challenges were identified: First, extending the wave propagation applications, in particular, formalizing a scattered field formulation and introducing Perfectly Matched Layers (PMLs), as needed to fully model GAAFET structures. Second, it was necessary to revisit both the analytical benchmark and the numerical implementation of the Mie scattering problem to establish a reliable verification baseline for subsequent work.
- **Exploring direct and iterative solvers to achieve GPU-resident Simulation**: *HELM*'s existing architecture was hybrid; while FEM assembly was performed on the GPU, the linear solver phase remained entirely on CPU (apart from a interface to a small part of *cuSOLVER* through *Amesos2*). A primary objective of this work was therefore to explore how the solution process can be migrated to the device. *HELM* is built on *Trilinos* which combines MPI with OpenMP, HIP and Cuda backends for on-node parallelism. Thus, the integration of GPU-accelerated direct and iterative solvers available in *Trilinos* was studied. For Cuda, in addition, Cuda solver libraries, in particular *cuSOLVER* and the new *cuDSS*, were evaluated. The goal is to create a GPU-resident computation, eliminating the costly data transfers of large matrices between the GPU and CPU, which currently negate many benefits of GPU acceleration.

The structure of this thesis is therefore organized to logically address these challenges.

1.5 Layout of the Thesis

Chapter 1: Physical and Mathematical Models

In this chapter, the mathematical foundation for the simulation of electromagnetic wave scattering is established. The governing equations are derived, beginning with Maxwell equations and culminating in the Helmholtz equation. A key discussion is dedicated to the total versus scattered field formulations, wherein the necessity of the latter for problems involving incident fields in heterogeneous media is explained. The derivation of the weak (variational) formulation, which forms the basis for the *Finite Element Method* is then presented. A significant portion of the chapter is focused on the critical issue of domain truncation, detailing the theory and implementation of Perfectly Matched Layers (PMLS). The chapter concludes with the full weak formulation for a 2D TM scattering problem and a semi-analytical solution to the Mie scattering problem, which is used as a benchmark for numerical validation.

Chapter 2: High-Performance Direct Solvers on GPUs

This chapter provides an in-depth analysis of direct solution methods, with a strong focus on GPU acceleration. The core methodology of sparse LU factorization and its implementation within the

Trilinos Amesos2 library is introduced. Following this, two prominent *NVIDIA* libraries for GPU-accelerated direct solves are explored: *cuSOLVER* and the more recent *cuDSS*. A central theme is the critical role of reordering algorithms in minimizing fill-in and reducing computational cost. The performance of these algorithms is analyzed and the chapter concludes with a solver performance analysis in which the GPU direct solvers are benchmarked against CPU-based methods on both a Poisson test case and the Helmholtz (Mie scattering) problem to evaluate their efficiency and memory usage.

Chapter 3: Advanced Iterative Solvers and Multigrid Preconditioning

In this chapter, the more scalable approach of iterative solvers, which is essential for tackling large-scale problems, is investigated. The GMRES method is introduced, and the reasons why standard preconditioning techniques often fail for the indefinite Helmholtz equation are explained. The core of the chapter is a detailed examination of the Shifted Laplacian preconditioner for Helmholtz problems. The construction of this method is detailed, and its dependence on an inner approximate solver, for which an algebraic multigrid (AMG) method is used and is discussed. The key components of the multigrid hierarchy are described, including on-node smoothers (*Jacobi*, *Gauss-Seidel*) and the MPI-parallel *Restricted Additive Schwarz* (RAS) smoother combined with *ILU* factorizaiton.

An extensive parameter search and optimization, performed to find the most effective configuration for the multigrid preconditioner, is then presented. The limitations of the classical RAS smoother for wave problems are discussed, and a path toward more advanced methods is explored. Finally, the challenges and outcomes of GPU acceleration for this complex iterative solver are analyzed, including the debugging and correction of the underlying *Trilinos* libraries to enable a more complete GPU-resident workflow. The chapter concludes with a performance comparison and scaling analysis against the direct solvers from Chapter 2, providing a comprehensive outlook on the most viable solution strategies.

CHAPTER 2

PHYSICAL MODELS

Contents

2.1 Introduction to the Formulation of Wave Propagation

As established in the previous chapter, the characterization of GAAFET structures is performed via optical metrology, which involves illuminating the device with a light source and analyzing the scattered field. The fundamental physics governing the propagation of this light and its interaction with the nanoscale materials of the transistor is described by Maxwell equations. This section details the derivation of the governing equation which forms the basis for the numerical simulations that will be used to model the GAAFET under incoming light.

2.2 Governing Equations for Optical Simulation

To numerically simulate the interaction between light and a GAAFET structure, it is essential to establish the governing mathematical model. This model begins with the fundamental laws of electromagnetism, Maxwell equations, and is systematically transformed into a form suitable for numerical computation by the *Finite Element Method* (FEM). This section details the derivation of the time- and frequency-domain equation for an inhomogeneous medium, which correctly accounts for the spatially varying material properties of the device.

2.2.1 The Time-Dependent Wave Equation for Inhomogeneous Media

The time-domain approach is the foundation of methods like FDTD, which can capture transient effects and obtain a spectral response from a single simulation. This section derives the time-dependent vector wave equation for an inhomogeneous medium.

We begin with Maxwell curl equations (Section 1.2.2 of [4]) in the time domain, assuming a source-free region ($\mathbf{J} = 0, \rho = 0$):

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{2.1}$$

$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} \tag{2.2}$$

$$\nabla \cdot \mathbf{D} = 0 \tag{2.3}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{2.4}$$

The constitutive relations (Section 1.2.5 of [4]) explicitly include the spatial dependence of the material properties,

$$\mathbf{D}(\mathbf{r}, t) = \varepsilon(\mathbf{r})\mathbf{E}(\mathbf{r}, t)$$

and

$$\mathbf{B}(\mathbf{r}, t) = \mu(\mathbf{r})\mathbf{H}(\mathbf{r}, t).$$

To derive a single equation for the electric field $\mathbf{E}$, we take the time derivative of Ampere Law (Equation (2.2)):

$$\frac{\partial}{\partial t}(\nabla \times \mathbf{H}) = \frac{\partial}{\partial t}\left(\frac{\partial \mathbf{D}}{\partial t}\right) = \frac{\partial^2 \mathbf{D}}{\partial t^2} \tag{2.5}$$

Substituting the constitutive relation for $\mathbf{D}$ on the right-hand side gives:

$$\nabla \times \left(\frac{\partial \mathbf{H}}{\partial t}\right) = \varepsilon(\mathbf{r})\frac{\partial^2 \mathbf{E}}{\partial t^2} \tag{2.6}$$

Next, we need an expression for $\frac{\partial \mathbf{H}}{\partial t}$. We can find this from Faraday Law (Equation (2.1)) by incorporating the constitutive relation for $\mathbf{B}$:

$$\nabla \times \mathbf{E} = -\frac{\partial}{\partial t}(\mu(\mathbf{r})\mathbf{H}) = -\mu(\mathbf{r})\frac{\partial \mathbf{H}}{\partial t} \tag{2.7}$$

Solving for $\frac{\partial \mathbf{H}}{\partial t}$ yields:

$$\frac{\partial \mathbf{H}}{\partial t} = -\frac{1}{\mu(\mathbf{r})}\nabla \times \mathbf{E} \tag{2.8}$$

Finally, we substitute this expression back into Equation (2.6):

$$\nabla \times \left(-\frac{1}{\mu(\mathbf{r})}\nabla \times \mathbf{E}\right) = \varepsilon(\mathbf{r})\frac{\partial^2 \mathbf{E}}{\partial t^2} \tag{2.9}$$

Rearranging the terms, we arrive at the time-dependent vector wave equation for the electric field in an inhomogeneous medium:

$$\nabla \times \left(\frac{1}{\mu(\mathbf{r})}\nabla \times \mathbf{E}\right) + \varepsilon(\mathbf{r})\frac{\partial^2 \mathbf{E}}{\partial t^2} = 0 \tag{2.10}$$

Similarly, a dual derivation yields the wave equation for the magnetic field:

$$\nabla \times \left(\frac{1}{\varepsilon(\mathbf{r})}\nabla \times \mathbf{H}\right) + \mu(\mathbf{r})\frac{\partial^2 \mathbf{H}}{\partial t^2} = 0 \tag{2.11}$$

2.2.2 Maxwell Equations in the Frequency Domain

From Section 1.2.4 of [4], the frequency-domain (time-harmonic) representation of Maxwell curl equations, which presumes a field dependency of $e^{i\omega t}$ is given by :

$$\nabla \times \mathbf{E} = -i\omega\mathbf{B} \tag{2.12}$$
$$\nabla \times \mathbf{H} = i\omega\mathbf{D} \tag{2.13}$$

2.2.3 Derivation of the General Wave Equation in the Frequency Domain

Our goal is to obtain a single equation for the electric field $\mathbf{E}$. We obtain the General Wave Equation in the Frequency Domain from its temporal counterpart by noting that

$$\frac{\partial^2 \mathbf{H}}{\partial t^2} = -\omega^2\mu(\mathbf{r})\mathbf{H}$$

and

$$\frac{\partial^2 \mathbf{E}}{\partial t^2} = -\omega^2\varepsilon(\mathbf{r})\mathbf{E}$$

and using Equation (2.10),

$$\nabla \times \left(\frac{1}{\mu(\mathbf{r})}\nabla \times \mathbf{E}\right) - \omega^2\varepsilon(\mathbf{r})\mathbf{E} = 0 \tag{2.14}$$

This is the general vector Helmholtz equation, also known as the "curl-curl" equation, for the electric field in an inhomogeneous, source-free medium. Unlike the simpler form that will be developed in Section 2.2.4, the $\frac{1}{\mu(\mathbf{r})}$ term remains inside the first curl operator because it varies with position.

Similarly for the magnetic density,

$$\nabla \times \left(\frac{1}{\varepsilon(\mathbf{r})} \nabla \times \mathbf{H} \right) - \omega^2 \mu(\mathbf{r}) \mathbf{H} = 0 \tag{2.15}$$

This equation exhibits a clear duality with its electric field counterpart (Equation (2.14)), where the roles of permittivity $\varepsilon(\mathbf{r})$ and permeability $\mu(\mathbf{r})$ are interchanged. In a numerical simulation, it is typically sufficient to solve for only one of the fields (either **E** or **H**), as the other can be directly calculated from the result using Maxwell curl equations.

2.2.4 The Simplified Helmholtz Equation for Homogeneous Media

For completeness, it is instructive to examine the specific conditions under which the general "curl-curl" equation (Equation (2.14)) reduces to the more familiar standard Helmholtz form.

The key step in this reduction is factoring the $1/\mu$ term out of the first curl operator. This is mathematically permissible only if the magnetic permeability μ is spatially constant. It is important to note that this specific step does not require the permittivity ε to be constant, as it appears in a separate term. If the medium has a constant permeability, we can write:

$$\frac{1}{\mu} \nabla \times (\nabla \times \mathbf{E}) - \omega^2 \varepsilon \mathbf{E} = 0 \tag{2.16}$$

Using the vector identity $\nabla \times (\nabla \times \mathbf{E}) = \nabla(\nabla \cdot \mathbf{E}) - \nabla^2 \mathbf{E}$, we get:

$$\frac{1}{\mu} (\nabla(\nabla \cdot \mathbf{E}) - \nabla^2 \mathbf{E}) - \omega^2 \varepsilon \mathbf{E} = 0 \tag{2.17}$$

In a source-free region ($\rho = 0$), Gauss Law ($\nabla \cdot \mathbf{D} = 0$) becomes $\nabla \cdot (\varepsilon \mathbf{E}) = 0$. If we now also assume that permittivity ε is constant, this simplifies to $\varepsilon(\nabla \cdot \mathbf{E}) = 0$, which implies $\nabla \cdot \mathbf{E} = 0$. Under both of these homogeneity assumptions, the equation becomes:

$$-\frac{1}{\mu} \nabla^2 \mathbf{E} - \omega^2 \varepsilon \mathbf{E} = 0 \tag{2.18}$$

Multiplying by $-\mu$ yields the vector Helmholtz equation:

$$\nabla^2 \mathbf{E} + \omega^2 \mu \varepsilon \mathbf{E} = 0 \quad \Longrightarrow \quad \nabla^2 \mathbf{E} + k^2 \mathbf{E} = 0. \tag{2.19}$$

wher $k = \omega \sqrt{\mu \varepsilon}$ is called the wavenumber.

As the derivation shows, the reduction to the standard Helmholtz form for the electric field requires constant magnetic permeability and permittivity. By duality, the equivalent simplification for the magnetic field wave equation,

$$\nabla^2 \mathbf{H} + k^2 \mathbf{H} = 0, \tag{2.20}$$

requires constant properties.

2.3 Total vs. Scattered Field Formulations

To analyze the electromagnetic challenges of GAAFETs, we choose to work in the frequency domain. This approach is effective for problems concerned with time-harmonic and single-frequency responses.

While the Helmholtz equation provides the governing mathematical model, its numerical implementation requires careful consideration of the problem physics, particularly how sources and boundaries are handled. Two primary approaches exist: the total field formulation and the scattered field formulation. This section introduces both, demonstrating why the scattered field approach is essential for modeling scattering from heterogeneous objects like GAAFETs.

2.3.1 The Total Field Formulation: A Homogeneous Case

Let us first consider the simplest case: a 1D problem solving for a wave traveling from left to right in a perfectly homogeneous medium, as illustrated in Figure 2.1.

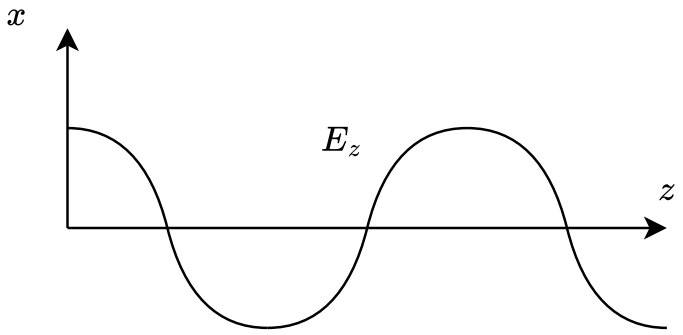


Figure 2.1: Illustration of a traveling wave from left to right within a homogeneous medium. A Dirichlet boundary condition imposes the incoming wave $E(0) = E_0$ on the left, and a Robin condition on the right acts as an absorbing boundary.

In this non-reflecting scenario, the total electric field E is simply the incident wave itself. The problem can be defined using Equation (2.14) in one dimension :

$$\begin{cases} \dfrac{d^2E_z}{dx^2} + k^2E_z = 0, & \text{for } x \in [0, L] \\ E_z(0) = E_0, & \text{at } x = 0 \\ \dfrac{dE_z}{dx}(L) = -ikE_z(L), & \text{at } x = L \end{cases} \tag{2.21}$$

The general solution to the homogeneous Helmholtz equation is $E(x) = Ae^{-ikx} + Be^{ikx}$, representing a superposition of right- and left-traveling waves. The Robin condition at $x = L$ is specifically designed to absorb right-traveling waves, ensuring $B = 0$. The Dirichlet condition at $x = 0$ then sets the amplitude $A = E_0$. The analytical solution is therefore trivial:

$$E(x) = E_0e^{-ikx} \tag{2.22}$$

This represents a simple traveling wave propagating towards the positive x-direction with amplitude E_0 and wavenumber k. The total field formulation is well-suited for this scenario where the wave properties are uniform and there are no reflections generated within the domain.

2.3.2 Limitations in Heterogeneous Media

The simplicity of the total field formulation breaks down when the medium is heterogeneous which the scenario when simulating a GAAFET. Consider a wave incident on a dielectric slab, as shown in Figure 2.2. The change in material properties at the interface ($x = l$) will cause both reflection and transmission.

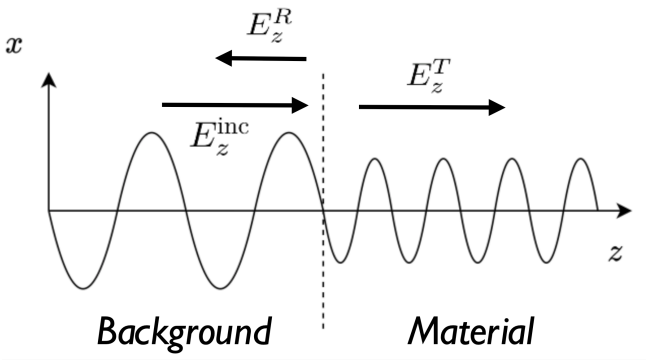


Figure 2.2: Illustration of the different electric fields present in a heterogeneous domain. An incident wave E_z^{inc} generates a reflected wave E_z^R in the background medium and a transmitted wave E_z^T in the scattering medium.

In the background region ($x < l$), the total field is now a superposition of the incident wave and the reflected wave. The analytical form of the total field is:

$$E_{tot}(x) = \begin{cases} E_0 e^{-ik_{bg}x} + Re^{ik_{bg}x}, & \text{for } x \in [0, l] \\ Te^{-ik_{sc}x}, & \text{for } x \in [l, L] \end{cases} \tag{2.23}$$

where E_0 is the known incident amplitude, but the reflection (R) and transmission (T) amplitudes are unknown and depend on the properties of the scattering object.

Herein lies the problem: if we apply a simple Dirichlet condition $E_{tot}(0) = E_0$ as before, we force the solution to satisfy

$$E_0 e^{-ik_{bg}(0)} + Re^{ik_{bg}(0)} = E_0.$$

This simplifies to

$$E_0 + R = E_0,$$

which incorrectly implies that the reflection coefficient R must be zero. The total field formulation is therefore unsuitable because the boundary conditions required to generate the incident wave interfere with the physical reflection that is part of the solution we seek.

2.3.3 The Scattered Field Formulation: A Solution

To solve the conflict between imposing a source and allowing for physical reflection, the problem is reformulated using the scattered field decomposition [4]. The total field is split into two distinct parts: a known incident field and an unknown scattered field.

$$E_{tot} = E_{inc} + E_{scat} \tag{2.24}$$

- **The Incident Field (E_{inc})** is defined as the field that would exist in the absence of any scattering object. It is a known quantity, typically a plane wave like $E_{inc}(x) = E_0 e^{-ik_{bg}x}$, that satisfies the Helmholtz equation in the background medium everywhere.

- **The Scattered Field (E_{scat})** is the unknown quantity we solve for. It represents the disturbance, or change, in the field caused by the presence of the heterogeneous object. Physically, in the background region ($x < l$), E_{scat} is exactly the reflected wave (E_R).

This decomposition elegantly solves the boundary condition paradox mentioned above. At the source boundary ($x = 0$), the total field is $E_{tot}(0) = E_{inc}(0) + E_{scat}(0)$. Since we know $E_{inc}(0) = E_0$, this gives:

$$E_{tot}(0) = E_0 + E_{scat}(0) \tag{2.25}$$

Instead of imposing a condition on the total field, we now impose a condition on the scattered field. Because the scattered field is generated by the object and radiates away from it, it must satisfy an outgoing wave condition at the boundaries of the computational domain. For the boundary at $x = 0$, this means any scattered wave component must be traveling away from the domain (to the left), which is precisely the physical behavior of a reflected wave.

By substituting the decomposition (2.24) into the original Helmholtz equation, one can derive a new governing equation for E_{scat} alone. This new equation correctly models the incident wave as a source term within the PDE itself, rather than as a boundary condition. This change decouples the source mechanism from the boundary physics, allowing the boundary conditions to correctly model the absorption of outgoing scattered waves without interfering with reflections.

2.4 Weak Formulation for the One Dimensional Homogeneous Case

The variational (or weak) formulation is derived from the strong form of the problem by recasting the differential equation as an integral equation. This process reduces the order of the derivatives and naturally incorporates certain types of boundary conditions.

We begin with the strong-form PDE for the homogeneous case:

$$\frac{d^2E}{dx^2} + k^2E = 0 \tag{2.26}$$

We multiply this equation by a complex-conjugated test function $\overline{f}$, which belongs to a suitable function space, and integrate over the domain $[0, L]$:

$$\int_0^L \left(\frac{d^2E}{dx^2}\overline{f} + k^2E\overline{f}\right) dx = 0 \tag{2.27}$$

Next, integration by parts is performed to the second-order derivative term. This "weakens" the derivative requirement on the solution E, shifting one order of differentiation onto the test function f.

$$\left[\frac{dE}{dx}\overline{f}\right]_0^L - \int_0^L \frac{dE}{dx}\frac{d\overline{f}}{dx}dx + \int_0^L k^2E\overline{f}, dx = 0 \tag{2.28}$$

Expanding the boundary term (the term in brackets) gives:

$$\left(\frac{dE}{dx}(L)\overline{f}(L)\right) - \left(\frac{dE}{dx}(0)\overline{f}(0)\right) - \int_0^L \frac{dE}{dx}\frac{d\overline{f}}{dx}dx + \int_0^L k^2E\overline{f}, dx = 0 \tag{2.29}$$

Now we incorporate the boundary conditions from the strong form (Equation (2.21)). This step highlights the crucial distinction between essential (Dirichlet) and natural (Neumann/Robin) boundary conditions.

- **Natural Boundary Condition** (at $x = L$): The Robin condition, $\frac{dE}{dx}(L) = ikE(L)$, involves a derivative and is called a "natural" boundary condition. It can be substituted directly into the boundary term of the weak form.
- **Essential Boundary Condition** (at $x = 0$): The Dirichlet condition, $E(0) = E_0$, specifies the value of the function itself and is called an "essential" boundary condition. This condition is enforced directly on the space of candidate solutions (the trial space). The test function f is chosen from a corresponding test space such that $f \in H^1([0, L])$ where the function is zero at the location of the Dirichlet BC, i.e., $f(0) = 0$. This causes the corresponding boundary term $\frac{dE}{dx}(0)\overline{f}(0)$ to vanish.

Substituting these conditions into Equation (2.29):

$$\big(ikE(L)\overline{f}(L)\big) - \big(\text{vanishes since } \overline{f}(0) = 0\big) - \int_0^L \frac{dE}{dx}\frac{d\overline{f}}{dx}dx + \int_0^L k^2 E\overline{f}, dx = 0 \tag{2.30}$$

Rearranging the terms yields the final weak formulation. The problem is stated as:

Find $E \in H^1([0, L])$ with $E(0) = E_0$:

$$\int_0^L \left(\frac{dE}{dx}\frac{d\overline{f}}{dx} - k^2 E\overline{f}\right) dx - ikE(L)\overline{f}(L) = 0. \tag{2.31}$$

for all test functions $f \in H^1([0, L])$ such that $f(0) = 0$.

2.5 Heterogeneous One Dimensional Helmholtz equation

It is first shown how the Helmholtz equation for a fully one dimensional heterogeneous medium can be derived from the general vector case.

Helmholtz equation for non-constant permittivity and permeability

The propagation of time-harmonic electromagnetic waves in a source-free, linear, and isotropic medium is governed by the vector Helmholtz equation for the electric field **E**:

$$\nabla \times \left(\frac{1}{\mu(\mathbf{r})}\nabla \times \mathbf{E}(\mathbf{r})\right) - \omega^2 \varepsilon(\mathbf{r})\mathbf{E}(\mathbf{r}) = 0 \tag{2.32}$$

where $\varepsilon(\mathbf{r})$ and $\mu(\mathbf{r})$ are the spatially varying permittivity and permeability, respectively.

To focus on the one-dimensional problem, several simplifying assumptions are introduced. A Transverse Magnetic (TM_z) polarized wave is considered, where the electric field has only a z-component, $\mathbf{E} = E_z(x)\hat{\mathbf{z}}$. Furthermore, the medium is assumed to be z-invariant ($\partial/\partial z = 0$), and its properties are assumed to vary only along the x-direction, such that $\varepsilon = \varepsilon(x)$ and $\mu = \mu(x)$.

Under these conditions, the curl-curl term simplifies directly. For $\mathbf{E} = E_z(x)\hat{\mathbf{z}}$, the first curl is

$$\nabla \times \mathbf{E} = -\frac{dE_z}{dx}\hat{\mathbf{y}}.$$

The full term becomes:

$$\nabla \times \left(\frac{1}{\mu(x)}\left(-\frac{dE_z}{dx}\hat{\mathbf{y}}\right)\right) = \frac{d}{dx}\left(\frac{1}{\mu(x)}\frac{dE_z}{dx}\right)\hat{\mathbf{z}} \tag{2.33}$$

The vector Helmholtz equation thus reduces to the following 1D scalar equation for the total field E_z:

$$\frac{d}{dx}\left(\frac{1}{\mu(x)}\frac{dE_z}{dx}\right) + \omega^2\varepsilon(x)E_z = 0 \tag{2.34}$$

Scattered field formulation

To address the formulation problem illustrated in Section 2.3.2, the total field E_{tot} is decomposed into two components: a known incident field (E_{inc}) and an unknown scattered field (E_{scat}):

$$E_{tot} = E_{inc} + E_{scat} \tag{2.35}$$

as introduced in Section 2.3.3.

The incident field, E_{inc}, is defined as the field that would exist in the absence of the scatterer, i.e., in a homogeneous background medium with properties μ_{bg} and ε_{bg}. By definition, E_{inc} is a solution to the homogeneous Helmholtz equation:

$$\frac{d}{dx}\left(\frac{1}{\mu_{bg}}\frac{dE_{inc}}{dx}\right) + \omega^2\varepsilon_{bg}E_{inc} = 0 \tag{2.36}$$

For a constant background, this simplifies to the familiar

$$\frac{d^2E_{inc}}{dx^2} + k_{bg}^2E_{inc} = 0,$$

where $k_{bg} = \omega\sqrt{\mu_{bg}\varepsilon_{bg}}$. As derived earlier, a simple plane wave solution is $E_{inc}(x) = E_0e^{-ik_{bg}x}$.

The decomposition $E_{tot} = E_{inc} + E_{scat}$ is now substituted into the total field equation (2.34). Rearranging for the unknown E_{scat} gives:

$$\frac{d}{dx}\left(\frac{1}{\mu}\frac{dE_{scat}}{dx}\right) + \omega^2\varepsilon E_{scat} = -\left[\frac{d}{dx}\left(\frac{1}{\mu}\frac{dE_{inc}}{dx}\right) + \omega^2\varepsilon E_{inc}\right] \tag{2.37}$$

We simplify the right-hand side by adding and subtracting terms related to the background medium equation (2.36), which is identically zero. The right-hand side becomes:

$$S(x) = -\left[\frac{d}{dx}\left(\frac{1}{\mu}\frac{dE_{inc}}{dx}\right) + \omega^2\varepsilon E_{inc}\right] + \left[\frac{d}{dx}\left(\frac{1}{\mu_{bg}}\frac{dE_{inc}}{dx}\right) + \omega^2\varepsilon_{bg}E_{inc}\right] \tag{2.38}$$

Combining terms yields the source of the scattered field:

$$S(x) = -\frac{d}{dx}\left(\left(\frac{1}{\mu} - \frac{1}{\mu_{bg}}\right)\frac{dE_{inc}}{dx}\right) - \omega^2(\varepsilon - \varepsilon_{bg})E_{inc} \tag{2.39}$$

This gives the final strong form for the scattered field, which is valid for heterogeneous $\mu(x)$ and $\varepsilon(x)$:

$$\frac{d}{dx}\left(\frac{1}{\mu}\frac{dE_{scat}}{dx}\right) + \omega^2\varepsilon E_{scat} = -\frac{d}{dx}\left(\left(\frac{1}{\mu} - \frac{1}{\mu_{bg}}\right)\frac{dE_{inc}}{dx}\right) - \omega^2(\varepsilon - \varepsilon_{bg})E_{inc} \tag{2.40}$$

where $\frac{1}{\mu(x)} - \frac{1}{\mu_{bg}}$ and $\varepsilon(x) - \varepsilon_{bg}$ represent the material contrast. The right-hand side acts as a source term that is non-zero only within the scattering object where the material properties differ from the background. As a result, the scatterer can be viewed as a source in the formulation and the wave radiates from it towards the boundaries.

To form a well-posed problem, this PDE must be supplemented with boundary conditions that ensure the scattered field is purely ‘outgoing”. This leads to the strong form of the boundary value problem:

$$\begin{cases} \dfrac{d}{dx}\left(\dfrac{1}{\mu}\dfrac{dE_{scat}}{dx}\right) + \omega^2 \varepsilon E_{scat} = S(x) & \text{for } x \in [0, L] \\ \text{Outgoing wave condition} & \text{at } x = 0 \\ \text{Outgoing wave condition} & \text{at } x = L \end{cases} \tag{2.41}$$

and considering Robin boundary conditions for the absorption:

$$\begin{cases} \dfrac{d}{dx}\left(\dfrac{1}{\mu}\dfrac{dE_{scat}}{dx}\right) + \omega^2 \varepsilon E_{scat} = S(x) & \text{for } x \in [0, L] \\ \dfrac{1}{\mu_{\text{bg}}}\dfrac{dE_{scat}}{dx}(0) + ik_{bg}E_{scat}(0) = 0 & \text{at } x = 0 \\ \dfrac{1}{\mu_{\text{sc}}}\dfrac{dE_{tot}}{dx}(L) - ik_{sc}E_{tot}(L) = 0 & \text{at } x = L \end{cases} \tag{2.42}$$

The material properties are defined piecewise across the domain:

$$\mu(x) = \begin{cases} \mu_{bg}, & x \in [0, l) \\ \mu_{sc}, & x \in [l, L] \end{cases} \quad \text{and} \quad \varepsilon(x) = \begin{cases} \varepsilon_{bg}, & x \in [0, l) \\ \varepsilon_{sc}, & x \in [l, L] \end{cases} \tag{2.43}$$

The wavenumbers k_{bg} and k_{sc} in the boundary conditions are defined as $k_{bg} = \omega\sqrt{\mu_{bg}\varepsilon_{bg}}$ and $k_{sc} = \omega\sqrt{\mu_{sc}\varepsilon_{sc}}$, respectively.

It is critical to analyze the structure of the boundary conditions in the final scattered field problem formulation (Equation (2.43)). The problem now incorporates two distinct absorbing boundary conditions, each with a specific physical purpose.

- **On the left boundary** ($x = 0$), the incident field source is handled by the right-hand side of the PDE, and the boundary condition is applied only to the scattered field. This absorbing condition, matched to the background wavenumber k_{bg}, ensures that any wave component of the scattered field traveling toward the negative x-direction (i.e., a reflection) is perfectly absorbed and removed from the simulation domain.
- **On the right boundary** ($x = L$), the situation is more subtle. The underlying reason for the different formulation is that a simple, first-order ABC can only perfectly absorb a plane wave of a single wavenumber. However, the scatter wave exiting the domain at $x = L$ is a complex superposition of the transmitted incident field and the forward-scattered field. This is not a simple wave that our ABC can handle correctly if applied to E_{scat} alone. As a practical workaround, the boundary condition is instead applied to the total field, E_{tot}, and is matched to the scatterer wavenumber, k_{sc}. The assumption is that the total field just inside this boundary behaves like a simple wave with wavenumber k_{sc}.

In conclusion, using the scattered field formulation is crucial when dealing with scattering in heterogeneous media, as it provides a robust framework to correctly model the source of the incident wave while allowing physical boundary interactions — reflection and transmission — to be calculated accurately.

2.6 Weak Formulation for the One Dimensional Heterogeneous Scattering Problem

The derivation of the weak formulation begins with the governing equation for the total field. This approach is sufficiently general to handle spatial variations in both permeability $\mu(x)$ and permittivity

$\varepsilon(x)$.

The advantage of this strategy is that integration by parts is applied before the total field is split into its incident and scattered components. This is a critical step because the total field, E_z^{tot}, and its corresponding flux term, $(1/\mu)dE_z^{\text{tot}}/dx$, are physically continuous across material interfaces. This physical continuity provides the necessary mathematical smoothness (regularity) to justify applying integration by parts over the entire domain.

Our starting point is therefore the strong form of the 1D Helmholtz equation for the total field in a fully heterogeneous medium (Equation (2.34)):

$$\frac{d}{dx}\left(\frac{1}{\mu(x)}\frac{dE_z^{\text{tot}}}{dx}\right) + \omega^2\varepsilon(x)E_z^{\text{tot}} = 0 \quad \text{for } x \in [0, L] \tag{2.44}$$

This partial differential equation is subject to the absorbing boundary conditions established previously:

$$\left.\frac{d(E_z^{\text{tot}} - E_z^{\text{inc}})}{dx}\right|_{x=0} - ik_{\text{bg}}(E_z^{\text{tot}} - E_z^{\text{inc}})\Big|_{x=0} = 0 \tag{2.45}$$

$$\left.\frac{dE_z^{\text{tot}}}{dx}\right|_{x=L} + ik_{\text{sc}}E_z^{\text{tot}}\Big|_{x=L} = 0 \tag{2.46}$$

With the strong form established, we proceed to the weak formulation.

Derivation of the Weak Formulation

The process begins by multiplying the PDE (Equation (2.44)) by a complex-conjugated test function $\overline{f} \in H^1([0, L])$ and integrating over the domain. Then, integration by parts is applied to the second-order term to yield:

$$\left[\frac{1}{\mu}\frac{dE_z^{\text{tot}}}{dx}\overline{f}\right]_0^L - \int_0^L \frac{1}{\mu}\frac{dE_z^{\text{tot}}}{dx}\frac{d\overline{f}}{dx}dx + \int_0^L \omega^2\varepsilon E_z^{\text{tot}}\overline{f}, dx = 0 \tag{2.47}$$

Next, the total field is decomposed, $E_z^{\text{tot}} = E_z^{\text{sc}} + E_z^{\text{inc}}$. By linearity, the equation is rearranged to isolate all terms involving the unknown scattered field (E_z^{sc}) on the left-hand side and all terms with the known incident field (E_z^{inc}) on the right-hand side. This results in:

$$\begin{aligned}&\int_0^L \left(\frac{1}{\mu}\frac{dE_z^{\text{sc}}}{dx}\frac{d\overline{f}}{dx} - \omega^2\varepsilon E_z^{\text{sc}}\overline{f}\right)dx - \left[\frac{1}{\mu}\frac{dE_z^{\text{sc}}}{dx}\overline{f}\right]_0^L \\ &= -\int_0^L \left(\frac{1}{\mu}\frac{dE_z^{\text{inc}}}{dx}\frac{d\overline{f}}{dx} - \omega^2\varepsilon E_z^{\text{inc}}\overline{f}\right)dx + \left[\frac{1}{\mu}\frac{dE_z^{\text{inc}}}{dx}\overline{f}\right]_0^L\end{aligned} \tag{2.48}$$

The final step is to incorporate the absorbing boundary conditions (Equation (2.45) and Equation (2.46)) into the boundary terms $[\cdots]_0^L$. After substituting the boundary conditions and performing the necessary algebraic grouping, we arrive at the final weak formulation.

Final Weak Formulation

Given E_z^{inc}, find $E_z^{\text{sc}} \in H^1([0, L])$:

$$\begin{aligned}&\int_0^L \left(\frac{1}{\mu}\frac{dE_z^{\text{sc}}}{dx}\frac{d\overline{f}}{dx} - \omega^2\varepsilon E_z^{\text{sc}}\overline{f}\right)dx + i\frac{k_{\text{sc}}}{\mu_{\text{sc}}}E_z^{\text{sc}}(L)\overline{f}(L) + i\frac{k_{\text{bg}}}{\mu_{\text{bg}}}E_z^{\text{sc}}(0)\overline{f}(0)\\ &= -\int_0^L \left(\frac{1}{\mu}\frac{dE_z^{\text{inc}}}{dx}\frac{d\overline{f}}{dx} - \omega^2\varepsilon E_z^{\text{inc}}\overline{f}\right)dx - i\frac{k_{\text{bg}}}{\mu_{\text{sc}}}E_z^{\text{inc}}(L)\overline{f}(L) - \frac{k_{\text{bg}}}{\mu_{\text{bg}}}\frac{dE_z^{\text{inc}}}{dx}(0)\overline{f}(0)\end{aligned} \tag{2.49}$$

such that for all test functions $f \in H^1([0, L])$

2.7 Analytical solution of the One Dimensional Scattering Heterogeneous Problem

The strong form associated the scattering heterogeneous was already discussed in Section 2.5 and Equation (2.43).

In order to compare and validate the formulation developed, the solution using the strong formulation can be obtained analytically as shown below.

Problem Setup

We consider a 1D domain with two regions separated by an interface at $x = l$:

- **Region 1 (Background):** For $x < l$, the medium has material properties ε_1, μ_1. The wavenumber is $k_1 = \omega\sqrt{\mu_1\varepsilon_1}$ and the wave impedance is $Z_1 = \sqrt{\mu_1/\varepsilon_1}$.
- **Region 2 (Scatterer):** For $x > l$, the medium has properties ε_2, μ_2. The wavenumber is $k_2 = \omega\sqrt{\mu_2\varepsilon_2}$ and the wave impedance is $Z_2 = \sqrt{\mu_2/\varepsilon_2}$.

An incident plane wave, E_{inc}, travels from the left and impinges on the interface, producing a reflected wave and a transmitted wave.

Field Representation

The total electric field, E_z^{tot}, in each region is a superposition of the relevant waves. Using the standard convention where e^{-ikx} represents a wave traveling in the positive x-direction:

- **Region 1 ($x < l$):** The total field is the sum of the incident and reflected waves.

$$E_z^{\text{tot}}(x) = E_0 e^{-ik_1 x} + Re^{ik_1 x} \tag{2.50}$$

  where E_0 is the known incident amplitude and R is the unknown reflection amplitude.

- **Region 2 ($x > l$):** The total field is the transmitted wave.

$$E_z^{\text{tot}}(x) = Te^{-ik_2 x} \tag{2.51}$$

  where T is the unknown transmission amplitude.

The **scattered field**, defined as $E_z^{\text{scat}} = E_z^{\text{tot}} - E_z^{\text{inc}}$, is therefore $E_z^{\text{scat}}(x) = Re^{ik_1 x}$ in Region 1.

Applying Interface Continuity Conditions

To solve for the two unknowns, R and T, we enforce the continuity of the tangential electric (E_z) and magnetic (H_y) fields at the interface $x = l$. From Faraday Law, $\nabla \times \mathbf{E} = -i\omega\mu\mathbf{H}$, we find the relation between the fields: $H_y = \frac{1}{i\omega\mu}\frac{dE_z}{dx}$.

The two continuity conditions at $x = l$ are:

1. **Continuity of E_z:**
$$E_0 e^{-ik_1 l} + Re^{ik_1 l} = Te^{-ik_2 l} \tag{2.52}$$

2. **Continuity of H_y:**
$$\frac{1}{i\omega\mu_1}\frac{d}{dx}\left(E_0 e^{-ik_1 x} + Re^{ik_1 x}\right)\Big|_{x=l} = \frac{1}{i\omega\mu_2}\frac{d}{dx}\left(Te^{-ik_2 x}\right)\Big|_{x=l} \tag{2.53}$$

   Evaluating the derivatives leads to the simplified form:
$$\frac{k_1}{\mu_1}(E_0 e^{-ik_1 l} - Re^{ik_1 l}) = \frac{k_2}{\mu_2}(Te^{-ik_2 l}) \tag{2.54}$$

Solving for Reflection and Transmission Coefficients

We now have a system of two linear equations (Equation (2.52)) and (Equation (2.54)) for the unknowns R and T. Solving this system yields the reflection amplitude:

$$R = E_0 \left(\frac{\frac{k_1}{\mu_1} - \frac{k_2}{\mu_2}}{\frac{k_1}{\mu_1} + \frac{k_2}{\mu_2}}\right) e^{-2ik_1 l} \tag{2.55}$$

By substituting the definitions for wave impedance, $Z_1 = \omega\mu_1/k_1$ and $Z_2 = \omega\mu_2/k_2$, this simplifies to the well-known formula:

$$R = E_0 \left(\frac{Z_2 - Z_1}{Z_2 + Z_1}\right) e^{-2ik_1 l} \tag{2.56}$$

Similarly, solving for the transmission amplitude T gives:

$$T = E_0 \left(\frac{2Z_2}{Z_2 + Z_1}\right) e^{i(k_2 - k_1)l} \tag{2.57}$$

The primary utility of this analytical solution is to provide a clear justification for adopting the scattered-field formulation. As its mathematical form shows, the total field at any boundary is a function of the unknown reflection, R. This makes it impossible to prescribe a correct boundary condition directly on the total field, thus necessitating the scattered-field approach that is central to our method. It also serves as the benchmark for the numerical results presented in Figure 2.3 as well as a comparison between the analytical and the numerical solutions in Figure 2.4.

2.8 Domain Truncation and the Perfectly Matched Layer

Electromagnetic scattering problems are inherently defined in open, unbounded domains, where scattered waves propagate outwards to infinity. Numerical methods such as the *Finite Element Method*, however, require a finite computational domain. This necessitates the use of a domain truncation strategy. The primary challenge is to terminate the simulation domain in such a way that outgoing waves are absorbed without generating non-physical reflections from the artificial boundary, which would

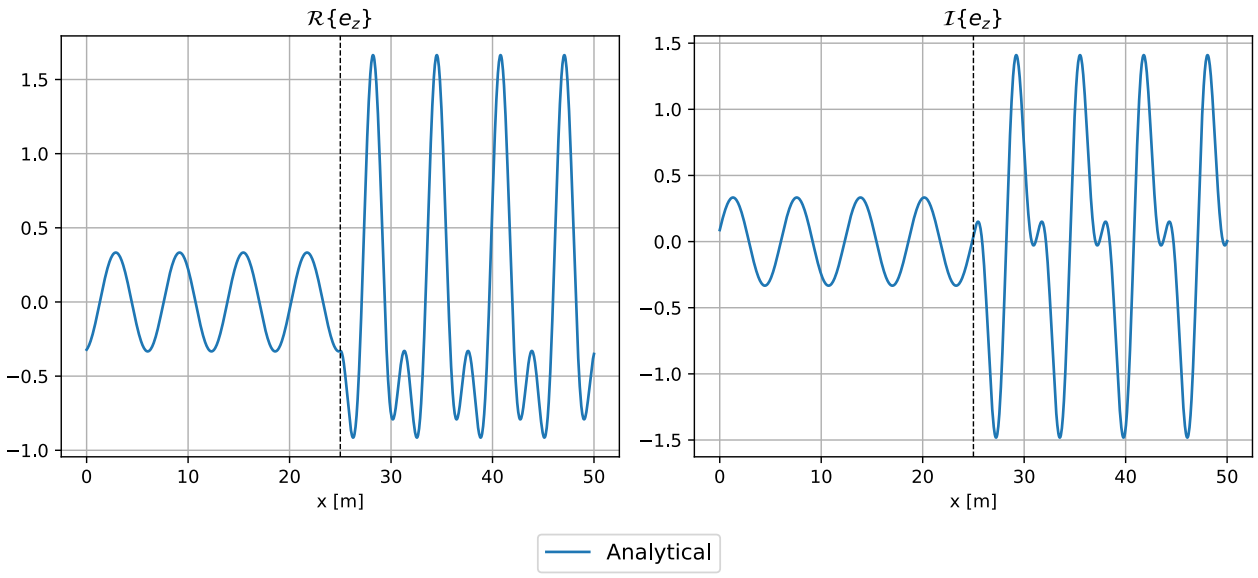


Figure 2.3: Real and imaginary component of the scattered field $E_z^{\text{scat}}(x)$ using the analytical solution of the one-dimensional scattering problem using $\varepsilon_r = 4$[-] and $\mu_r = 1$[-]

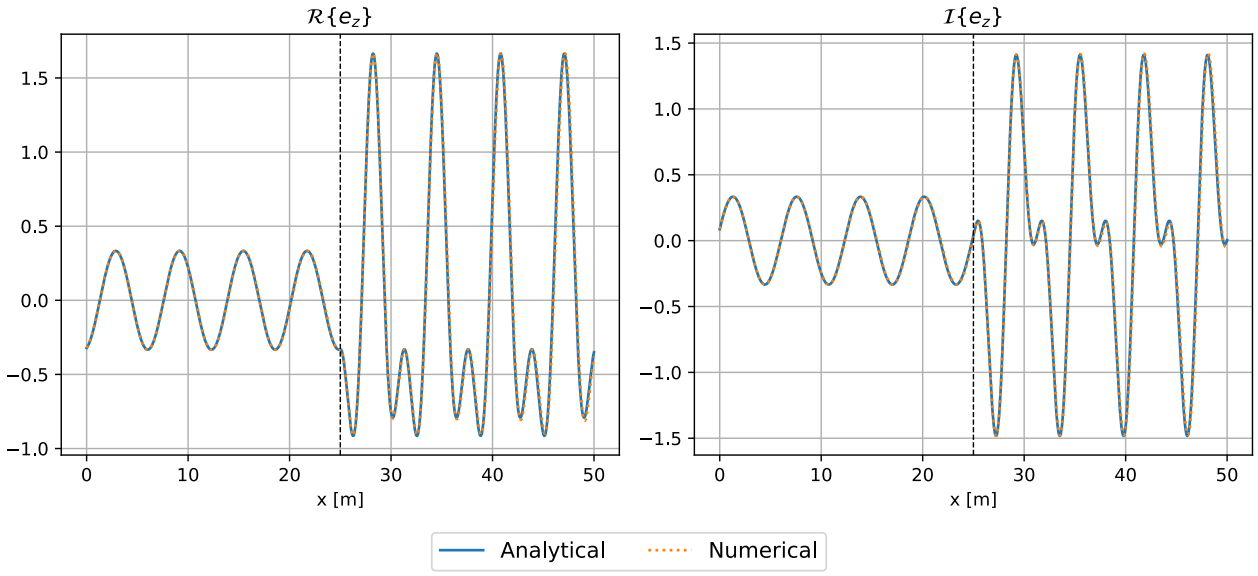


Figure 2.4: Real and imaginary component of the scattered field $E_z^{\text{scat}}(x)$ using the analytical and numerical solution of the one-dimensional scattering problem using $\varepsilon_r = 4$[-] and $\mu_r = 1$[-]

contaminate the solution. This section details the evolution of methods developed for this purpose, culminating in the *Perfectly Matched Layer* (PML) formulation employed in this thesis.

So far, our formulation has relied on a simple first-order *Absorbing Boundary Condition* (ABC), a Robin-type condition, to truncate the computational domain. As we have seen, a significant limitation of this approach is its inability to perfectly absorb waves composed of more than a single, predefined wavenumber. To overcome this and other shortcomings, we now introduce a more effective technique: the *Perfectly Matched Layer*.

The challenge of truncating an open domain without introducing spurious reflections is a long-standing problem in computational science, and it has been addressed by a wide variety of techniques. These include the use of *artificial boundary conditions*, the *Calderon operator*, *infinite element methods*, and the *overlapping Schwarz method of Hazard and Lenoir* [12]. Among these, a conceptually straightforward approach is to surround the computational domain with an artificial "sponge" layer designed to absorb outgoing waves.

However, a simple sponge layer presents a fundamental dilemma. If an artificial conducting medium is used to attenuate the wave, a high conductivity is needed for rapid absorption. This, in turn, creates a large impedance mismatch at the interface, leading to significant, non-physical reflections. Conversely, a low conductivity minimizes reflections but fails to absorb the wave effectively within a reasonably sized layer [12]. The *Perfectly Matched Layer*, introduced by Bérenger [13], resolves this conflict. It is an artificial absorbing medium meticulously designed to be, in the continuous case, reflectionless at its interface for waves of any frequency and any angle of incidence, thus providing a far more accurate and robust solution for domain truncation.

2.8.1 The Analytical Sommerfeld Radiation Condition

The physical behavior of an outgoing wave far from its source is described by the Sommerfeld radiation condition. For a two-dimensional problem, this analytical condition specifies that a wave field ψ must satisfy:

$$\lim_{\rho\to\infty} \sqrt{\rho}\left(\frac{\partial\psi}{\partial\rho} + ik\psi\right) = 0 \tag{2.58}$$

where $\rho = \sqrt{x^2+y^2}$ is the radial distance from the source and k is the wavenumber of the background medium as in [4]. This condition ensures that the wave is purely outgoing and that its amplitude decays appropriately with distance. As this condition is formulated at an infinite radius ($\rho \to \infty$), it cannot be implemented directly on a finite boundary. Consequently, numerical approximations are required.

2.8.2 Absorbing Boundary Conditions (ABCs)

A common approach to approximate the Sommerfeld condition is the use of *Absorbing Boundary Conditions* (ABCs). These are local, differential operators applied at the finite boundary Γ to damp incident waves. The first-order ABC assumes the wave impinges on the boundary at normal incidence, leading to the condition:

$$\frac{\partial\psi}{\partial n} + ik\psi = 0 \quad \text{on } \Gamma \tag{2.59}$$

where $\frac{\partial}{\partial n}$ is the derivative normal to the boundary.

It is noteworthy that from a mathematical classification standpoint, the first-order ABC (Equation (2.59)) is a specific type of **Robin boundary condition**. The general form of a Robin condition is a linear combination of the function value and its normal derivative on the boundary, typically expressed as $\alpha\psi + \beta\frac{\partial\psi}{\partial n} = g$. The first-order ABC directly corresponds to this form with coefficients $\alpha = ik$, $\beta = 1$,

and $g = 0$. This connection is significant, as numerical frameworks and mathematical literature often categorize boundary conditions as Dirichlet, Neumann, or Robin. Thus, the physical concept of a first-order absorbing boundary is implemented as a homogeneous Robin condition with a complex, frequency-dependent coefficient.

This condition provides good absorption for waves arriving at or near normal incidence which is why they are used for one dimensional cases. However, for waves arriving at oblique angles, an impedance mismatch occurs, leading to significant reflections. While higher-order ABCs have been developed to improve absorption for a wider range of incident angles (see Section 9.1.2 for high-order curved boundaries of [4]), they introduce greater mathematical complexity and do not achieve reflectionless absorption for all angles. These residual reflections motivated the development of alternative methods.

2.8.3 The Perfectly Matched Layer (PML)

The *Perfectly Matched Layer* offers a different methodology for domain truncation as presented in Section 9.6 of [4]. Rather than defining a condition at the boundary, the PML is an auxiliary domain, a layer of artificial material with engineered lossy properties, that surrounds the primary physical domain, as illustrated in Figure 2.5.

The design of the PML is based on three key principles:

- **Impedance-Matched Interface**: The interface between the physical domain and the PML is designed to be non-reflecting. In the ideal continuous case, a plane wave entering the PML from the physical domain does so without reflection, irrespective of its frequency or angle of incidence.
- **Wave Attenuation**: Within the PML, the wave amplitude is progressively attenuated. The layer is constructed with sufficient thickness and loss such that the wave amplitude decays to a negligible value before reaching the outer boundary of the PML.
- **Simple Outer Termination**: Because the wave is effectively absorbed within the layer, the outermost boundary of the PML can be terminated with a simple boundary condition, such as a homogeneous Dirichlet condition ($\psi = 0$), without introducing significant spurious reflections back into the physical domain.

The mathematical realization of the PML is achieved through **complex coordinate stretching**. This technique modifies the spatial derivatives in the governing wave equation within the PML region. For instance, a derivative with respect to a Cartesian coordinate x is transformed as:

$$\frac{\partial}{\partial x} \quad \rightarrow \quad \frac{1}{s_x(x)} \frac{\partial}{\partial x}, \quad \text{where } s_x(x) = 1 - i\frac{\sigma_x(x)}{\omega} \tag{2.60}$$

Here, ω is the angular frequency of the wave. The function $\sigma_x(x)$ is an artificial conductivity profile that is zero within the physical domain (where $s_x = 1$ and the equation is unchanged) and increases within the PML. The complex value of the stretching factor s_x introduces the desired wave attenuation.

In effect, the PML formulation replaces the challenge of implementing a complex radiation boundary condition with the task of modeling propagation through an artificial, lossy medium. Due to its effective absorption properties across a wide range of incidence angles and frequencies, the PML formulation is adopted for the numerical models in this work.

2.9 PML for the One Dimensional Homogeneous Case

Since we consider a homogeneous domain, the total field approach is used and we can write the weak form straightaway as the weak form assuming a Robin boundary condition as been derived in

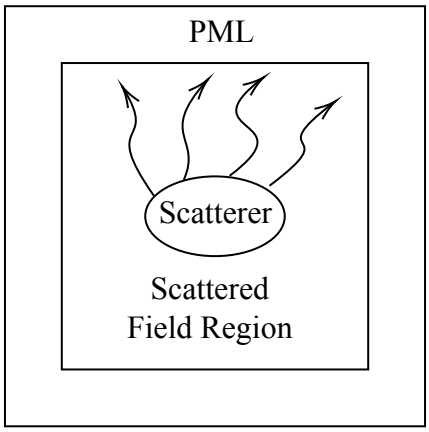


Figure 2.5: Addition of the artificial domain needed arising from the usage of PML.

Equation (2.61):

Find $E_{\text{tot}} \in H^1([0, L])$ such that $E_z^{\text{tot}}(0) = E_0$:

$$\int_0^L \left(\frac{1}{\tilde{\mu}} \frac{dE_z^{\text{tot}}}{dx} \frac{d\overline{f}}{dx} - \omega^2 \tilde{\varepsilon} E_z^{\text{tot}} \overline{f} \right) dx = 0. \tag{2.61}$$

for all test functions $f \in H^1([0, L])$ such that $f(0) = 0$ and where $\tilde{\mu} = \mu s_x$ and $\tilde{\varepsilon} = \varepsilon s_x$ are the *modified* permeability and permittivity to incorporate the PMLs.

As a result of using the PML truncation approach, the boundary term incorporating the Robin boudnary condition is removed and replaced by the "artifically lossy" material properties within the PML region and a homogeneous Dirichlet condition on the outer boundary as shown in Figure 2.6 and the *Finite Element solution* is shown in Figure 2.7.

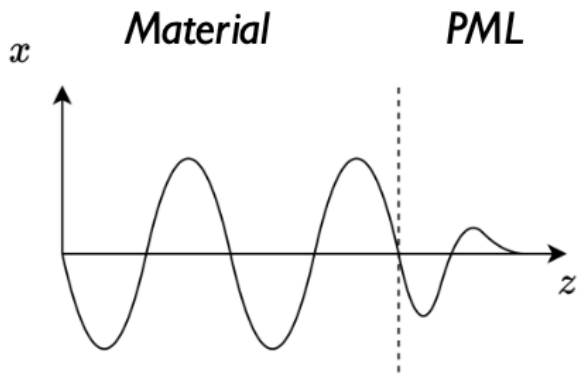


Figure 2.6: PML effect on the right part of the physical domain.

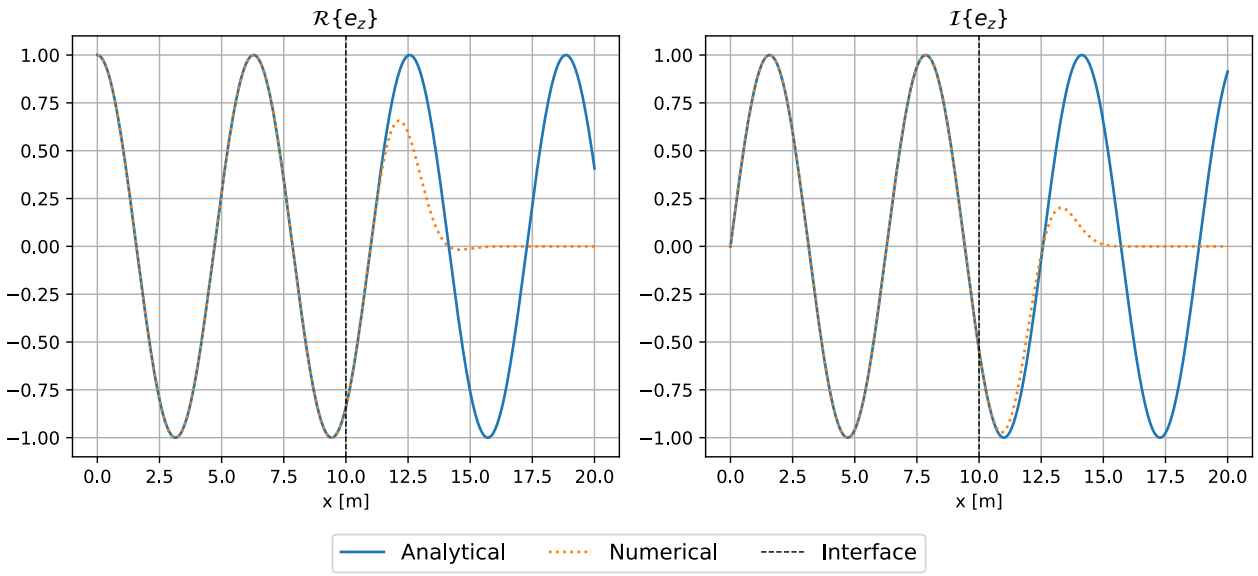


Figure 2.7: Analytical and numerical solution of the one dimensional homogeneous case with PML region starting at $x = 10$[-]. ($\varepsilon = 1$ and $\mu = 1$)

2.10 PML for the 1D GAAFET-equivalent case

The final one-dimensional model considered is a 1D representation of a layered nanoelectronic device, such as a *Gate-All-Around Field-Effect Transistor*. This multilayer stack, a common but challenging form of heterogeneity, consists of a superstrate, a central scatterer (e.g., the channel), and a substrate. The entire physical domain is terminated on both ends by PMLs to absorb outgoing waves, as depicted in Figure 2.8.

The key difference from previous cases lies in the material properties, which are now piecewise-constant functions corresponding to the distinct layers. If the physical domain spans from x_1 to x_4, with interfaces at x_2 and x_3, the permittivity profile $\varepsilon_r(x)$ is defined as:

$$\varepsilon_r(x) = \begin{cases} \varepsilon_{\text{super}} & \text{for } x \in [x_1, x_2) \\ \varepsilon_{\text{scat}} & \text{for } x \in [x_2, x_3) \\ \varepsilon_{\text{sub}} & \text{for } x \in [x_3, x_4] \end{cases} \tag{2.62}$$

A similar definition applies to the permeability $\mu_r(x)$.

2.10.1 A Localized Source Formulation via a Secondary Scattered Field

For a z-polarized transverse electric (TE) wave, the problem is governed by the scalar Helmholtz equation:

$$L_{\varepsilon,\mu}(E_z) := \frac{d}{dx}\left(\frac{1}{\mu_r(x)}\frac{dE_z}{dx}\right) + k_0^2 \varepsilon_r(x) E_z = 0 \tag{2.63}$$

However, for a layered background like this, that approach is problematic. The resulting source term for the scattered field E_z^{sc} is distributed across all layer interfaces, which can create source terms within the PMLs themselves, compromising their ability to absorb waves cleanly.

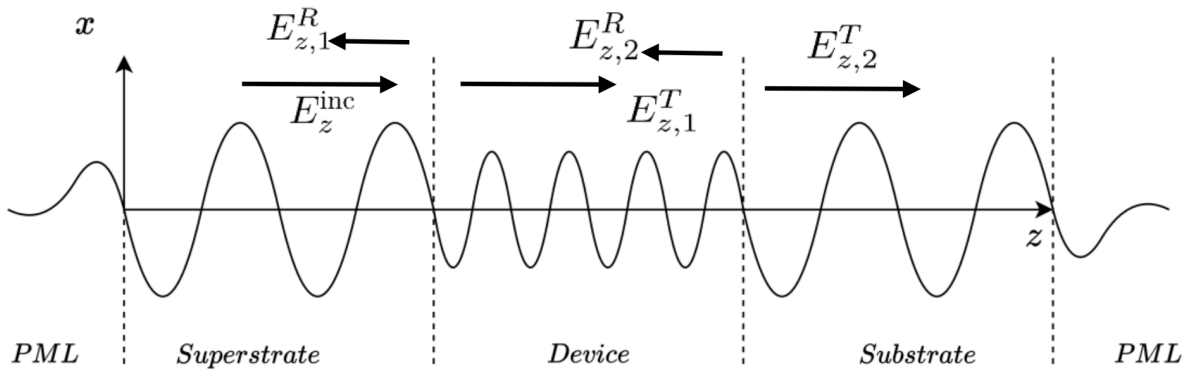


Figure 2.8: Conceptual diagram of the 1D GAAFET-equivalent model. The computational domain Ω consists of a central physical domain Ω_{phys} comprising the superstrate, scatterer, and substrate layers, terminated by PMLs on both sides.

To overcome this, we adopt a more sophisticated formulation, as detailed in [14]. The core idea is to introduce a different decomposition of the total field:

$$E_z = E_z^1 + E_z^{\text{sc},2} \tag{2.64}$$

Here, E_z^1 is the known solution to a simplified *annex* problem, while $E_z^{\text{sc},2}$ is the new unknown, which we term the secondary scattered field.

The annex problem is a reduced version of the full geometry. For our GAAFET model, it consists of the superstrate-substrate system without the central scatterer. This background is described by material properties $\varepsilon_{r,1}(x)$ and $\mu_{r,1}(x)$. The annex field E_z^1 is the solution within this simplified structure, satisfying $L_{\varepsilon_1,\mu_1}(E_z^1) = 0$, and can be determined analytically using Fresnel equations. Specifically, for an incident plane wave of amplitude M traveling from the superstrate (medium 'super') into the substrate (medium 'sub') at an interface located at $x = L_{\text{int}}$, the annex field E_z^1 is given by:

$$E_z^1(x) = \begin{cases} Ae^{ik_{\text{super}}x} + AR_{\text{annex}}e^{-ik_{\text{super}}x}, & \text{for } x < L_{\text{int}} \\ AT_{\text{annex}}e^{i(k_{\text{super}}-k_{\text{sub}})L_{\text{int}}}e^{ik_{\text{sub}}x}, & \text{for } x \geq L_{\text{int}} \end{cases} \tag{2.65}$$

where A is the amplitude of the incoming wave, $k = \omega\sqrt{\varepsilon\mu}$ is the wavenumber and the TE-wave reflection and transmission coefficients are defined by the medium impedances $Z = \sqrt{\mu/\varepsilon}$:

$$R_{\text{annex}} = \frac{Z_{\text{super}} - Z_{\text{sub}}}{Z_{\text{super}} + Z_{\text{sub}}}, \qquad T_{\text{annex}} = \frac{2Z_{\text{super}}}{Z_{\text{super}} + Z_{\text{sub}}}. \tag{2.66}$$

A critical property of this decomposition is that since both E_z and E_z^1 satisfy an outgoing wave condition, their difference $E_z^{\text{sc},2}$ inherently does as well, ensuring its compatibility with PMLs.

The primary advantage of this formulation is that it transforms the *original scattering problem* into an *equivalent radiation problem* with a highly localized source. Substituting the decomposition into the original Helmholtz equation yields a PDE for the new unknown, $E_z^{\text{sc},2}$:

$$L_{\varepsilon,\mu}(E_z^{\text{sc},2}) = -L_{\varepsilon,\mu}(E_z^1) \tag{2.67}$$

The right-hand side, $S(x) = -L_{\varepsilon,\mu}(E_z^1)$, acts as a known source term. We can reveal its compact support by adding $L_{\varepsilon_1,\mu_1}(E_z^1)$ (which is identically zero):

$$\begin{aligned} S(x) &= -\left(L_{\varepsilon,\mu}(E_z^1) - L_{\varepsilon_1,\mu_1}(E_z^1)\right) \\ &= -\left[\frac{d}{dx}\left(\left(\frac{1}{\mu_r} - \frac{1}{\mu_{r,1}}\right)\frac{dE_z^1}{dx}\right) + k_0^2(\varepsilon_r - \varepsilon_{r,1})E_z^1\right] \end{aligned} \tag{2.68}$$

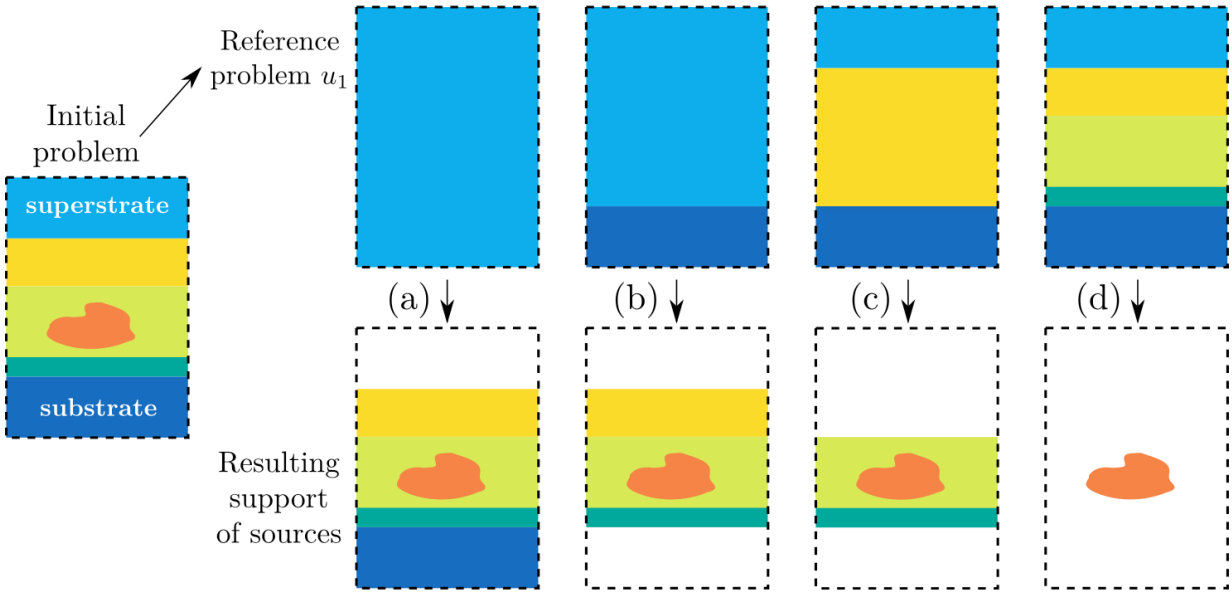


Figure 2.9: Illustration of the possible annex problem extracted from [14].

The material difference terms, $(\varepsilon_r - \varepsilon_{r,1})$ and $(1/\mu_r - 1/\mu_{r,1})$, are non-zero only within the domain of the scatterer. This proves that the source term $S(x)$ is confined to a compact region.

The judicious choice of the annex problem is key to this method efficiency, as it directly determines the extent of the source support (Figure 2.9). This localization is the ultimate benefit of the method: it transforms the original problem into an equivalent radiation problem where the FEM solver computes a field generated by a source strictly confined to the device active region. This prevents any artificial excitation within the PMLs, which is essential for numerical stability and accuracy.

2.10.2 Derivation of the Weak Formulation

We begin with the strong form for the second scattered field, as stated in Eq. (2.67):

$$L_{\varepsilon,\mu}(E_z^{\mathrm{sc},2}) = S(x) \tag{2.69}$$

where $S(x)$ is the known source term with compact support, defined in Eq. (2.68).

We multiply the entire equation by a suitable test function, which we denote as the complex conjugate $\overline{f}(x)$, and integrate over the entire computational domain Ω (physical domain + PMLs). The test function f is chosen from a function space that satisfies the same homogeneous boundary conditions as the solution, which in this case is the Sobolev space $H^1(\Omega)$ with $f\big|_{\partial\Omega} = 0$.

$$\int_\Omega \left[\frac{d}{dx}\left(\frac{1}{\mu_r(x)}\frac{dE_z^{\mathrm{sc},2}}{dx}\right) + k_0^2\varepsilon_r(x)E_z^{\mathrm{sc},2}\right]\overline{f}dx = \int_\Omega S(x)\overline{f}dx \tag{2.70}$$

Next, integration by parts is applied to the second-order derivative term:

$$\int_\Omega \frac{d}{dx}\left(\frac{1}{\mu_r}\frac{dE_z^{\mathrm{sc},2}}{dx}\right)\overline{f}dx = \left[\frac{1}{\mu_r}\frac{dE_z^{\mathrm{sc},2}}{dx}\overline{f}\right]_{\partial\Omega} - \int_\Omega \frac{1}{\mu_r}\frac{dE_z^{\mathrm{sc},2}}{dx}\frac{d\overline{f}}{dx}dx \tag{2.71}$$

The first term on the right-hand side is the boundary term, evaluated at the outer edges of the computational domain, $\partial\Omega$. Since the test function $\overline{f}$ is zero on the boundary $\partial\Omega$ because of the Dirichlet

condition, the boundary term vanishes identically:

$$\left[\frac{1}{\mu_r}\frac{dE_z^{\mathrm{sc},2}}{dx}\overline{f}\right]_{\partial\Omega} = 0 \tag{2.72}$$

Substituting the result from Step 2 back into Eq. (2.80), we obtain:

$$-\int_\Omega \frac{1}{\mu_r}\frac{dE_z^{\mathrm{sc},2}}{dx}\frac{d\overline{f}}{dx}dx + \int_\Omega k_0^2\varepsilon_r E_z^{\mathrm{sc},2}\overline{f}dx = \int_\Omega S(x)\overline{f}dx \tag{2.73}$$

By convention, we can multiply by -1 and rearrange the terms to match the standard form $a(u,v) = l(v)$.

The final weak formulation is stated as: Find $E_z^{\mathrm{sc},2} \in H^1(\Omega)$ with $E_z^{\mathrm{sc},2}\big|_{\partial\Omega} = 0$ and that for all test functions $f \in H^1(\Omega)$ such that $f\big|_{\partial\Omega} = 0$:

$$a(E_z^{\mathrm{sc},2}, f) = l(f) \tag{2.74}$$

where the sesquilinear form, $a(\cdot,\cdot)$, is:

$$a(E_z^{\mathrm{sc},2}, f) := \int_\Omega \left(\frac{1}{\mu_r(x)}\frac{dE_z^{\mathrm{sc},2}}{dx}\frac{d\overline{f}}{dx} - k_0^2\varepsilon_r(x)E_z^{\mathrm{sc},2}\overline{f}\right)dx \tag{2.75}$$

and the antilinear form, $l(\cdot)$, which contains the source terms, is:

$$l(f) := -\int_\Omega S(x)\overline{f}, dx \tag{2.76}$$

The comparison between the final numerical solution for the total field, $E_z = E_z^{\mathrm{sc},2} + E_z^1$, and the analytical solution is presented in Figure 2.10, Figure 2.11 and Figure 2.12 where the secondary scattered field, the scattered field and the total field are illustrated.

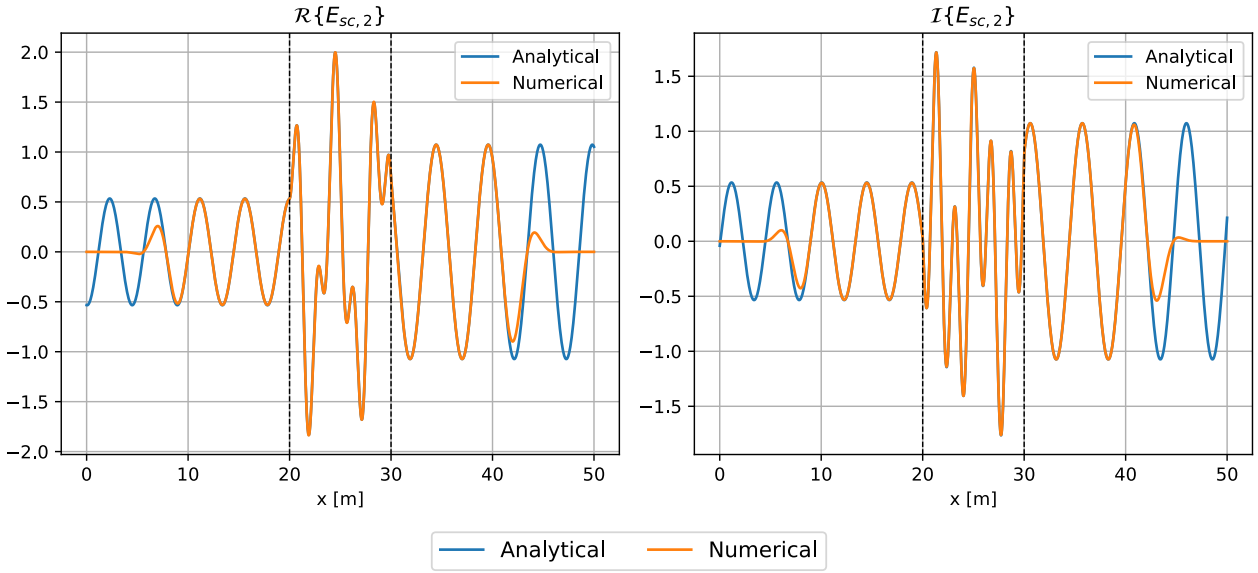


Figure 2.10: Comparison between the analytical and numerical secondary scattered field $E_z^{\mathrm{sc},2}$. Material properties are $\varepsilon_r = (1,3,3)$ and $\mu_r = (2,4,1.5)$ for the superstrate, scatterer, and substrate, respectively.

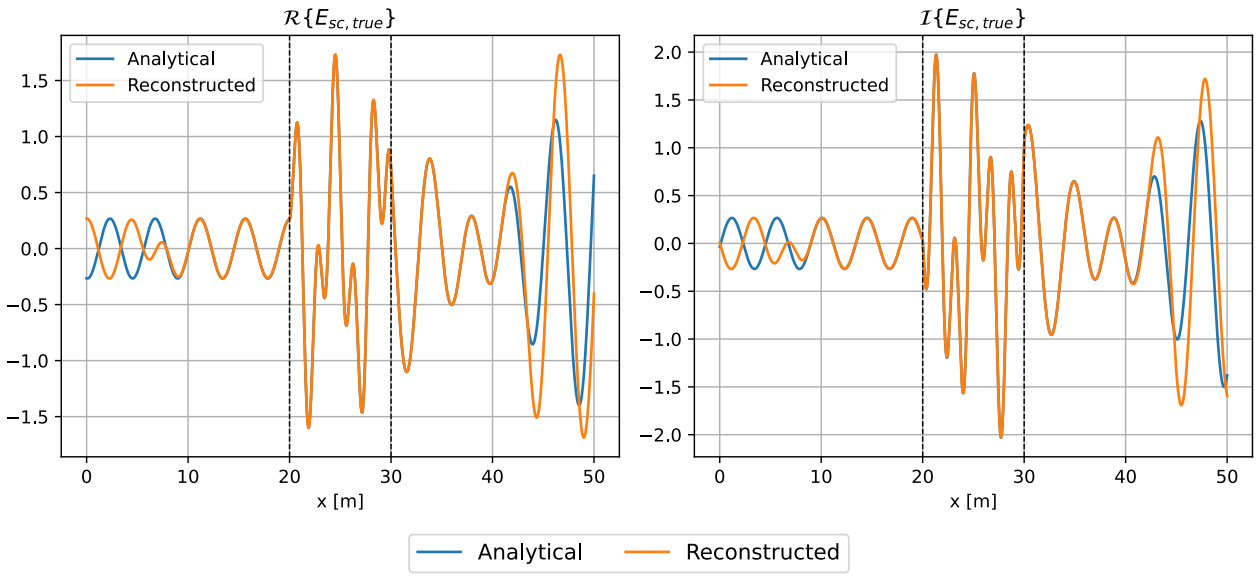


Figure 2.11: Comparison between the analytical and numerical scattered field E_z^{sc}. Material properties are $\varepsilon_r = (1, 3, 3)$ and $\mu_r = (2, 4, 1.5)$ for the superstrate, scatterer, and substrate, respectively.

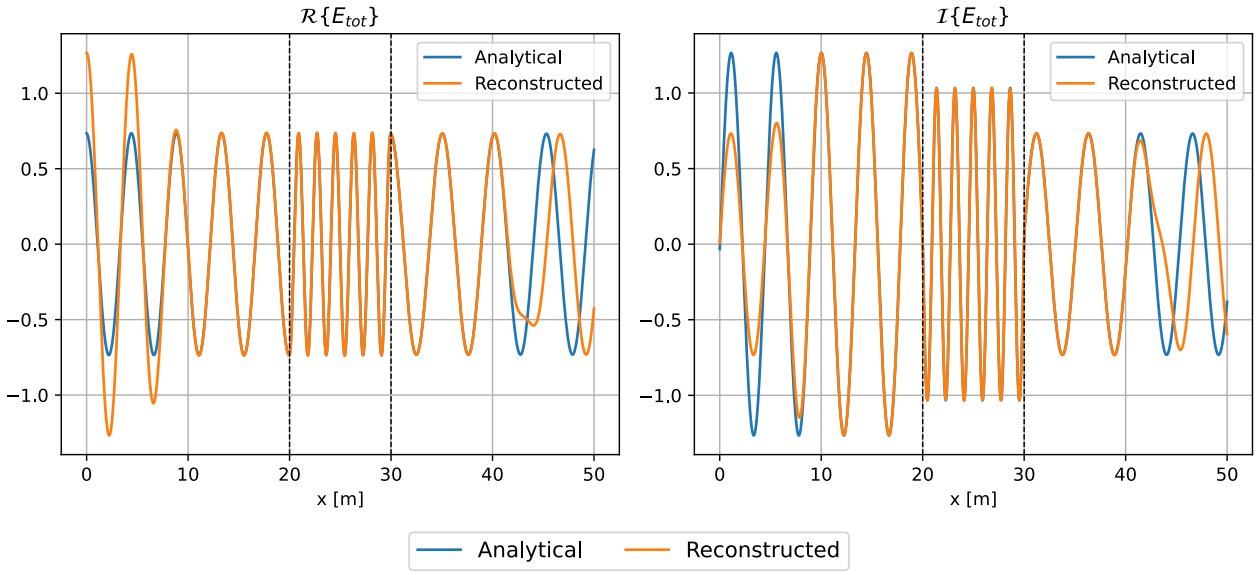


Figure 2.12: Comparison between the analytical and numerical total field E_z^{tot}. Material properties are $\varepsilon_r = (1, 3, 3)$ and $\mu_r = (2, 4, 1.5)$ for the superstrate, scatterer, and substrate, respectively.

2.11 Formulation for TM_z Scattering with PML Boundaries

This section provides a detailed derivation of the variational formulation for the 2D Transverse Magnetic (TM) scattering problem, where the magnetic field is $\mathbf{H} = H_z(x, y)\hat{\mathbf{z}}$, within a domain terminated by PMLs. Moreover, we assume a constant permittivity ε in order to simplify the following derivation.

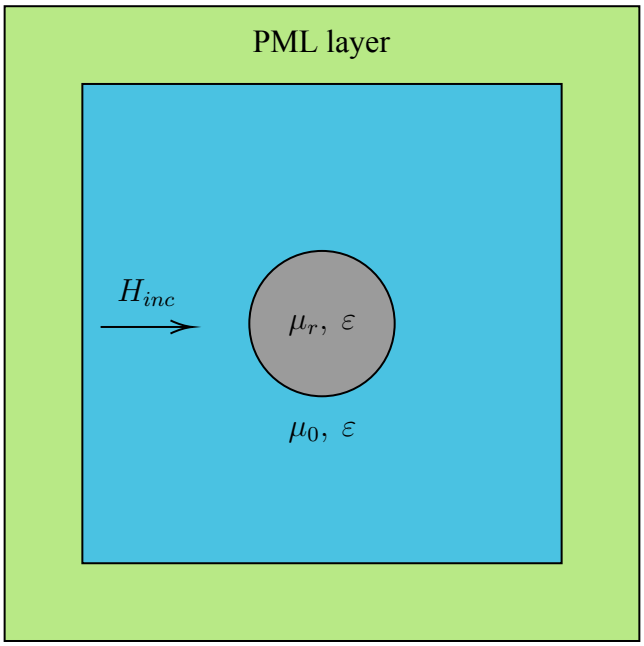


Figure 2.13: Illustration of the Mie test case with the different material properties.

The Strong Formulation

For a problem with a *homogeneous permittivity* ε and *permeability* μ, the standard Helmholtz operator is modified within the PML regions. This results in the following anisotropic wave equation for the total field H_z^{tot} in the full computational domain Ω (physical + PML), similarly to [4]:

$$\frac{1}{\varepsilon}\left[\frac{1}{s_x}\frac{\partial}{\partial x}\left(\frac{1}{s_x}\frac{\partial H_z^{\text{tot}}}{\partial x}\right)+\frac{1}{s_y}\frac{\partial}{\partial y}\left(\frac{1}{s_y}\frac{\partial H_z^{\text{tot}}}{\partial y}\right)\right]+\omega^2\mu H_z^{\text{tot}}=0 \tag{2.77}$$

The entire domain Ω is terminated by an outer boundary Γ_{PML}, on which a homogeneous Dirichlet condition ($H_z^{\text{tot}}=0$) is enforced.

Applying the scattered field decomposition, $H_z^{\text{tot}}=H_z^{\text{inc}}+H_z^{\text{scat}}$, and leveraging the linearity of the operators, we can formulate the problem in terms of the unknown scattered field H_z^{scat}. This results in the following strong form:

$$\frac{1}{s_x}\frac{\partial}{\partial x}\left(\frac{1}{s_x}\frac{\partial H_z^{\text{scat}}}{\partial x}\right)+\frac{1}{s_y}\frac{\partial}{\partial y}\left(\frac{1}{s_y}\frac{\partial H_z^{\text{scat}}}{\partial y}\right)+k^2H_z^{\text{scat}}=f_s \tag{2.78}$$

where the source term f_s contains all terms related to the known incident field H_z^{inc}:

$$f_s=-\left[\frac{1}{s_x}\frac{\partial}{\partial x}\left(\frac{1}{s_x}\frac{\partial H_z^{\text{inc}}}{\partial x}\right)+\frac{1}{s_y}\frac{\partial}{\partial y}\left(\frac{1}{s_y}\frac{\partial H_z^{\text{inc}}}{\partial y}\right)\right]+k^2H_z^{\text{inc}} \tag{2.79}$$

Derivation of the Variational Formulation

We begin with the strong form (Equation (2.78)), multiply by a complex-conjugated test function $\overline{f}\in H_0^1(\Omega)$, and integrate over the domain Ω. The space $H_0^1(\Omega)$ consists of functions that are zero on the Dirichlet boundary Γ_{PML}.

$$\int_\Omega\left[\frac{1}{s_x}\frac{\partial}{\partial x}\left(\frac{1}{s_x}\frac{\partial H_z^{\text{scat}}}{\partial x}\right)+\frac{1}{s_y}\frac{\partial}{\partial y}\left(\frac{1}{s_y}\frac{\partial H_z^{\text{scat}}}{\partial y}\right)+k^2H_z^{\text{scat}}\right]\overline{f}\,d\Omega=\int_\Omega f_s\overline{f}\,d\Omega \tag{2.80}$$

We apply Green first identity (integration by parts) to the second-order derivative terms to move the derivatives onto the test function. For the first term:

$$\int_\Omega \left[\frac{1}{s_x}\frac{\partial}{\partial x}\left(\frac{1}{s_x}\frac{\partial H_z^{\text{scat}}}{\partial x}\right)\right]\overline{f}\,d\Omega = \oint_{\Gamma_{\text{PML}}} \frac{\overline{f}}{s_x}\left(\frac{1}{s_x}\frac{\partial H_z^{\text{scat}}}{\partial x}\right) n_x\, d\Gamma - \int_\Omega \frac{1}{s_x^2}\frac{\partial H_z^{\text{scat}}}{\partial x}\frac{\partial \overline{f}}{\partial x} d\Omega \tag{2.81}$$

The boundary integral vanishes because the test function f is zero on the boundary Γ_{PML} (i.e., $f\big|_{\Gamma_{\text{PML}}} = 0$). Applying the same process to the y-derivative term and rearranging the equation, we arrive at the standard variational form: Find $H_z^{\text{scat}} \in H_0^1(\Omega)$ such that:

$$a(H_z^{\text{scat}}, f) = b(f) \quad \forall f \in H_0^1(\Omega) \tag{2.82}$$

where the sesquilinear form $a(\cdot,\cdot)$ is defined as:

$$a(H_z^{\text{scat}}, f) := \int_\Omega \left(\frac{1}{s_x^2}\frac{\partial H_z^{\text{scat}}}{\partial x}\frac{\partial \overline{f}}{\partial x} + \frac{1}{s_y^2}\frac{\partial H_z^{\text{scat}}}{\partial y}\frac{\partial \overline{f}}{\partial y} - k^2 H_z^{\text{scat}}\overline{f}\right) d\Omega \tag{2.83}$$

and the antilinear form $b(f)$ on the right-hand side is given by:

$$b(f) := \int_\Omega f_s \overline{f}\, d\Omega \tag{2.84}$$

2.11.1 Simplification of the Source Term

Instead of directly evaluating the integral involving f_s, we can derive a more elegant and computationally efficient form for $b(f)$. This method subtracts the weak form of the incident field equation from that of the total field equation.

Weak Form of the Total Field

The total field H_z^{tot} satisfies Eq. (2.77). Its weak form is:

$$a(H_z^{\text{tot}}, f; \varepsilon, \mu) = a(H_z^{\text{inc}} + H_z^{\text{scat}}, f; \varepsilon, \mu) = 0 \tag{2.85}$$

Weak Form of the Incident Field

The incident field H_z^{inc} is a solution in the background medium, characterized by properties ε_b and μ_b. Its governing equation is:

$$\frac{1}{s_x}\frac{\partial}{\partial x}\left(\frac{1}{s_x}\frac{\partial H_z^{\text{inc}}}{\partial x}\right) + \frac{1}{s_y}\frac{\partial}{\partial y}\left(\frac{1}{s_y}\frac{\partial H_z^{\text{inc}}}{\partial y}\right) + k_b^2 H_z^{\text{inc}} = 0 \tag{2.86}$$

The weak form of this equation is identically zero:

$$a(H_z^{\text{inc}}, f; \varepsilon_b, \mu_b) = 0 \tag{2.87}$$

Derivation of the Scattered Field System

Using the linearity of the sesquilinear form, we expand Eq. (2.85):

$$a(H_z^{\text{scat}}, f; \varepsilon, \mu) + a(H_z^{\text{inc}}, f; \varepsilon, \mu) = 0 \implies a(H_z^{\text{scat}}, f; \varepsilon, \mu) = -a(H_z^{\text{inc}}, f; \varepsilon, \mu) \tag{2.88}$$

Since $a(H_z^{\text{inc}}, f; \varepsilon_b, \mu_b) = 0$, we can subtract it from the right-hand side:

$$\begin{aligned} a(H_z^{\text{scat}}, f; \varepsilon, \mu) &= -a(H_z^{\text{inc}}, f; \varepsilon, \mu) - \underbrace{a(H_z^{\text{inc}}, f; \varepsilon_b, \mu_b)}_{} = 0 \\ &= a(H_z^{\text{inc}}, f; \varepsilon_b, \mu_b) - a(H_z^{\text{inc}}, f; \varepsilon, \mu) \end{aligned} \tag{2.89}$$

This subtraction isolates the material and PML contrast terms, yielding the simplified anitlinear form $b(f)$:

$$b(f) = \int_\Omega -(k_b^2 - k^2) H_z^{\text{inc}} \overline{f} \, d\Omega \tag{2.90}$$

$$= \int_\Omega -\varepsilon \omega^2 (\mu_b^2 - \mu^2) H_z^{\text{inc}} \overline{f} \, d\Omega \tag{2.91}$$

Final Variational Problem

The final variational problem is to find $H_z^{\text{scat}} \in H^1(\Omega)$ with $H_z\big|_{\Gamma_{\text{PML}}} = 0$ such that:

$$a(H_z^{\text{scat}}, f) = b(f) \quad \forall f \in H_1(\Omega) \text{ with } f\big|_{\Gamma_{\text{PML}}} = 0 \tag{2.92}$$

where the bilinear form is

$$a(H_z^{\text{scat}}, f) = \int_\Omega \left(\frac{1}{s_x^2} \frac{\partial H_z^{\text{scat}}}{\partial x} \frac{\partial \overline{f}}{\partial x} + \frac{1}{s_y^2} \frac{\partial H_z^{\text{scat}}}{\partial y} \frac{\partial \overline{f}}{\partial y} - k^2 H_z^{\text{scat}} \overline{f} \right) d\Omega \tag{2.93}$$

and the linear form $b(f)$ is given by Eq. (2.94).

The key advantage of this formulation is that the integrand of $b(f)$ is non-zero only in regions where the local properties (ε, μ) differ from the background. If we denote the scatterer domain as Ω_{scat}, the integration for the source term reduces to:

$$b(f) = \int_{\Omega_{\text{scat}}} -\varepsilon \omega^2 (\mu_b - \mu) H_z^{\text{inc}} \overline{f} \, d\Omega \tag{2.94}$$

This simplification provides clearer physical insight by explicitly showing that the scattered field is generated by the material contrast between the object and the background medium.

2.12 Semi-analytical solution to the Mie problem

The problem of a transverse-magnetic (TM) plane wave scattering from an infinite, homogeneous dielectric cylinder is a canonical benchmark in computational electromagnetics. The solution can be expressed semi-analytically as an infinite series of cylindrical wave functions. This section details the formulation implemented in the validation script based on the solution in [15], which is used to verify the accuracy of the *Finite Element Method*solver.

Let the cylinder have a radius R and a relative permittivity ε_r, situated in a background medium with properties (ε_b, μ_b). The refractive index of the cylinder is $n = \sqrt{\varepsilon_r \mu_r}$. The wavenumbers in the background and inside the cylinder are k_b and $k_1 = nk_b$, respectively. The incident magnetic field, H_z^{inc}, is a plane wave propagating along the x-axis.

2.12.1 Field Representations

The total field is decomposed into incident, scattered, and internal (transmitted) components. Each field is expanded as a series of cylindrical harmonics in polar coordinates (r, θ). The series is truncated for computation at a finite mode number M.

- **Incident Field** ($r \geq 0$): The incident plane wave $H_z^{\text{inc}} = e^{-ik_b r\cos\theta}$ is expanded using the Jacobi-Anger identity with Bessel functions of the first kind, J_m:

$$H_z^{\text{inc}}(r,\theta) = \sum_{m=-M}^{M} i^m J_m(k_b r) e^{im\theta} \tag{2.95}$$

- **Scattered Field** ($r \geq R$): The scattered field must satisfy the Sommerfeld radiation condition, representing an outgoing wave. This is achieved using Hankel functions of the first kind, $H_m^{(1)}$, with unknown scattering coefficients a_m:

$$H_z^{\text{scat}}(r,\theta) = \sum_{m=-M}^{M} i^m a_m H_m^{(1)}(k_b r) e^{im\theta} \tag{2.96}$$

- **Internal Field** ($r \leq R$): The field inside the cylinder must be finite at the origin ($r = 0$). This requires the use of Bessel functions, J_m, with unknown transmission coefficients b_m:

$$H_z^{\text{int}}(r,\theta) = \sum_{m=-M}^{M} i^m b_m J_m(k_1 r) e^{im\theta} \tag{2.97}$$

The total field outside the cylinder is the sum of the incident and scattered fields: $H_z^{\text{tot}} = H_z^{\text{inc}} + H_z^{\text{scat}}$.

2.12.2 Scattering and Transmission Coefficients

The unknown coefficients a_m and b_m are found by enforcing the continuity of the tangential magnetic field (H_z) and the tangential electric field (E_ϕ^t) at the cylinder interface, $r = R$.

The solution for these coefficients is computed by first calculating an intermediate term, γ_h, which simplifies the expressions.

The scattering coefficient a_m is calculated as:

$$a_m = -\frac{J_m(k_b R) - \gamma_h J_m'(k_b R)}{H_m^{(2)}(k_b R) - \gamma_h H_m^{(2)'}(k_b R)} \tag{2.98}$$

where the intermediate term γ_h is given by:

$$\gamma_h = \frac{Z_b}{Z_1}\frac{J_m(k_1 R)}{J_m'(k_1 R)} \tag{2.99}$$

Here, J_m is the Bessel function of the first kind, $H_m^{(2)}$ is the Hankel function of the second kind, and the prime denotes the derivative with respect to the function argument (e.g., $J_m'(z) = dJ_m(z)/dz$).

The transmission coefficient b_m is not calculated from an independent explicit formula but is found sequentially from the scattering coefficient a_m by enforcing the continuity of H_z at $r = R$:

$$H_z^{\text{inc}}(R) + H_z^{\text{scat}}(R) = H_z^{\text{int}}(R) \quad \Longrightarrow \quad J_m(k_b R) + a_m H_m^{(2)}(k_b R) = b_m J_m(k_1 R) \tag{2.100}$$

Solving for b_m yields the expression:

$$b_m = \frac{J_m(k_b R) + a_m H_m^{(2)}(k_b R)}{J_m(k_1 R)} \tag{2.101}$$

2.12.3 Final Solution Representation

The Python code calculates the total field directly, which is the quantity to be compared with the FEM solution.

- **Total Field Outside Cylinder** ($r \geq R$):

$$H_z^{\text{tot}}(r,\theta) = \sum_{m=-M}^{M} i^m \left(J_m(k_b r) + a_m H_m^{(1)}(k_b r) \right) e^{im\theta} \tag{2.102}$$

- **Total Field Inside Cylinder** ($r \leq R$):

$$H_z^{\text{tot}}(r,\theta) = H_z^{\text{int}}(r,\theta) = \sum_{m=-M}^{M} i^m b_m J_m(k_1 r) e^{im\theta} \tag{2.103}$$

These equations provide a precise value for the magnetic field at any point in space, serving as the ground truth for assessing the accuracy of the numerical simulation.

2.13 Solution of the Mie problem

These equations provide a precise value for the magnetic field at any point in space, serving as the ground truth for assessing the accuracy of the numerical simulation.

The physical setup consists of an infinitely long, homogeneous cylinder with a circular cross-section of radius $r = 1.0[m]$. This cylindrical inclusion is characterized by a relative magnetic permeability of $\mu_r = 4.0[-]$ and the relative electric permittivity of the background. The cylinder is embedded in a background medium representing free space ($\mu_0 = 1.0[\frac{\text{T·m}}{\text{A}}]$, $\varepsilon_0 = 1.0[\frac{\text{C}}{\text{V·m}}]$) and is illuminated by a time-harmonic wave with an angular frequency of $\omega = 1.0$[rad/s].

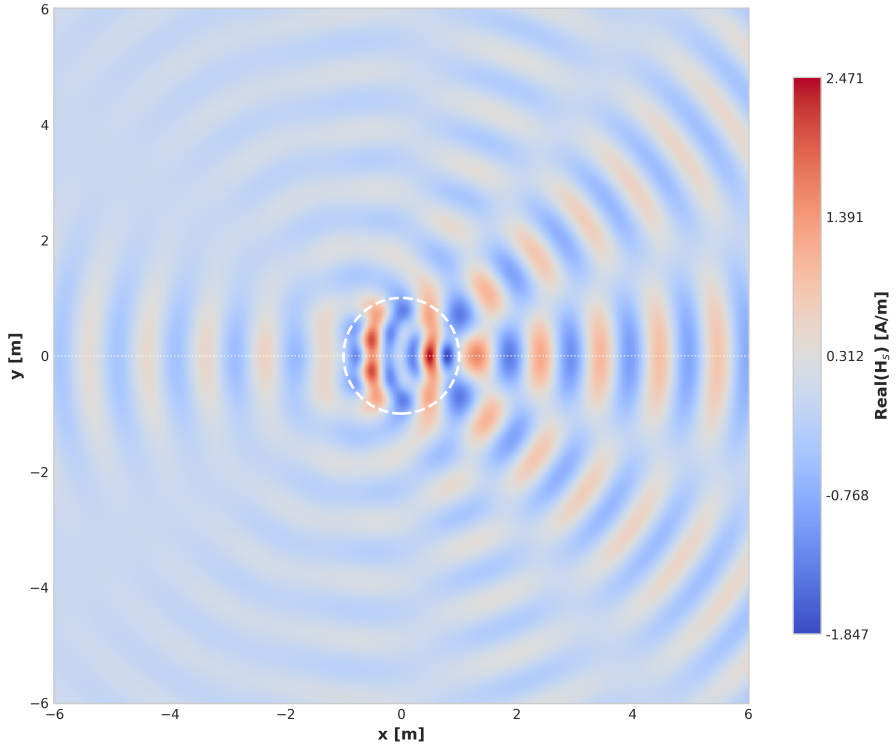


Figure 2.14: Real part of the semi-analytical scattered solution of the Mie test case using $\mu_0 = 1$, $\varepsilon_0 = 1$, $\mu_r = 4$ and $\varepsilon_r = 1$.

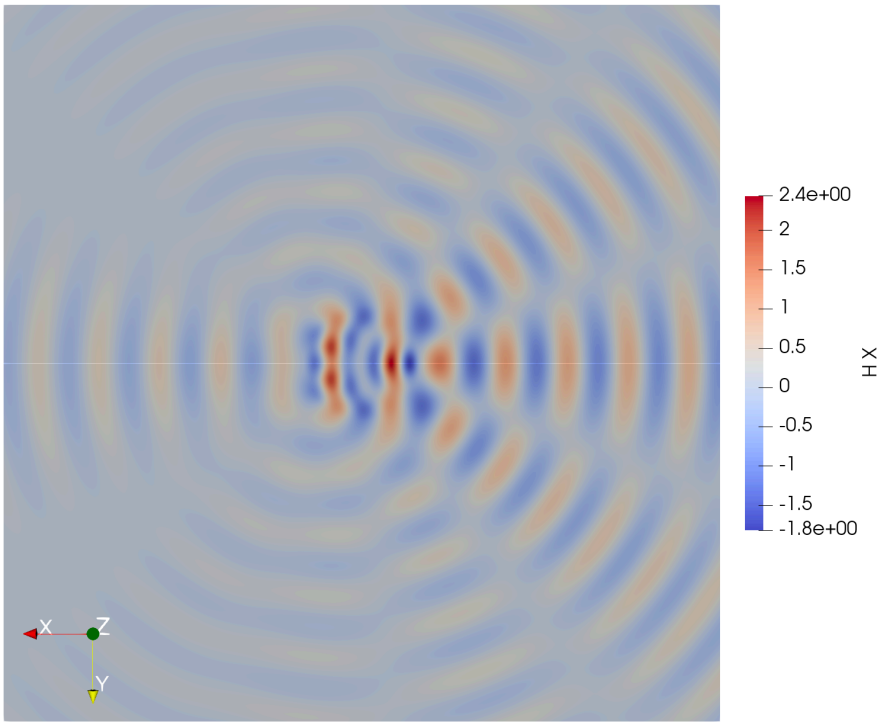


Figure 2.15: Real part of the numerical scattered solution of the Mie test case using $\mu_0 = 1$, $\varepsilon_0 = 1$, $\mu_r = 4$ and $\varepsilon_r = 1$.

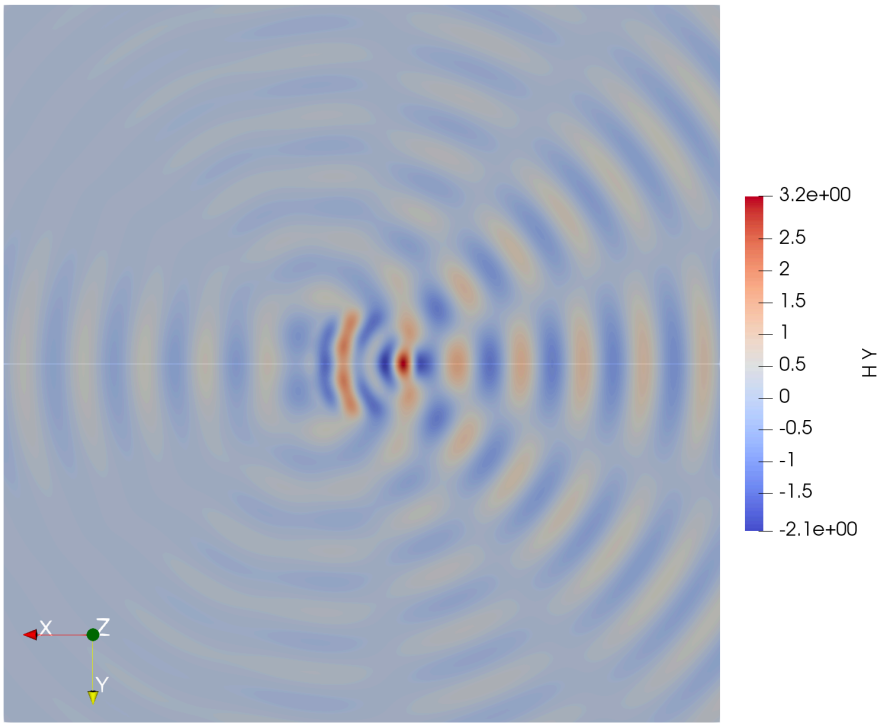


Figure 2.16: Complex part of the numerical scattered solution of the Mie test case using $\mu_0 = 1$, $\varepsilon_0 = 1$, $\mu_r = 4$ and $\varepsilon_r = 1$.

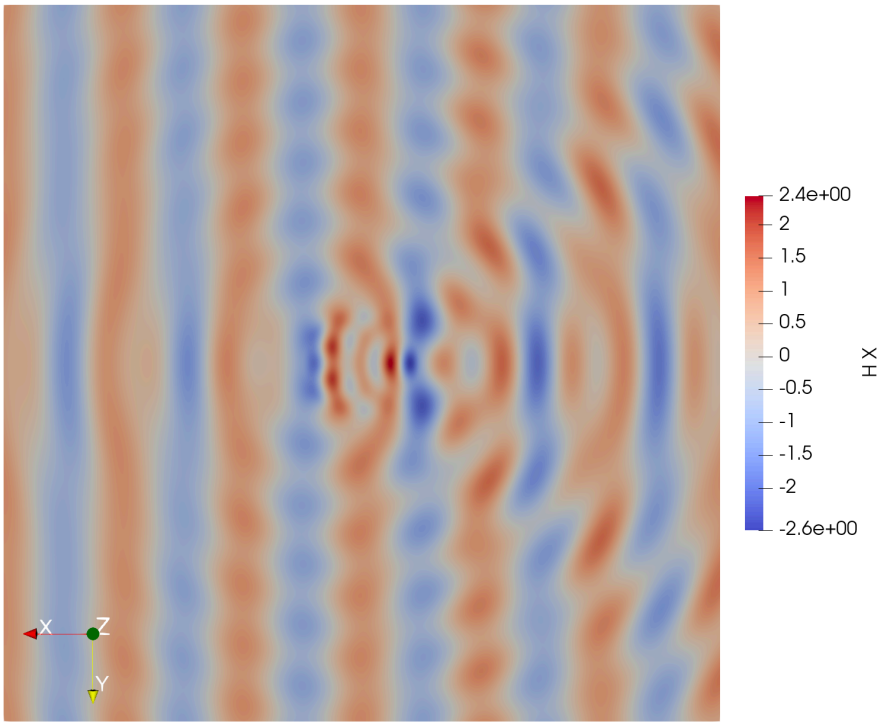


Figure 2.17: Real part of the reconstructed total solution of the Mie test case using $\mu_0 = 1$, $\varepsilon_0 = 1$, $\mu_r = 4$ and $\varepsilon_r = 1$.

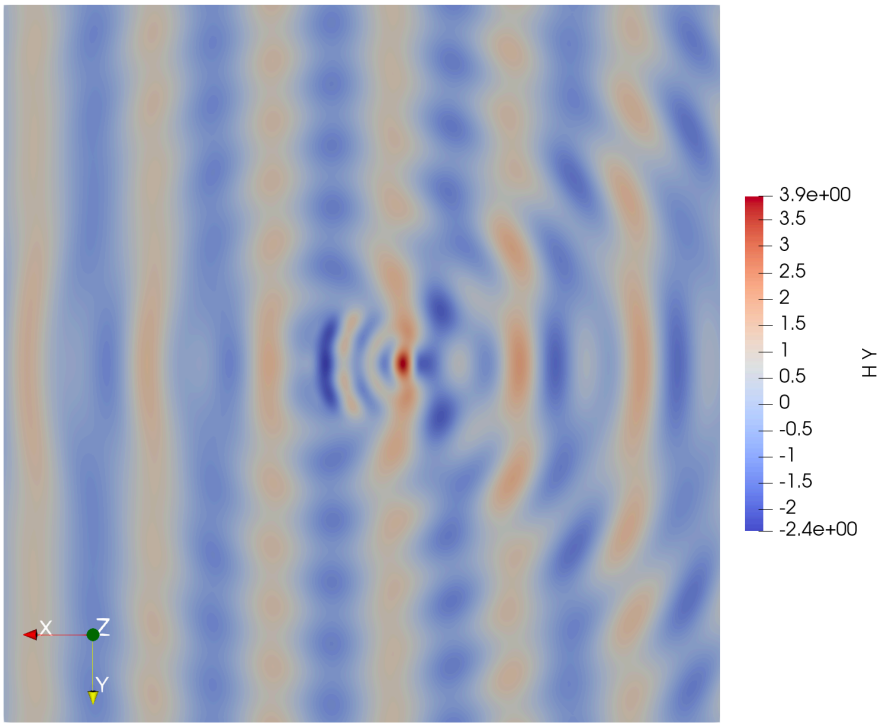


Figure 2.18: Complex part of the reconstructed total solution of the Mie test case using $\mu_0 = 1$, $\varepsilon_0 = 1$, $\mu_r = 4$ and $\varepsilon_r = 1$.

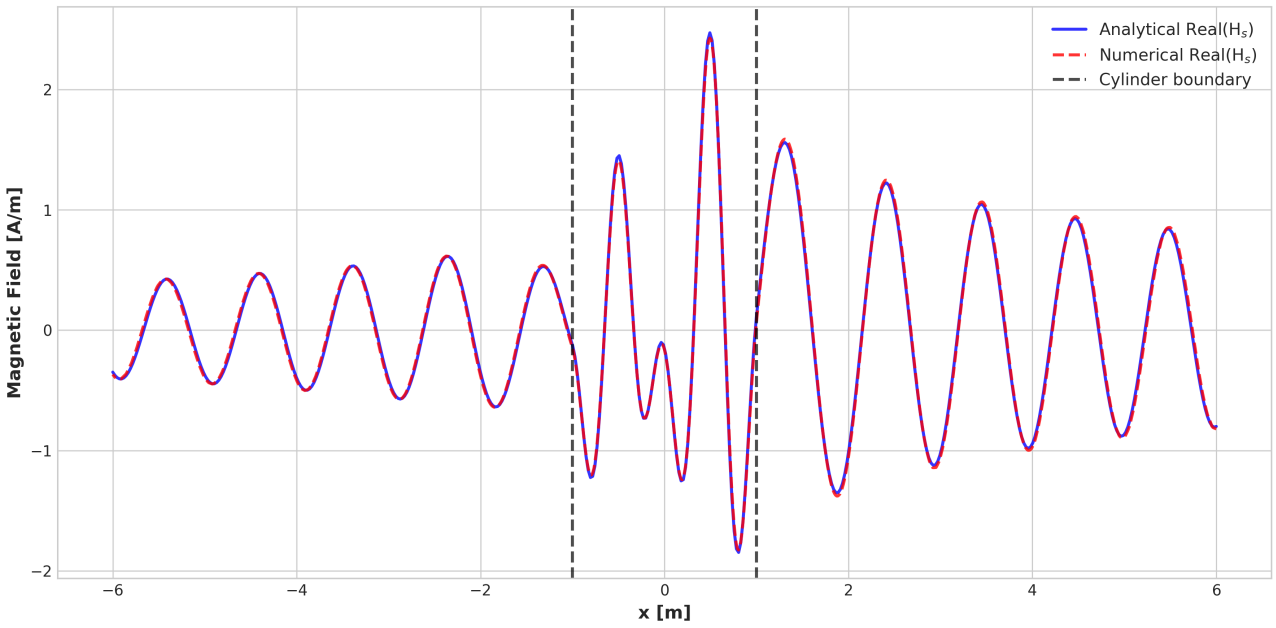


Figure 2.19: Centerline values for the real part of the scattered solution of the Mie test case using $\mu_0 = 1$, $\varepsilon_0 = 1$, $\mu_r = 4$ and $\varepsilon_r = 1$.

CHAPTER 3

DIRECT SOLVERS

Contents

3.1 Introduction

The preceding chapter established the mathematical and numerical formulation for modeling GAAFETs, culminating in a large system of linear equations of the form $Ax = b$. The next critical step is to solve this matrix system to determine the unknown field coefficients in the vector x.

This chapter focuses on identifying an optimal direct solver for the *HELM* framework. A core design principle of HELM is to maintain a "GPU-resident" computation, minimizing data movement between the host and device. This led us to explore GPU-accelerated direct solvers.

Our investigation started with the options in *Trilinos Amesos2*, which pointed to *NVIDIA cuSOLVER*. Subsequently, we identified *NVIDIA* newer, more specialized *cuDSS* library as a promising alternative. This chapter will therefore analyze these prominent GPU-based direct solvers, comparing their performance and memory footprints to determine the best fit for our problems.

3.2 *Amesos2* solvers

Within the extensive solver ecosystem provided by the *Trilinos* framework, direct solvers serve as a crucial baseline for robustness and accuracy. The *KLU2* solver, accessible in *Trilinos* via the *Amesos2* interface, is a highly effective direct method for sparse, non-symmetric linear systems. It is particularly optimized for problems with a block-triangular structure, such as those arising from circuit simulations [16].

Due to its versatility and general purpose, the *KLU2* solver has been adopted as the reference solving technique within the research group of *Professor M. Arnst* As it is capable of robustly handling any non-singular linear system generated by our methods, provides sufficiently rapid solutions for the small- to medium-sized problems used for testing and is therefore used to validate all cases in *HELM*.

3.2.1 Core Methodology: Sparse *LU* Factorization

To appreciate both its strengths and limitations, it is instructive to examine the sequence of operations *KLU2* performs. The algorithm is a multi-phase process that combines techniques from graph theory and numerical linear algebra to solve the system $Ax = b$ [16, 17].

Symbolic Preordering

The process begins by analyzing the matrix sparsity pattern to find an optimal ordering. A key feature of *KLU2* is its ability to permute the matrix into Block Triangular Form (BTF). This decomposition breaks the larger problem into a sequence of smaller, simpler sub-problems that can be solved one after another. Within each of these sub-blocks, a second ordering is applied to minimize the creation of new non-zero entries (fill-in) during the factorization, thus saving memory and computational effort [16]. By default, *KLU2* performs an *approximate minimum degree* reordering.

Numerical Factorization

With the optimal ordering in place, the algorithm proceeds to the numerical factorization. It computes the L and U factors for each diagonal block, incorporating partial pivoting to ensure the process is numerically stable. This is the most computationally intensive phase of the solution.

Solution Phase

Finally, the solution is efficiently obtained through a sequence of forward and backward substitutions, leveraging the factored block-triangular structure.

This multi-phase approach is particularly advantageous for problems requiring solutions with multiple right-hand sides. In such cases, *KLU2* can reuse the expensive symbolic preordering from the first step, making any subsequent "refactorization" significantly faster than the initial solve.

However, a critical limitation is that *KLU2*, as available through the *Amesos2* interface, is a single-process and single-threaded solver. This means it is neither MPI-aware for distributed-memory parallelism nor GPU-aware for accelerator-based computing, which is a major drawback when seeking effective solutions for the large-scale linear systems that are the focus of this work.

3.2.2 The Fundamental Bottleneck: An Inherently Serial Algorithm

Despite its robustness, the primary limitation of *KLU2* in the context of high-performance computing is its inherently serial nature. This is not a shortcoming of its implementation but rather a fundamental characteristic of the "Gilbert-Peierls algorithm" [17] on which it is based.

The lack of parallelism stems from data dependencies inherent in the factorization process. The calculation of the k-th column of the L and U factors depends directly on the results of the preceding $k-1$ columns as explained in [17]. This creates a long dependency chain that forces the algorithm to proceed in a largely sequential manner, with no significant independent tasks that can be distributed across multiple processor cores or nodes. The algorithm reliance on dynamic data structures to manage fill-in further complicates attempts at parallelization.

This stands in stark contrast to parallel direct solvers like *MUMPS* [18] or *SUPERLU_DIST* [19] (also available in *Trilinos*through the use of third-party libraries), which are built on different algorithmic principles. These advanced methods are based on multifrontal techniques for instance, which reorganize the factorization process into a dependency graph known as an "elimination tree" as shown in [17]. The tasks associated with different branches of this tree can be computed independently, enabling effective parallelization across many cores and distributed-memory nodes.

KLU2 does not employ such a parallel strategy by design. Consequently, its performance is limited by the clock speed of a single CPU core. On modern multi-core architectures, using *KLU2* means that the majority of the available computational resources will remain idle during the linear solve.

3.3 GPU-accelerated Solver

3.3.1 *cuSOLVER*

To effectively accelerate the solution of the linear systems at hand, one attractive option is to leverage the parallel processing power of modern GPU. The *Amesos2* package provides a pathway to this by offering an adapter to the *NVIDIA® cuSOLVER* library, a high-performance collection of dense and sparse linear algebra routines [20].

The Existing Interface and Its Limitation

A detailed examination of the *Amesos2- cuSOLVER* adapter reveals a critical limitation for the problem central to this thesis. The currently implemented interface is designed specifically for *Cholesky* factorization. *Cholesky* factorization is a highly efficient matrix decomposition technique that decomposes a matrix A into the product of a lower triangular matrix L and its transpose L^T as explained

in [17], that is, $A = LL^T$. Crucially, its use is restricted to matrices that are Hermitian and, most importantly, positive-definite. As established in previous sections, the linear system resulting from the finite element discretization of the Helmholtz equation is Hermitian but highly indefinite, particularly at high frequencies. Therefore, the default *cuSOLVER* interface provided by *Amesos2* is not applicable for this work.

Available Alternatives

This limitation is within the *Amesos2* adapter, not the *cuSOLVER* library itself. The full *cuSOLVER* library is a comprehensive suite that includes other robust factorizations suitable for general matrices. The most relevant alternatives are:

- ***LU* factorization with partial pivoting and optional reordering**: This method decomposes a general square matrix A into a permuted lower-triangular matrix L and an upper-triangular matrix U, such that $PA = LU$ (now deprecated in *cuSOLVER*V12.8).
- ***QR* factorization**: This method decomposes any rectangular matrix A into the product of an orthogonal matrix Q and an upper-triangular matrix R, such that $A = QR$.

Neither of these methods requires the matrix to be positive-definite, making them theoretically sound choices for solving the indefinite systems arising from the Helmholtz equation.

Given that these suitable factorization routines are not yet wrapped by the standard *Amesos2* interface, a direct, custom implementation is required to bridge the gap between our HELM framework and the high-performance capabilities of *cuSOLVERQR* solver.

Limitations of *LU* in *cuSOLVER*

However, the critical point to note regarding the LU factorization, is that in the official *NVIDIA* documentation, one may find *only CPU (Host) path is provided*, indicating that the function is executed on CPU. Moreover, this routine "follows Gilbert and Peierls algorithm" [20]. As we have established, the Gilbert-Peierls algorithm is an inherently sequential, left-looking method. Its data dependency structure makes it fundamentally unsuitable for the massively parallel architecture of a GPU. The mention of this specific algorithm therefore serves as confirmation that *cusolverSpXcsrlsvlu* is a serial implementation.

Consequently, while an LU solver is present in the *cuSOLVER*library, it does not provide the GPU acceleration sought for this project. This reinforces the conclusion that a custom interface to a different, truly GPU-accelerated routine—such as the sparse *QR* factorization also available in *cuSOLVER*, is the necessary path forward to achieve the desired performance gains.

3.3.2 *cuDSS*

As GPU architectures evolve, so do the software libraries designed to harness their power. *NVIDIA*® has recently introduced the *cuDSS* library, a new generation sparse direct solver intended as the successor to the sparse routines in *cuSOLVER* [21]. It has been designed to provide a more flexible and modern interface for solving large sparse linear systems on GPU.

Available Factorizations

A key improvement in *cuDSS* is a more specialized and comprehensive set of matrix factorization routines. This allows users to select a solver that precisely matches the mathematical properties of their matrix, leading to gains in efficiency and memory usage. The library implements the following primary decomposition techniques:

- **LDU Factorization**: This is the most general method, applicable to any non-singular square matrix. It decomposes the matrix A into the product of a unit lower-triangular matrix L, a diagonal matrix D, and a unit upper-triangular matrix U. This is the robust, go-to factorization when no special properties of the matrix can be assumed.
- **LDL$^{\mathbf{T}}$ Factorization**: This is a specialized and more efficient version for real symmetric matrices. It decomposes the matrix A into LDL^T, taking advantage of the symmetry to reduce computation and memory requirements by only needing to store the L and D factors.
- **LDL$^{\mathbf{H}}$ Factorization**: This extends the symmetric concept to the complex domain for complex Hermitian matrices (where A equals its own conjugate transpose, $A = A^H$). It decomposes the matrix into LDL^H, again providing significant performance benefits by exploiting this specific structure.
- **LL$^{\mathbf{T}}$ and LL$^{\mathbf{H}}$ Factorizations (*Cholesky*)**: These are the highly efficient *Cholesky* decompositions, applicable only to matrices that are symmetric positive-definite (SPD) for the real case (LL^T) or Hermitian positive-definite for the complex case (LL^H).

Factorization for Helmholtz

The significance of this diverse offering becomes clear when considering the properties of the linear system derived from the Helmholtz equation. In this work, the matrix is large, sparse, complex-valued, and symmetric ($A = A^T$), but critically, it is indefinite.

This analysis immediately rules out the *Cholesky*-based routines (LL^T, LL^H) which requires positive-definiteness. Furthermore, the complex-symmetric nature of the matrix means it is not Hermitian ($A \neq A^H$), making the specialized LDL^H factorization inapplicable. The LDL^T solver is also not an option, as the matrix is complex.

Therefore, the general-purpose LDU factorization provided by *cuDSS* emerges as the correct and most suitable choice. It provides a direct, GPU-accelerated pathway for precisely the class of matrices encountered in this work, without imposing any requirements that the Helmholtz matrix fails to meet. The availability of this robust, general-purpose decomposition in a modern, high-performance library makes *cuDSS* a highly compelling candidate for accelerating the solution process in the HELM framework.

3.4 Reordering Algorithms

Reordering algorithms are essential for sparse direct solvers, as they minimize memory usage and computational cost. In the current workflow, this crucial reordering step is performed on the host CPU, even for GPU-based solvers like cuDSS. This necessitates transferring the matrix structure from the device to the host and the resulting permutation back to the device. This host-side computation and data transfer represent a key bottleneck and an important remaining step on the roadmap to a fully GPU-resident solver.

3.4.1 Fill-in

The performance of a direct solver during the factorization of a sparse matrix is primarily determined by the matrix sparsity pattern, rather than the numerical values of its non-zero entries. The main computational bottleneck is a phenomenon known as fill-in: the creation of new non-zero entries during Gaussian elimination in the factorization LU, QR or LL^T. These new entries occupy positions where

the original matrix, A, contained zeros. Excessive fill-in dramatically increases both the memory required to store the factors and the floating-point operations needed for the factorization. An illustration of simple fill-in effect is given in Fig. 3.1.

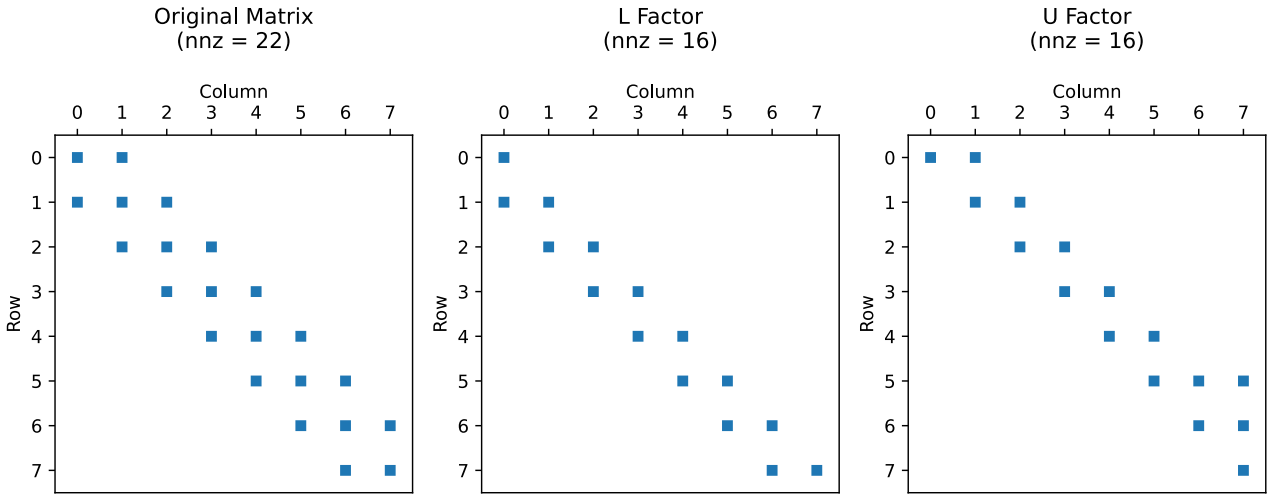


Figure 3.1: Illustration of simple fill-in effect on a small matrix.

To mitigate fill-in, a crucial pre-processing step known as reordering is employed. Reordering algorithms find a permutation matrix P that rearranges the rows and columns of the original matrix. The solver then factorizes the symmetrically permuted matrix PAP^T. While producing a numerically equivalent solution (in exact arithmetic), this permuted system can yield drastically less fill-in. This process is purely symbolic, operating on the graph representation of the matrix sparsity. Modern numerical libraries, such as *NVIDIA cuSOLVER* and *cuDSS*, provide access to a suite of established reordering algorithms.

The problem is defined on an unstructured triangular mesh generated by the open-source software `Gmsh`. The solution within each triangular element is approximated using a polynomial basis of degree 1 (specifically, Lagrange elements).

This method generates a large, sparse, symmetric positive-definite matrix, which is a common test case for direct solvers.

For this study, we use a grid size of [e.g., 100×100], leading to $11,515$ nodes and $23,032$ elements. The matrix is composed of $11,515$ rows and an initial non-zero count (nnz) of $79,807$.

We first establish a baseline by factorizing the matrix in its original, natural ordering using the *Cholesky* decomposition routine in *cuSOLVER*. The sparsity pattern of this matrix is shown in Figure 3.2.

Without reordering, the factorization process suffers from significant fill-in, increasing the non-zero count in the L factor (nnz_{Chol}) to $10,704,641$. This represents a fill-in factor of (nnz_{Chol}/nnz) of 134.132, demonstrating the high computational and memory cost when reordering is omitted.

In order to obtain the fill-in factor and the number of non-zeros, the permutation vectors are exported and a *Cholesky* factorization is applied in Python to obtain the relevant quantities. The need of such process is needed as *cuSOLVER* and *cuDSS* do not provide ways to count the number of non-zeros in their factorization.

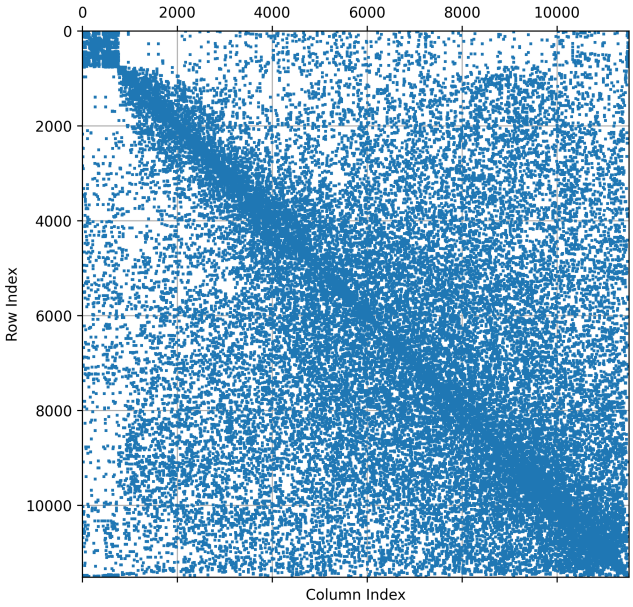


Figure 3.2: Sparsity pattern of the benchmark matrix in its natural ordering.

3.4.2 Symmetric Reverse Cuthill-McKee (SYMRCM)

The SYMRCM algorithm is a classic and widely used method for reducing matrix bandwidth and profile [22, 17]. The core idea is to reorder the nodes of the matrix graph to cluster non-zero entries as closely as possible to the main diagonal. This is achieved by performing a Breath-fist search (BFS) starting from a peripheral node (a node of low degree). The "Reverse" in the name comes from the empirical observation that reversing the ordering produced by the original Cuthill-McKee algorithm often yields even less fill-in for *Cholesky* factorization.

By narrowing the matrix bandwidth, SYMRCM limits the range of row/column interactions during elimination, thereby reducing fill-in. Applying SYMRCM to our benchmark matrix produces the sparsity pattern shown in Figure 3.3. The visual change is important: the non-zeros are now tightly packed around the diagonal. This structural change leads to a significant performance improvement. After factorization, the resulting nnz_{Chol} is $1,303,705$, a dramatic reduction from the baseline. The fill-in factor is now only 16.34, demonstrating the power of this simple and computationally inexpensive reordering strategy.

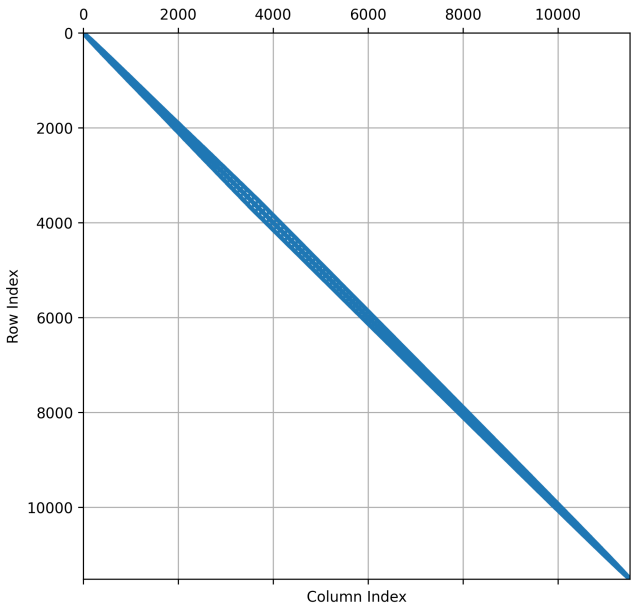


Figure 3.3: Sparsity pattern of the benchmark matrix after applying the SYMRCM reordering algorithm. The non-zero entries are now clustered in a narrow band around the main diagonal.

3.4.3 Symmetric Approximate Minimum Degree (SYMAMD)

Unlike bandwidth-reduction methods like SYMRCM, the Symmetric Approximate Minimum Degree (SYMAMD) algorithm directly targets fill-in reduction [23]. It is a greedy algorithm that, at each stage of a simulated elimination process, selects the node (row/column) with the minimum degree in the current graph. Eliminating this node is heuristically optimal, as its degree serves as a fast-to-compute upper bound on the new non-zero entries it would create in that step.

SYMAMD employs sophisticated heuristics and it identifies supernodes, sets of nodes with identical connectivity, and treats them as single units to accelerate the process. It also performs mass elimination to quickly remove nodes that would not create any fill-in. For many problems, especially those from unstructured 2D and 3D meshes, SYMAMD is considered a ‘workhorse” algorithm, as it often produces significantly lower fill-in than SYMRCM.

Applying SYMAMD to our Poisson benchmark matrix results in the sparsity pattern shown in Figure 3.4. Visually, the pattern lacks the clean, narrow band of SYMRCM, instead showing a more complex, nested structure. However, this ordering is highly effective. The subsequent LU factorization produces a final non-zero count, nnz_{Chol}, of just $570,358$. This corresponds to a fill-in factor of 7.15, which is a marked improvement over both the baseline case and the SYMRCM ordering, highlighting its superior performance for this type of problem.

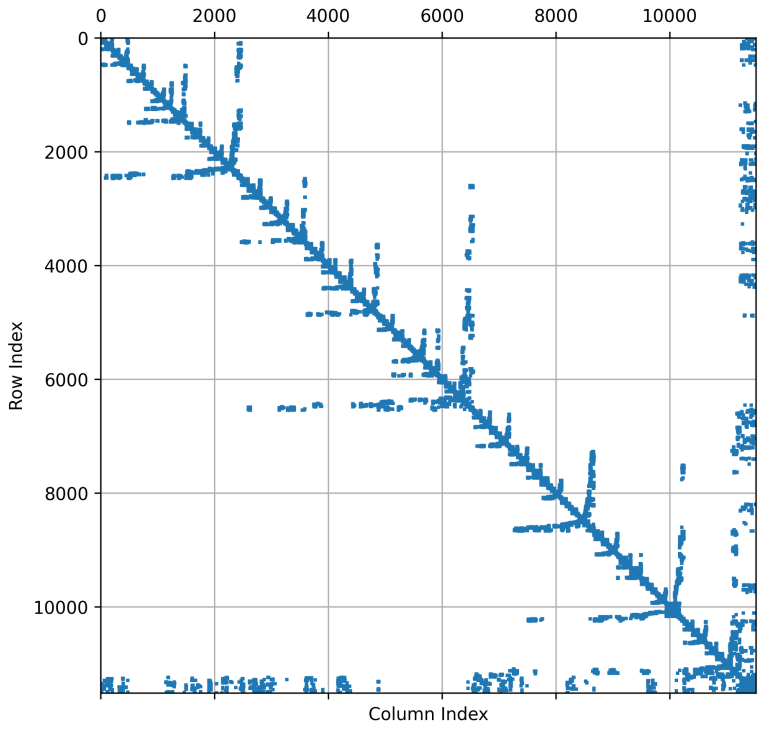


Figure 3.4: Sparsity pattern of the benchmark matrix after applying the SYMAMD reordering algorithm. While visually less regular than the SYMRCM result, this complex ordering is highly effective at minimizing fill-in.

3.4.4 Multilevel Nested Dissection (MND)

Multilevel Nested Dissection is a powerful divide-and-conquer algorithm that operates differently from local, greedy methods. It works by recursively partitioning the matrix graph. At each step, it identifies a small vertex separator—a set of nodes that, when removed, splits the graph into disconnected subgraphs. These separator nodes are ordered last, and the process repeats on the subgraphs.

The MND library, for example, employs an efficient multilevel method [24]: it coarsens the graph to a simpler version, partitions the small graph, and then refines the separator while projecting it back to the original graph.

Applying this algorithm to our benchmark matrix produces the characteristic block-structured pattern shown in Figure 3.5. This method demonstrates its superior quality by yielding the lowest fill-in of all tested algorithms. The final number of non-zeros in the LU factors, nnz_{Chol}, is just $356,563$, corresponding to a fill-in factor (nnz_{Chol}/nnz) of only 4.47. While MND typically has the highest reordering cost, this dramatic reduction in fill-in and its creation of a structure ideal for parallel factorization often make it the optimal choice for large-scale systems.

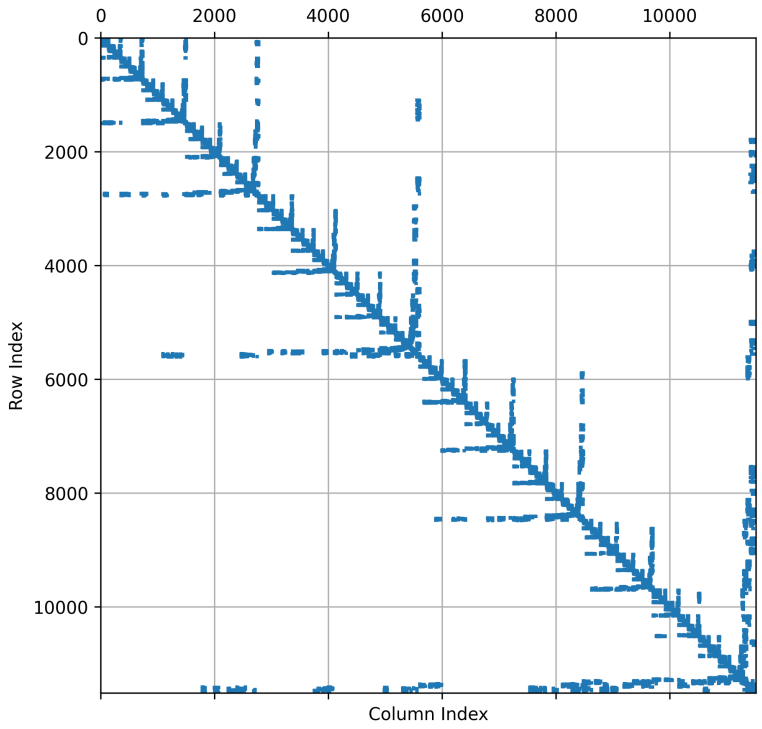


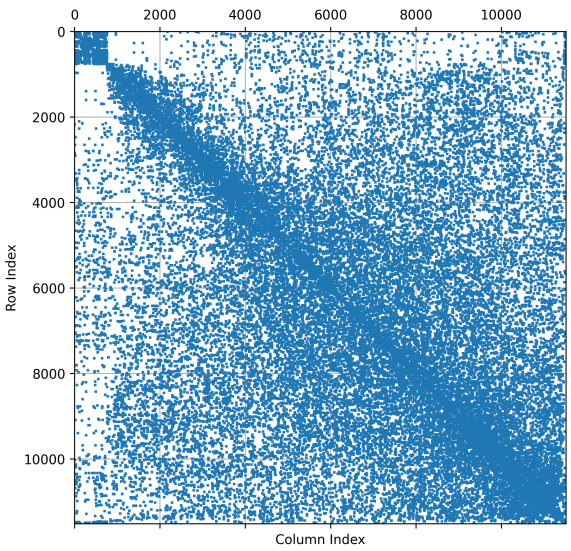
Figure 3.5: Sparsity pattern of the benchmark matrix after applying the Multilevel Nested Dissection algorithm. The characteristic recursive, block-diagonal structure results from the divide-and-conquer approach, with the vertex separators forming the denser blocks along the diagonal.

3.4.5 Effect of different reodering algorithm

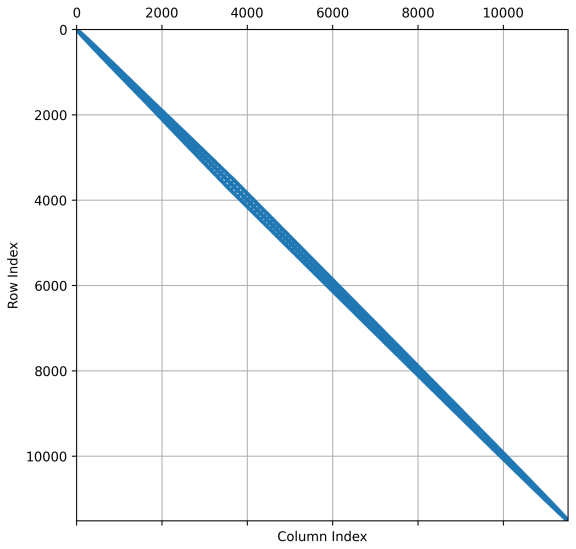
An analysis of reordering methods in *NVIDIA cuSOLVER* and *cuDSS* libraries for a 2D Poisson benchmark (Figures 3.6, 3.7) reveals a clear hierarchy of performance based on the underlying algorithmic strategy:

- **Ineffective Strategies**: The Original (Unordered) Matrix, with its large bandwidth, represents a poor baseline for factorization. More dramatically, the *ALG1* routine in *cuDSS* (Fig. 3.7b) proves to be inefficient, producing a large fill-in ratio of 198.05.
- **Suboptimal (Bandwidth Reduction)**: The SYMRCM algorithm (*cuSOLVER*, Fig. 3.6b) improves on the original ordering by clustering non-zeros near the diagonal. However, its focus is not primarily on minimizing fill-in, resulting in a fill-in ratio of 16.34.
- **Effective (Direct Fill-in Minimization)**: The Approximate Minimum Degree (SYMAMD) algorithm, available in both libraries, is a consistently strong performer. It directly targets fill-in, achieving low ratios of 7.15 (*cuSOLVER*, Fig. 3.6c) and 6.33 (*cuDSS*, Fig. 3.7c), which significantly reduces computational work.
- **Optimal (Nested Dissection)**: Advanced graph-partitioning methods provide the best results. The MND implementation in *cuSOLVER* (Fig. 3.6d) and the customized MND in *cuDSS* (Fig. 3.7d) achieve the lowest fill-in ratios of 4.47 and 5.23, respectively. These establish Multilevel Nested Dissection as the most effective strategy for minimizing memory and computational overhead for this problem.

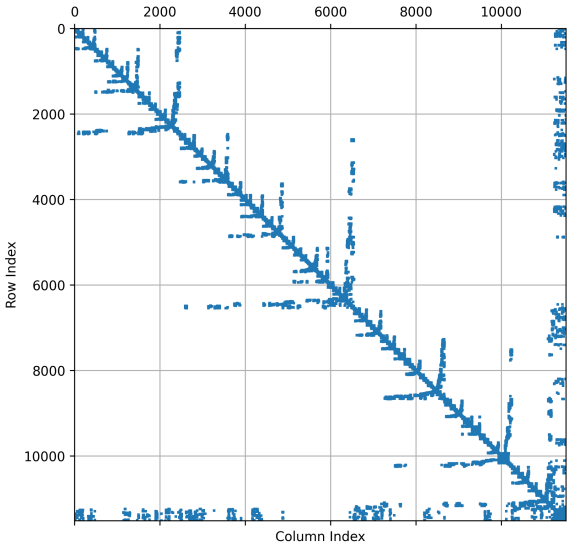
The results from both libraries consistently demonstrate that for the regular structure of the 2D Poisson problem, algorithms that directly minimize fill-in, namely SYMAMD and especially MND, are superior to bandwidth-reduction methods.



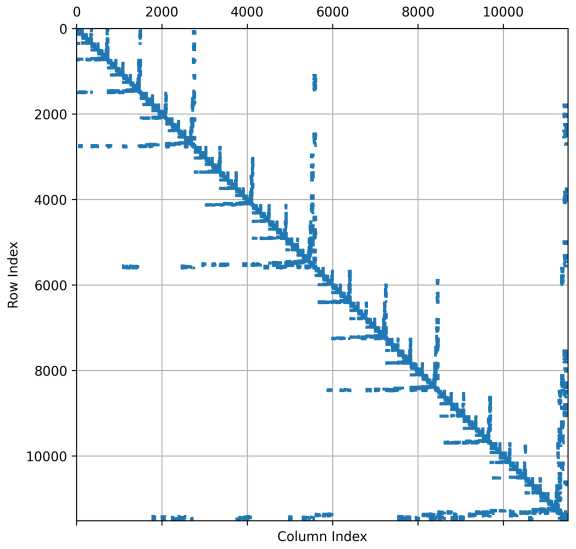
(a) Original ordering of the matrix before the solve phase.



(b) Reordered matrix using the SYMRCM algorithm before solve phase. Fill-in ratio $r = 16.34$ w.r.t the original matrix

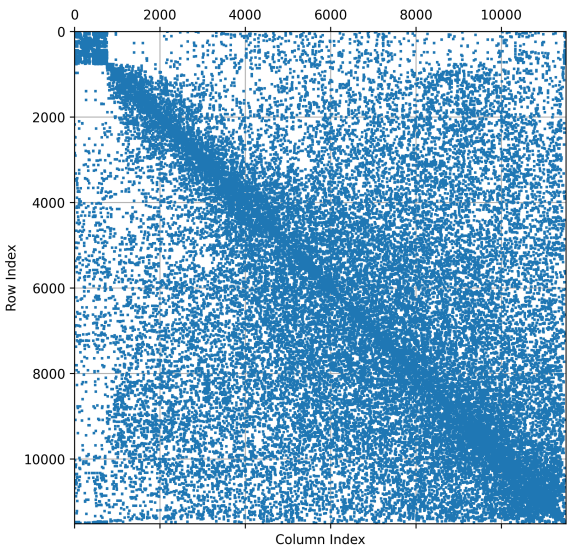


(c) Reordered matrix using the SYMAMD algorithm before solve phase. Fill-in ratio $r = 7.15$ w.r.t the original matrix

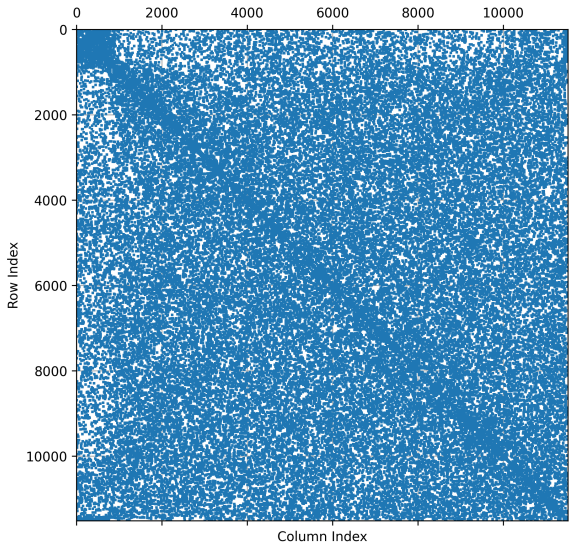


(d) Reordered matrix using the MND algorithm before solve phase. Fill-in ratio $r = 4.47$ w.r.t the original matrix

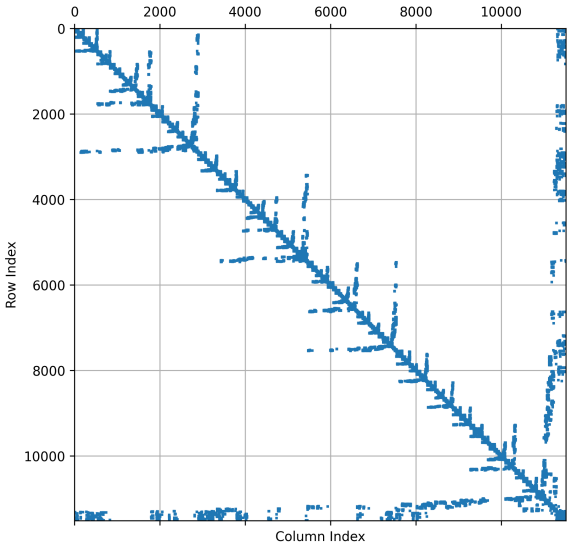
Figure 3.6: Reordering methods provided by the *cuSOLVER* library.



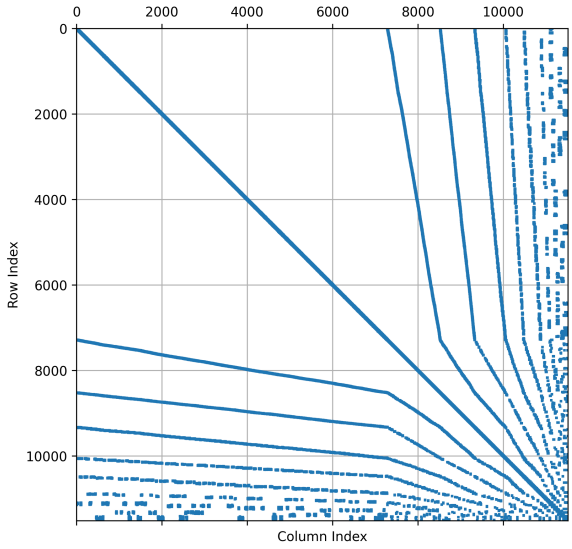
(a) Original ordering of the matrix before the solve phase.



(b) Reordered matrix using the *ALG1*1 algorithm before solve phase. Fill-in ratio $r = 198.05$ w.r.t the original matrix



(c) Reordered matrix using the SYMAMD algorithm before solve phase. Fill-in ratio $r = 6.33$ w.r.t the original matrix



(d) Reordered matrix using the MND algorithm before solve phase. Fill-in ratio $r = 5.23$ w.r.t the original matrix

Figure 3.7: Reordering methods provided by the *cuDSS* library.

3.4.6 Results summary

Reordering	*cuSOLVER*		*cuDSS*	
Algorithm	NNZ in Factors	Fill-in Ratio	NNZ in Factors	Fill-in Ratio
SYMRCM/ALG1	1 303 705	16.34	15 805 417	198.05
SYMAMD	570 358	7.15	505 321	6.33
MND (METIS)	**356563**	**4.47**	**417295**	**5.23**

Table 3.1: Reordering Algorithm Performance in *cuSOLVER* and *cuDSS*. The benchmark uses a 2D Poisson matrix (11,515 x 11,515, initial NNZ = 79 807). The table shows the number of non-zeros (NNZ) in the *Cholesky* factorization and the corresponding fill-in ratio ($NNZ_{LU}/NNZ_{original}$). The best-performing algorithm for each library is highlighted in **bold**.

3.4.7 Memory Management

An evaluation of the *cuSOLVER* library revealed high memory demands during sparse Cholesky factorization, which in some cases exceeded the available 12 GB of GPU memory and prevented the solution of larger systems.

The memory requirements were examined using *cuSOLVER* estimation routines. Two primary buffers were identified: one for storing the L-factor and another for the numerical factorization. Among these, the numerical factorization buffer consistently represented the dominant contribution, with its size increasing rapidly with the number of degrees of freedom. This behavior was found to be largely unaffected by the choice of polynomial degree or reordering algorithm, indicating that the scaling is intrinsic to the algorithmic design of the sparse *Cholesky* factorization.

In comparison, the *cuDSS* library employs a single, unified buffer that integrates all workspace requirements. Moreover, *cuDSS* requires significantly less memory usage, around an order of magnitude smaller than the combined buffers required by *cuSOLVER*. Figure 3.8 illustrates these differences, showing the growth of memory buffers as a function of the problem size.

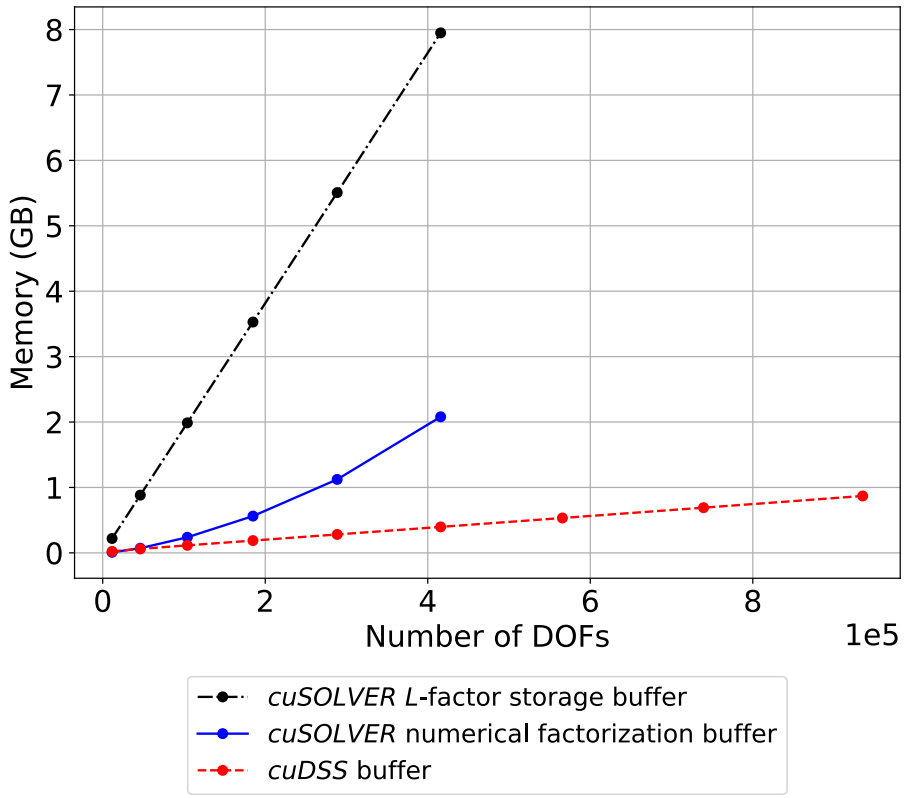


Figure 3.8: Memory buffer requirements for *cuSOLVER* and *cuDSS* as a function of the number of Degrees of Freedom. While both scale with problem size, *cuDSS* demonstrates superior memory efficiency, requiring approximately an order of magnitude less buffer space than the *cuSOLVER* implementation.

3.5 Solver Performance Analysis

3.5.1 Experimental Setup for Two Dimensional Poisson and Mie

To evaluate the performance of the direct solvers and their reordering algorithms, two benchmark problems are considered: a two-dimensional Poisson equation and a two-dimensional Mie scattering problem. Both problems are discretized on triangular meshes generated with `Gmsh` [25], and a polynomial basis of varying order is employed. A parametric study is conducted to assess solver performance under different conditions by systematically varying two key parameters: the polynomial degree of the basis functions (p) and the mesh resolution.

Poisson Case: For the Poisson benchmark, the problem is defined on a square domain. The mesh density is controlled by specifying the number of elements N along each side, resulting in a grid of $N \times N$ nodes and $2N^2$ triangular elements (because of the transfinite usage). The study includes three sets of test cases corresponding to different polynomial degrees, chosen so as to maintain a relatively constant number of DOFs across cases as shown in Section 3.5.1

Polynomial Degree p	N
1	100 to 900
2	50 to 450
4	25 to 225

Mie Case: For the Mie scattering benchmark, the mesh resolution is instead specified by the target element size rather than the number of elements per side. Multiple configurations are generated for polynomial degrees $p = 1, 2, 4$, with the selected element sizes summarized in Section 3.5.1.

Polynomial Degree p	**Average Element Sizes**
1	0.50 to 0.06
2	1.00 to 0.12
4	2.00 to 0.24

3.5.2 Performance on the Two Dimensional Poisson Test Case

The performance of each solver is evaluated based on the total solution time, which encompasses the reordering, symbolic analysis, numerical factorization, and solve phases. To ensure reliable measurements, the reported timings are the average of three independent runs for each test case.

To identify the best GPU approach, we first benchmarked *Cholesky* factorization on the 2D Poisson problem presented in Section 3.5.1 using both *cuSOLVER* (Fig.3.9) and *cuDSS* (Fig.3.10). This initial comparison yielded several key findings:

- **Superiority of MND and SYMAMD:** Algorithms that directly minimize fill-in (MND, SYMAMD) consistently outperformed bandwidth-reduction methods (SYMRCM). In *cuSOLVER*, METIS was the unambiguous winner, as AMD delivered poor performance.
- **Nuanced Performance in *cuDSS*:** The optimal choice in *cuDSS* depended on the problem complexity. SYMAMD was faster for lower-order finite elements, while MND became superior for higher-order elements (e.g., Degree 4) and larger problem sizes.
- **Surprising *cuSOLVER* Underperformance:** Across all tests, the *cuSOLVER* library was consistently and significantly slower than *cuDSS*, even when using the same reordering algorithm.

Based on these results, *cuDSS* paired with an adaptive choice between SYMAMD and MND was identified as the optimal GPU strategy. With this optimized pipeline established, we compared its performance against the serial CPU-based *KLU2* solver. As shown across multiple finite element degrees (Figs. 3.11, 3.12, and 3.13), the GPU solution is consistently faster by nearly an order of magnitude. for reference, the *NATURAL* reordering refers to the original sparsity pattern of the matrix, where no reordering is done.

Another critical aspect differentiating the GPU solvers is their memory efficiency. As discussed in Section 3.4.7, the *Cholesky* solver in *cuSOLVER* has high memory requirements, which drastically limits the maximum solvable problem size. This limitation is evident in Figure 3.9, where the number of degrees of freedom (DOF) does not exceed 4×10^5. In contrast, *cuDSS* demonstrates superior memory management, enabling it to solve problems with nearly an order of magnitude more DOFs, as shown in Figure 3.10.

While *cuDSS* is more scalable, the largest problem it could solve was 700×700 DOFs. This is still smaller than the problem sizes handled by the CPU-based *KLU2* solver, which benefits from access to the larger capacity of system RAM compared to the GPU VRAM.

In conclusion, this analysis reveals two insights. First, the optimal reordering strategy depends on both the software library and problem complexity. Second, this investigation validates that a GPU-accelerated direct solver—specifically, *cuDSS Cholesky* factorization combined with a carefully selected reordering algorithm—provides a compelling high-performance alternative to serial CPU solvers, offering a speedup for the finite element problems studied.

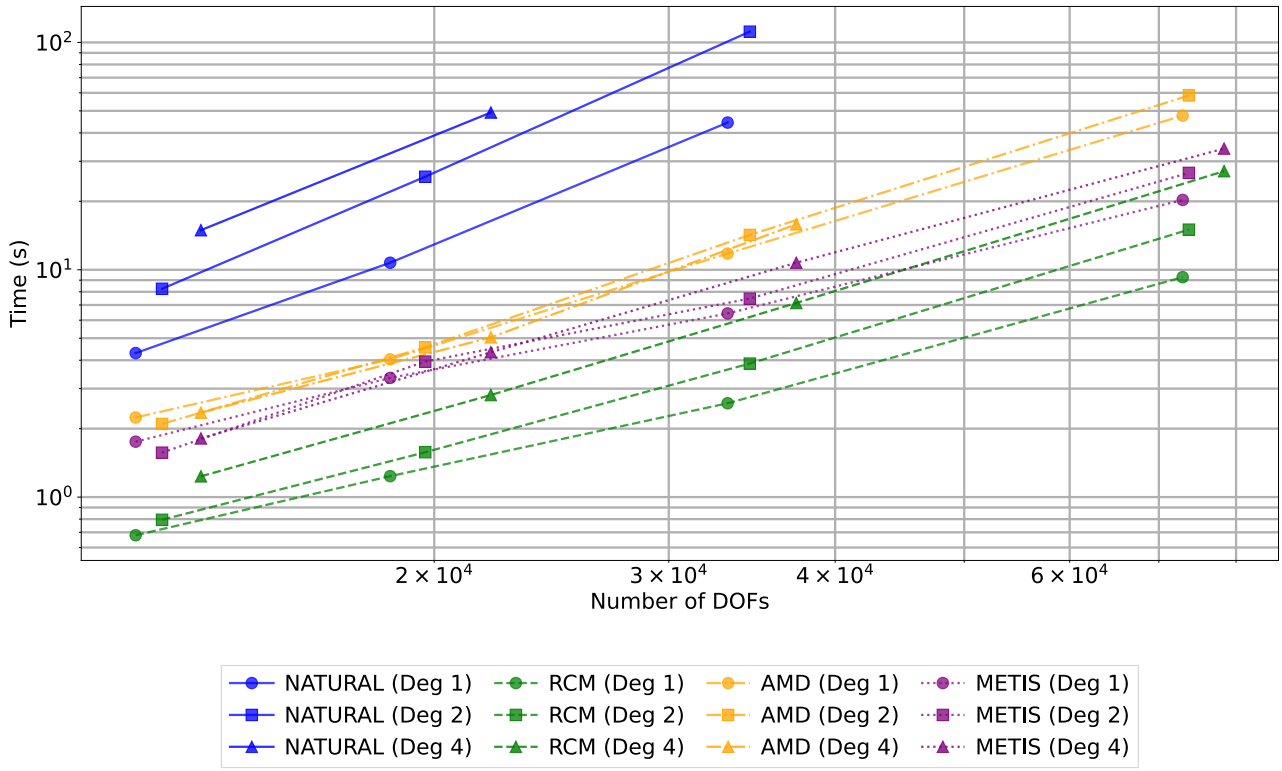


Figure 3.9: Total solution time for different reordering algorithm available for *cuSOLVER*. Note that the tests were run with a maximal number of DOFs of $931,265$ although *cuSOLVER* fails to solve this case whatever the logarithm used.

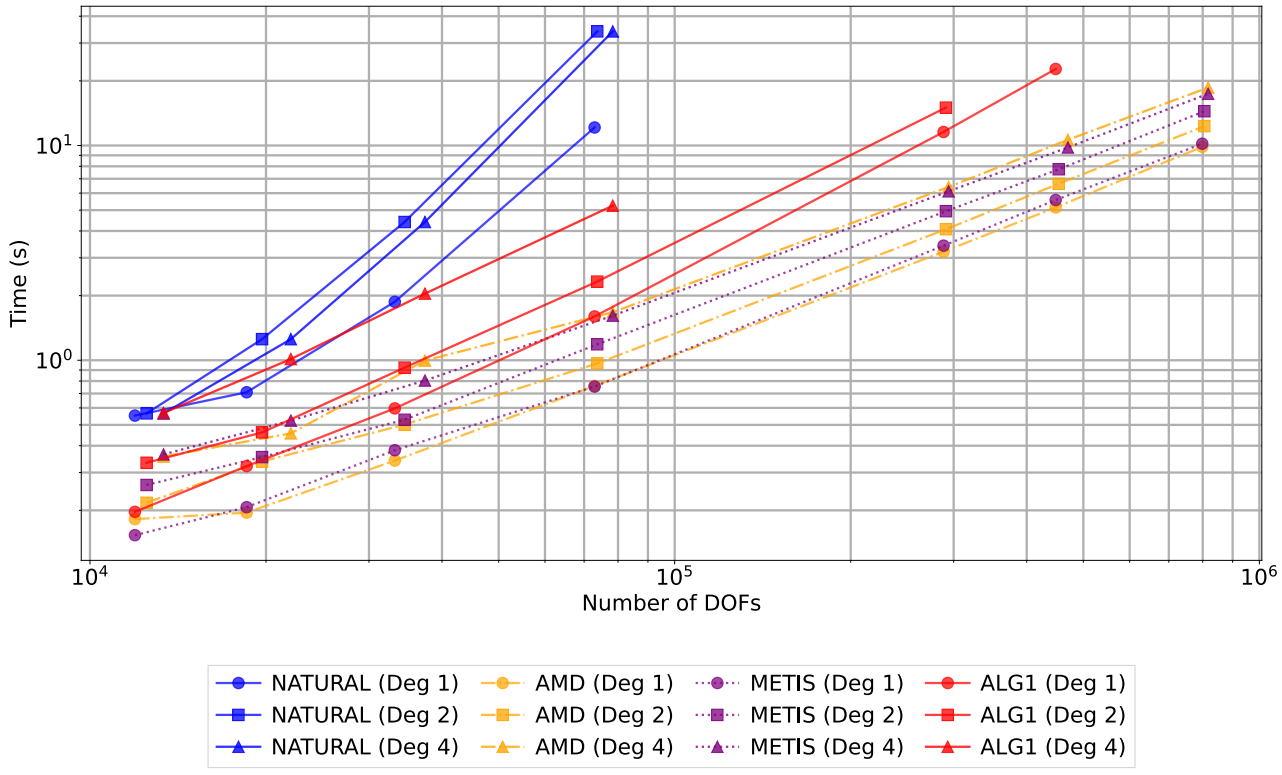


Figure 3.10: Total solution time for different reordering algorithm available for *cuDSS*.

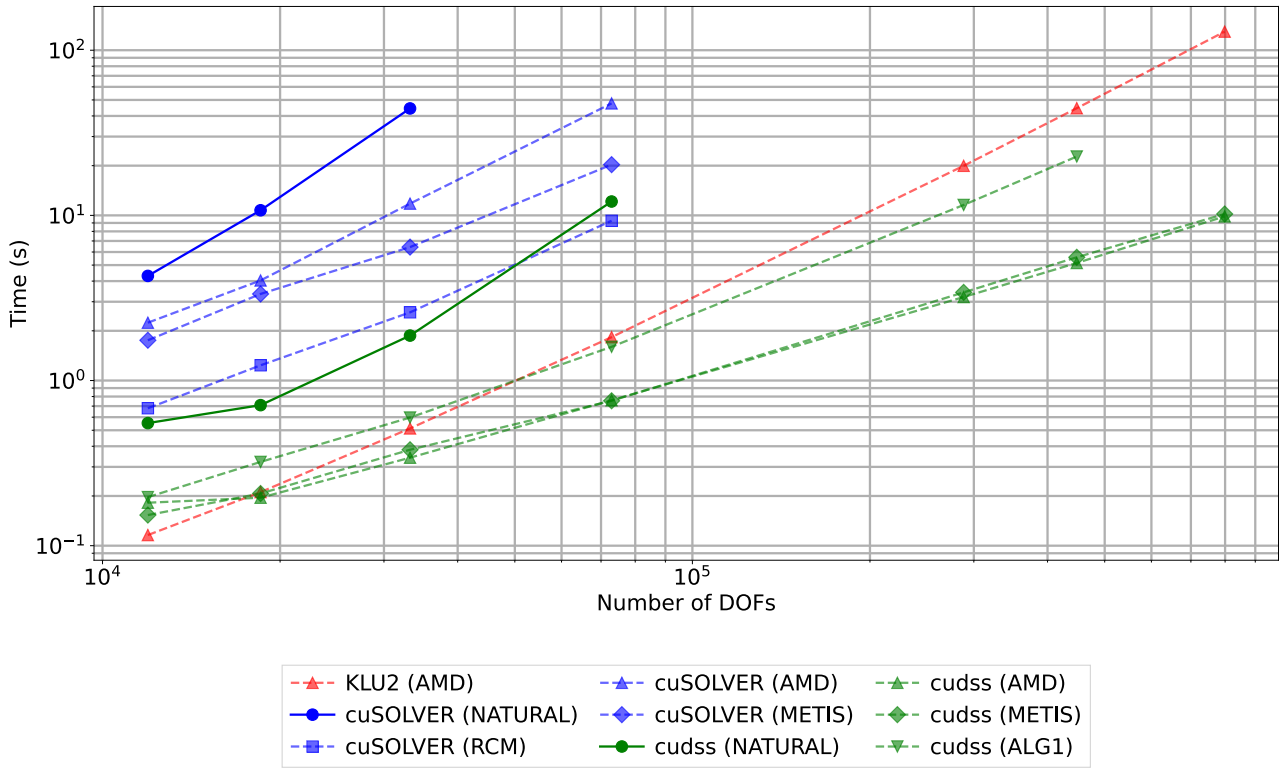


Figure 3.11: Total solution time based reordering algorithms for degree 1 elements.

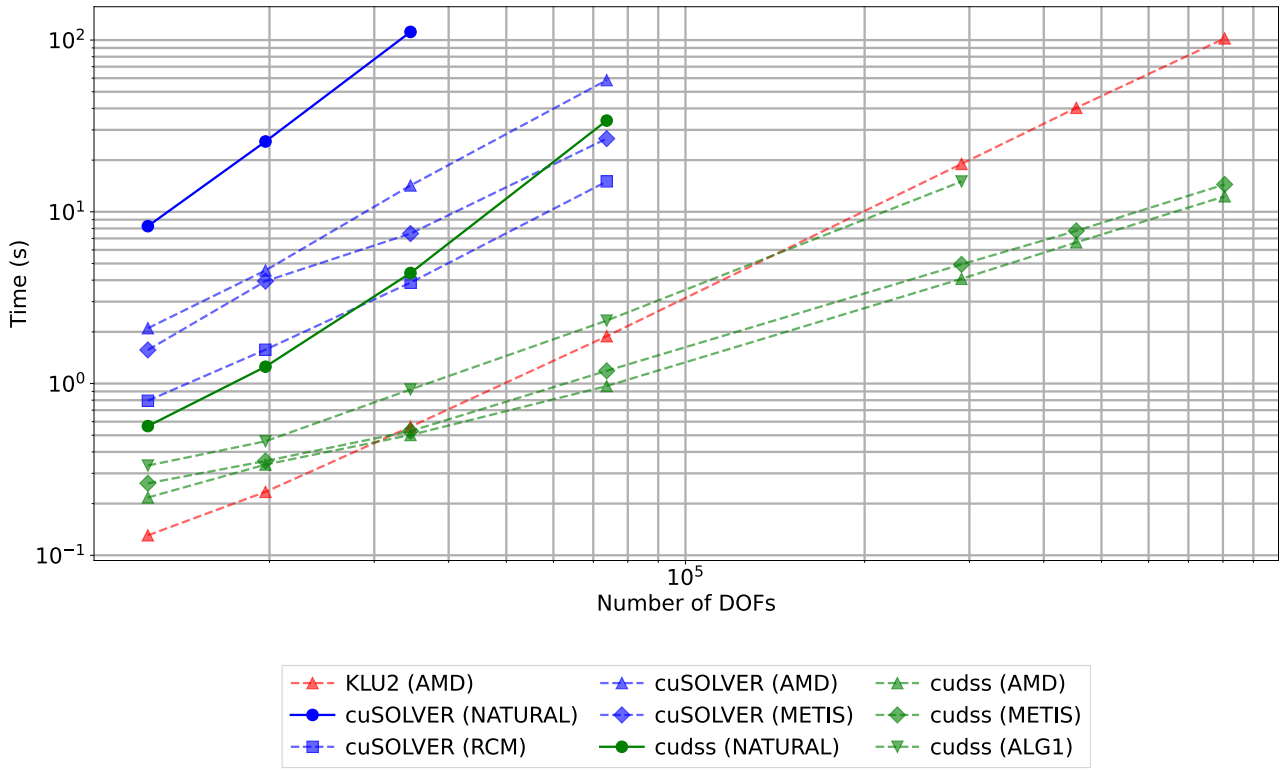


Figure 3.12: Total solution time to solution based reordering algorithms for degree 2 elements.

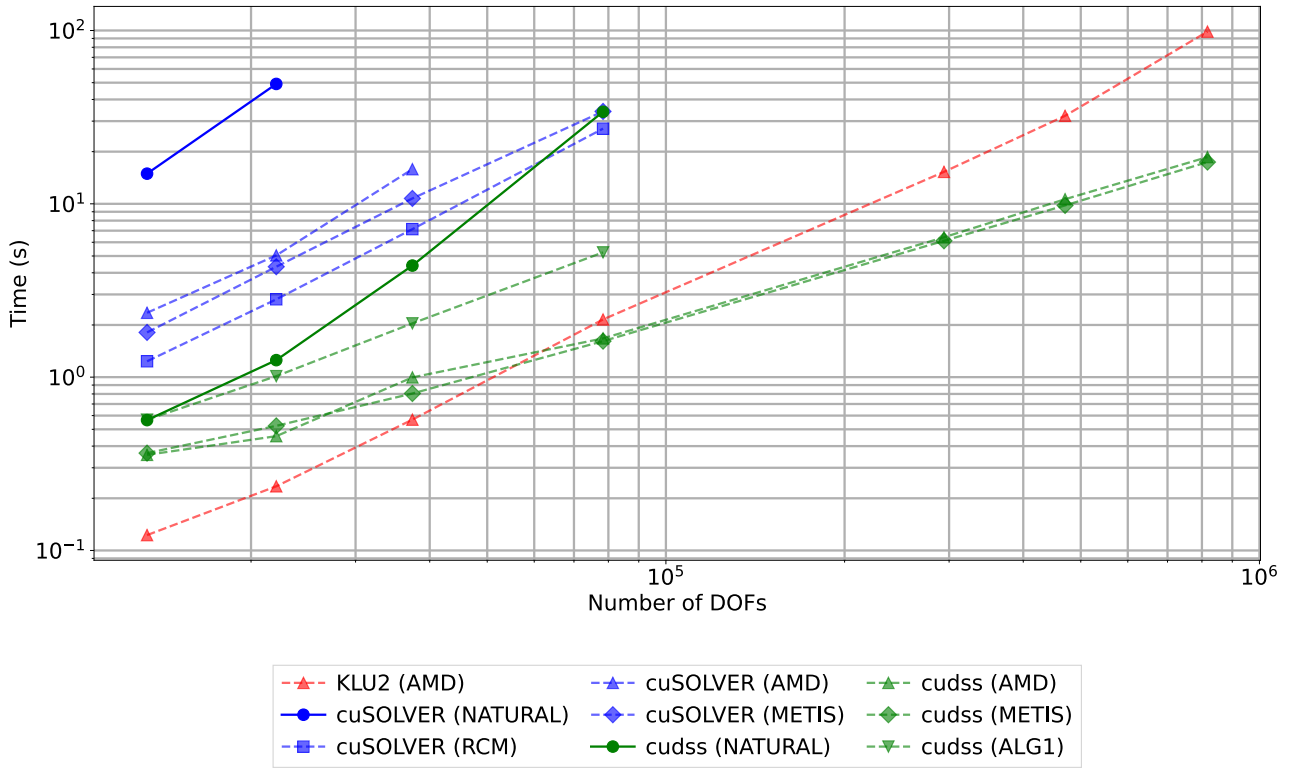


Figure 3.13: Total solution time to solution based reordering algorithms for degree 4 elements.

3.5.3 Performance on the Two Dimensional Mie Test Case

Having established the critical role of reordering, we now address the central challenge: the efficient solution of the linear system arising from the 2D Mie scattering problem. This problem is discretized on an unstructured mesh generated by `Gmsh`. As the resulting Helmholtz matrix is indefinite, this property renders *Cholesky* factorization inapplicable and demands robust numerical methods.

Therefore, for this comparative analysis, we selected the most appropriate factorization from each GPU-accelerated library:

- From *cuSOLVER*[20], we selected the QR factorization, as it is a robust method for general indefinite systems and, crucially, offers a parallel implementation, unlike the library's serial sparse LU routine.
- From *cuDSS* [21], we employed the *LDU* factorization, the library's general-purpose solver designed to handle precisely this class of indefinite, non-Hermitian matrices.

The performance analysis on this problem confirmed the trends observed previously. As shown in the comparisons of the total time to solution (Figures 3.13, 3.16 and 3.17), *cuDSS* is the clear performance leader, consistently delivering faster solutions than both *cuSOLVER* and *KLU2* solver. Within *cuDSS*, the performance of SYMAMD and MND reordering is competitive and often nearly identical, establishing both as excellent choices (Figure 3.15).

Conversely, the *ALG1*routine proved unsuitable, leading to performance degradation. As seen in Figure 3.8, the problem size is still limited for the GPU solvers, since the required buffer size eventually becomes too large to be allocated on the device VRAM.

In conclusion, the analysis of the Helmholtz problem provides that, for indefinite systems of this type, *cuDSS* with either SYMAMD or MND reordering is the undisputed optimal choice among the tested GPU-accelerated methods.

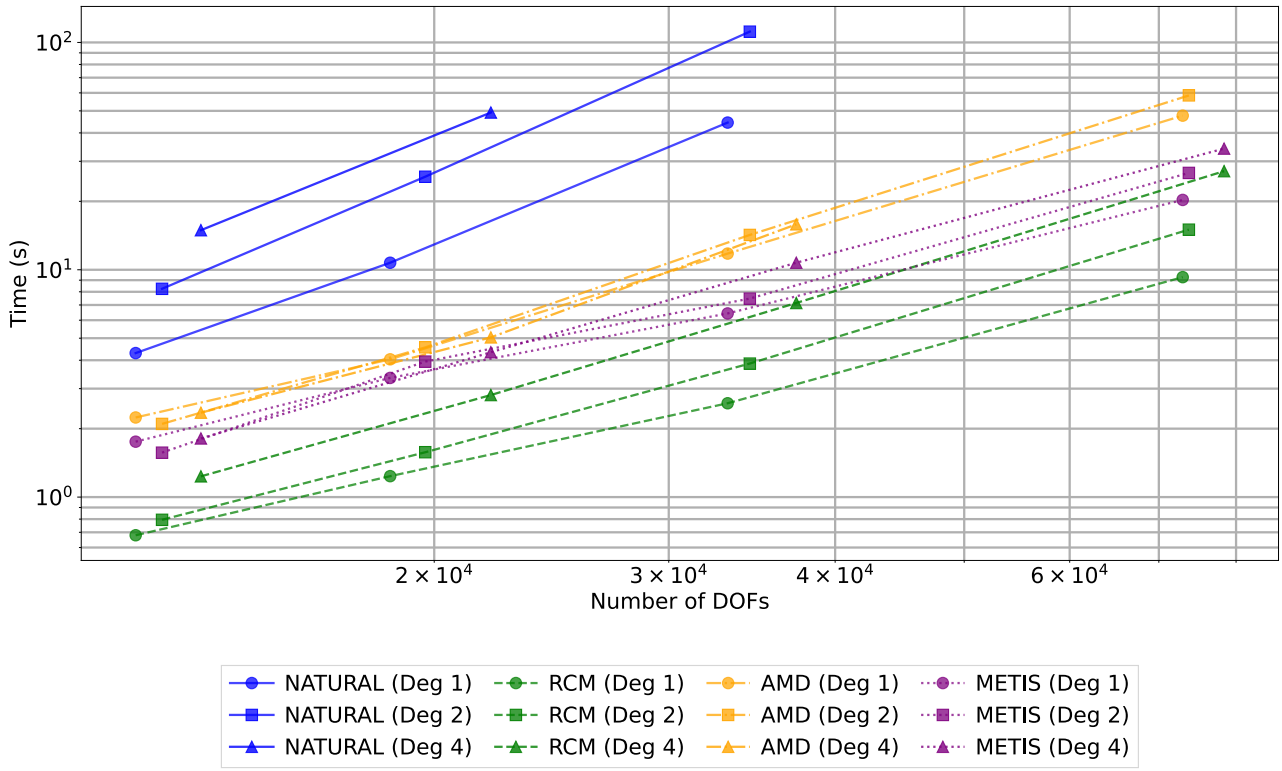


Figure 3.14: Total solution time for different reordering algorithm available for *cuSOLVER*. Note that the test were run with a maximal number of DOFs of $817,361$ although *cuSOLVER* fails to solve this case whatever the laogrithm used.

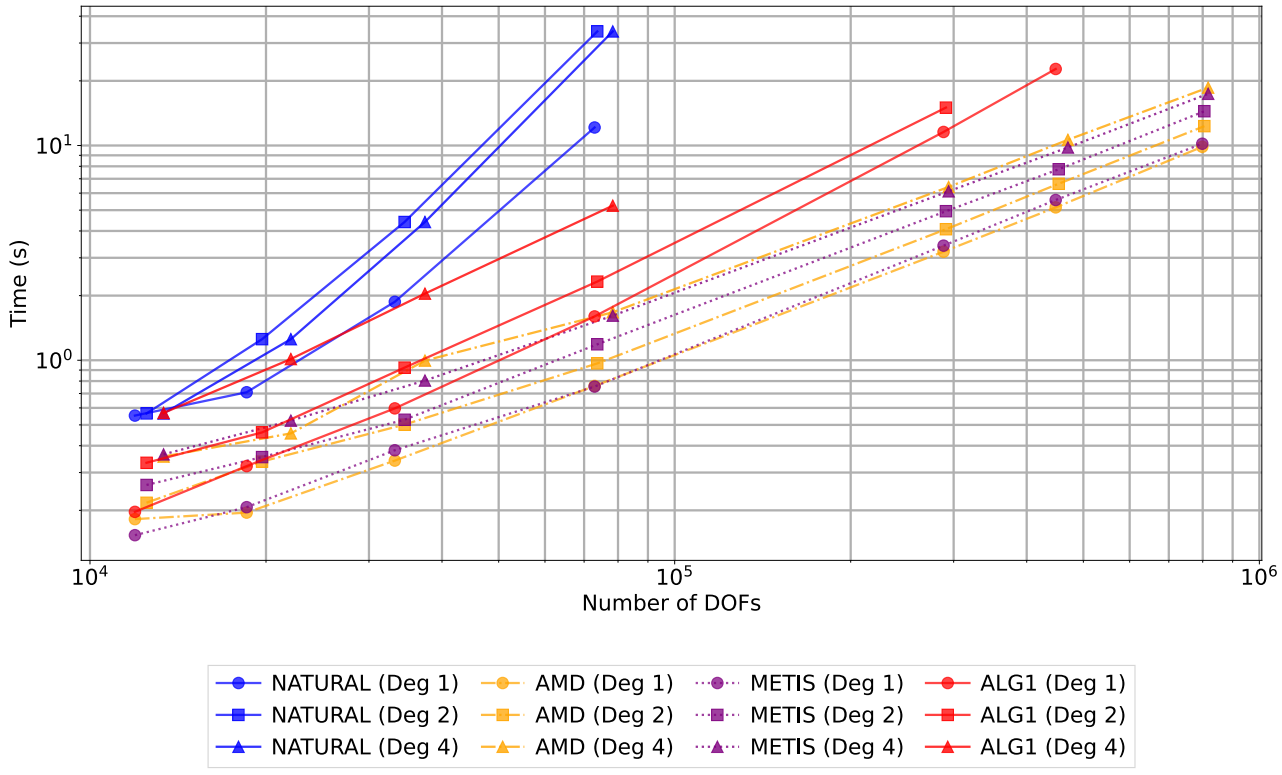


Figure 3.15: Total solution time for different reordering algorithm available for *cuDSS*.

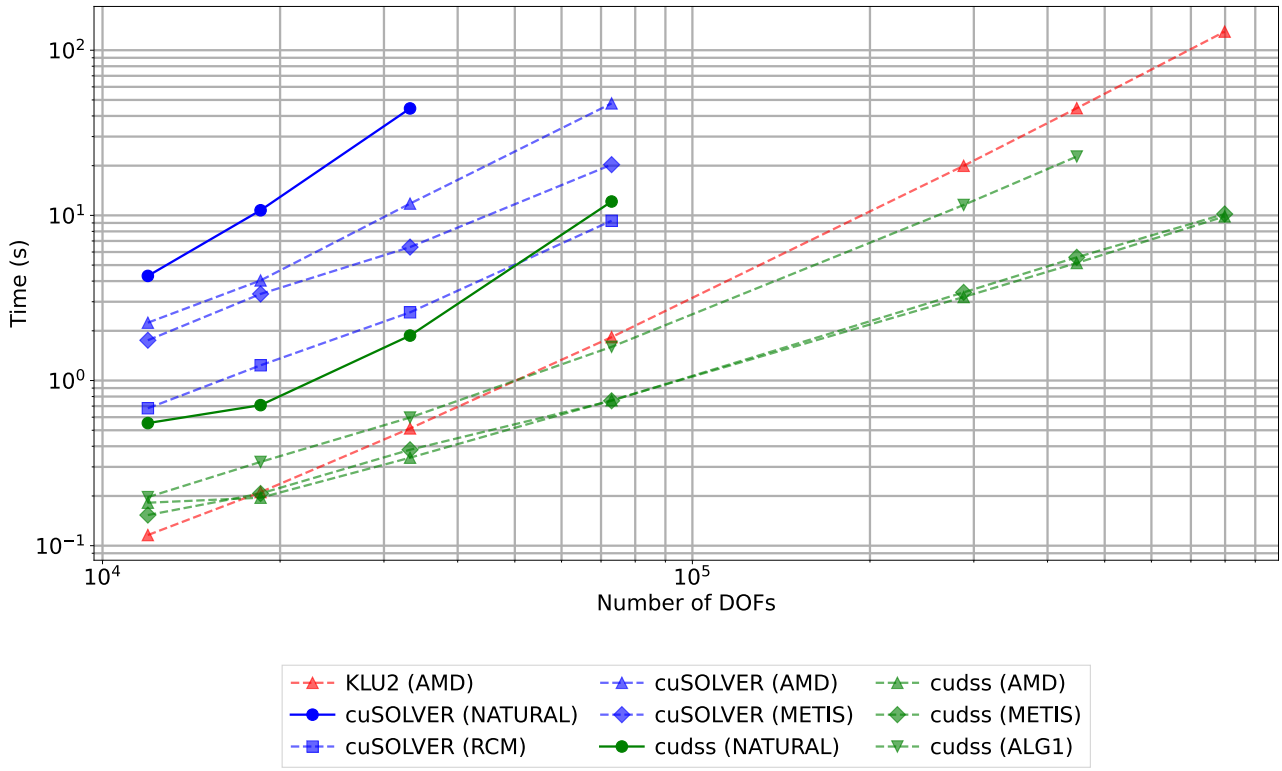


Figure 3.16: Total time to solution based reordering algorithms for degree 1 elements.

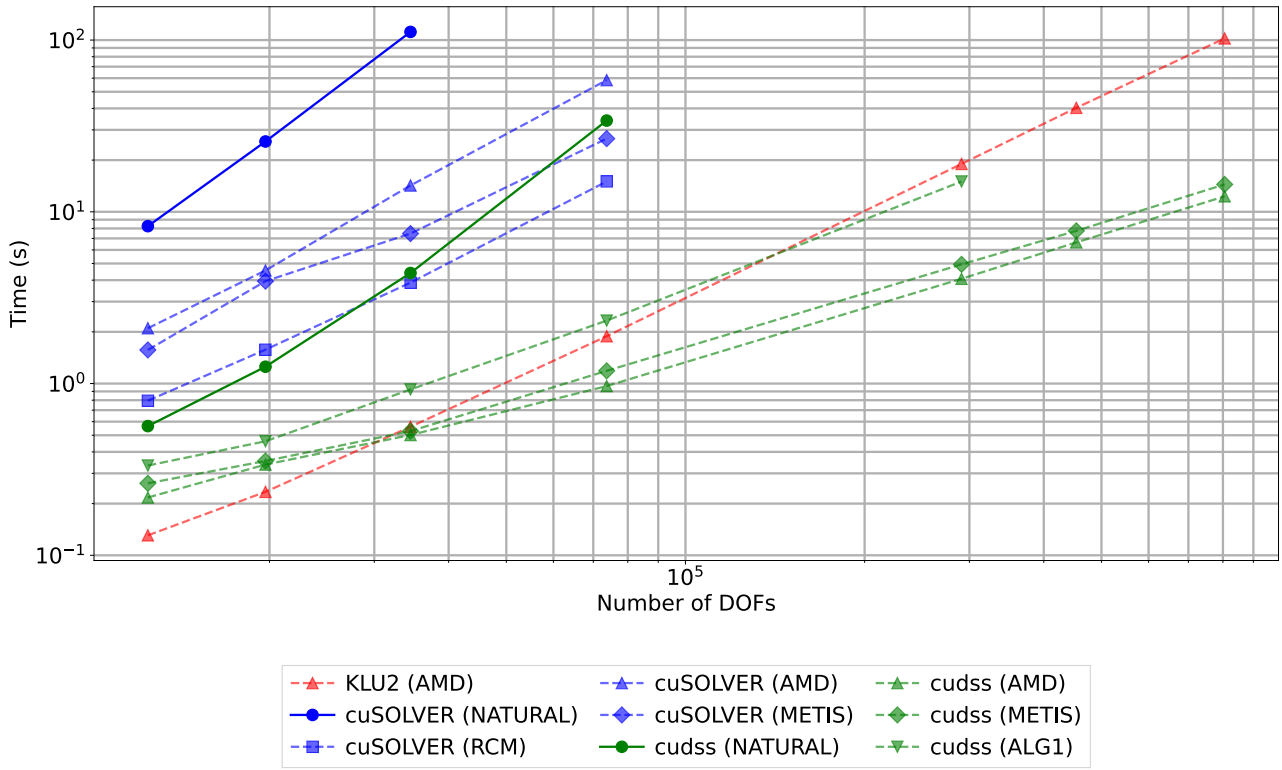


Figure 3.17: Total time to solution based reordering algorithms for degree 2 elements.

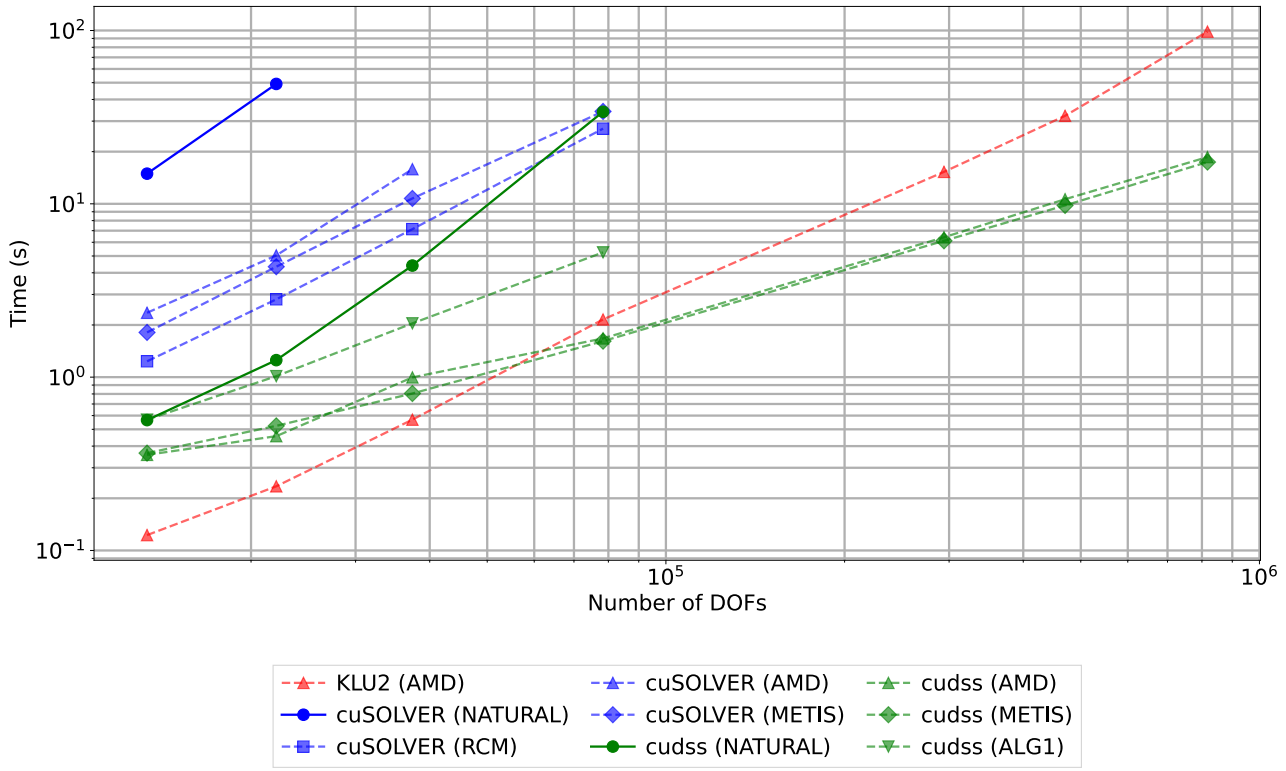


Figure 3.18: Total time to solution based reordering algorithms for degree 4 elements.

3.6 Outcomes

This chapter investigated the performance of modern GPU-accelerated sparse direct solvers, aiming to identify effective alternatives to established CPU-based methods for finite element analysis. The study was conducted within the HELM framework, utilizing the CPU-based *KLU2* solver from the *Amesos2* interface as the baseline for accuracy and serial performance.

The initial evaluation of GPU acceleration focused on the *cuSOLVER* library. For symmetric positive-definite systems from the Poisson problem, a significant limitation was identified: high memory consumption by the *Cholesky* factorization, which constrained the maximum solvable problem size. For the indefinite Helmholtz problem, the library parallel *QR* factorization was evaluated but did not offer competitive performance.

The subsequent analysis of the newer *cuDSS* library showed it to be a more memory-efficient alternative. For both its *Cholesky* (*SPD*) and *LDU* (indefinite) factorizations, *cuDSS* required less memory than *cuSOLVER*. This key advantage enabled the solution of problems with substantially more degrees of freedom on the same GPU hardware.

- **For the Poisson problem:** *cuDSS* with *Cholesky* factorization was the most effective method. The analysis of reordering algorithms showed that while SYMAMD was effective for lower-order elements, MND provided better performance for higher-order discretizations.
- **For the Helmholtz problem:** the *LDU* factorization in *cuDSS* consistently outperformed both *cuSOLVER QR* method and the CPU-based *KLU2* solver. For this class of problems, SYMAMD and MND reordering delivered competitive and often nearly identical performance.

The central finding of this chapter is that a well-configured GPU-based pipeline offers a notable performance improvement over the serial CPU method for the problems studied. The combination of *cuDSS* with an appropriate reordering algorithm achieved speedups of approximately an order of magnitude

over the *KLU2* reference solver, demonstrating its utility for accelerating these computational workflows.

A key aspect of this work is its context within the *Trilinos* framework. While the custom implementation demonstrated the capability of *cuDSS*, the library is not yet integrated into *Trilinos* standard direct solver interface, *Amesos2*. This presents a clear direction for future work: integrating the *cuDSS* library as a new solver option within *Amesos2*. Such an integration would make its performance benefits more accessible to the scientific computing community that relies on *Trilinos*.

CHAPTER 4

ITERATIVE SOLVER

Contents

While direct solvers offer a robust, "black-box" approach to solving linear systems, their computational and memory requirements can scale poorly with problem size. The fill-in generated during factorization, particularly for three-dimensional problems, can cause memory usage to grow, quickly exceeding the capacity of available hardware as illustrated in Chapter 3. For the large-scale, distributed-memory simulations targeted in this work, a different strategy is required.

In this regime, iterative solvers become the method of choice. Unlike direct solvers, which compute a solution in a finite number of steps, iterative methods begin with an initial guess and generate a sequence of approximate solutions that converge toward the exact solution [5]. This approach trades the guaranteed termination of a direct solver for lower memory footprints and superior parallel scalability, making it the cornerstone of modern, large-scale scientific computing.

4.0.1 Choice of Iterative Solver: GMRES

The choice of an appropriate iterative solver is dictated by the mathematical properties of the system matrix. As established, the discretization of the Helmholtz equation yields a matrix that is large, sparse, complex-valued, symmetric, non-Hermitian, and indefinite. This profile immediately disqualifies the well-known Conjugate Gradient (CG) algorithm, which is restricted to symmetric/Hermitian positive-definite systems.

For the general class of systems encompassing these properties, methods based on Krylov subspaces are employed. Among these, the Generalized Minimal Residual (GMRES) method is a standard and robust choice [26]. At each iteration, GMRES constructs an approximate solution from the Krylov subspace that minimizes the Euclidean norm of the residual vector. Given its robustness, GMRES is selected as the outer iterative solver for this work. The implementation is provided by the *Trilinos Belos* package [27].

4.0.2 GMRES Implementation in Trilinos

Within the *Trilinos* framework, a comprehensive suite of iterative solvers is provided by the *Belos* package [27]. *Belos*is designed as a high-level framework that orchestrates the iterative process, delegating the low-level linear algebra operations to other specialized components.

To solve a system using GMRES within this ecosystem, a user typically interacts with three key components:

1. ***Tpetra*:** This package provides the core data structures for distributed linear algebra, including sparse matrices (`Tpetra::CrsMatrix`) and dense multi-vectors (`Tpetra::MultiVector`) that can be partitioned across many MPI ranks [10].
2. ***Kokkos*/ *Kokkos Kernels*:** This is the performance-portability layer [11]. *Tpetra* uses *Kokkos* to manage its data on different architectures (CPU, GPU) and relies on `Kokkos Kernels` for optimized, hardware-specific implementations of computational kernels like the Sparse Matrix-Vector product (SpMV) and vector dot products.
3. ***Belos*:** The user configures a `Belos::SolverManager` by specifying the desired solver (e.g., GMRES), setting convergence parameters (tolerance, maximum iterations), and providing the *Tpetra* objects representing the matrix and vectors.
4. ***MueLu*:** This package provides state-of-the-art algebraic multigrid (AMG) preconditioners [28]. Operating on *Tpetra* data structures, *MueLu* builds a hierarchy of coarser problems to create a highly scalable and efficient preconditioner for iterative solvers like those in *Belos*.

When the `solve()` method is called, *Belos* executes the GMRES algorithm, making calls to *Tpetra* methods which, in turn, are executed as high-performance *Kokkos Kernels* on the target hardware.

4.1 Modernization and Update of the *MueLu* Helmholtz Implementation

An experimental implementation of a Shifted Laplacian preconditioner (detailled later) for the Helmholtz equation previously existed within the *MueLu* library. This codebase, which implements many of the techniques detailed in subsequent sections, served as the starting point for this work. However, it required significant updates to be made functional and compatible with modern high-performance computing standards.

Several critical deficiencies were identified.

- **First**, a technical hurdle was its reliance on deprecated instantiation macros, specifically `TPETRA_INST_INT_INT`. This macro explicitly hardcoded the C++ template parameters for local and global ordinals — the integer types used for indexing matrix and vector entries within a single process and across the entire distributed system, respectively — to `int`. This created a direct incompatibility with the *HELM* framework, which utilizes `long long` for global ordinals to support larger problem sizes.
- **Second**, due to a lack of maintenance, the code was no longer integrated into the *Trilinos* continuous integration and deployment (CI/CD) pipeline. Its functionality was therefore not being regularly validated against updates to the library, leading to bit-rot and its eventual removal from the automated testing suite.
- **Finally**, the existing tests were insufficient for verifying solver correctness. They were limited to superficial checks, such as asserting that the number of iterations remained below an arbitrary threshold (e.g., `num_iter < 100`). Such a test provides no guarantee that the solver actually converged to a correct solution; a solver could fail in few iterations and still pass.

This work addressed these deficiencies. The deprecated macros were removed to enable full template compatibility with modern *Tpetra* standards and the *HELM* codebase. The implementation was updated, reintegrated into the testing framework with new, robust tests that properly validate convergence based on solver status and residual norms, and its status was elevated from “experimental”.

4.2 The Generalized Minimal Residual (GMRES) Method

The Generalized Minimal Residual (GMRES) method, introduced by Saad and Schultz [26], is a powerful and widely used iterative algorithm for solving large, sparse linear systems of the form $\mathbf{Ax} = \mathbf{b}$. It belongs to the family of Krylov subspace methods and is particularly valued for its robustness, as it is designed to handle the non-symmetric and indefinite matrices that frequently arise from the discretization of physical problems, such as the Helmholtz equation.

4.2.1 Conceptual Basis of GMRES

The core idea of GMRES is to find the best possible approximate solution within a gradually expanding search space. At each iteration, m, the method constructs an orthonormal basis for the Krylov subspace of degree m, defined as:

$$\mathcal{K}_m(\mathbf{A}, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, \mathbf{Ar}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{m-1}\mathbf{r}_0\}, \tag{4.1}$$

where $\mathbf{r}_0 = \mathbf{b} - \mathbf{Ax}_0$ is the initial residual. To achieve this, GMRES employs the Arnoldi iteration to generate an orthonormal basis $\mathbf{V}_m = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$ for $\mathcal{K}_m$. The result of this process is the relation

$$\mathbf{AV}_m = \mathbf{V}_{m+1}\tilde{\mathbf{H}}_m,$$

where $\tilde{\mathbf{H}}_m$ is a small ($m+1 \times m$) upper Hessenberg matrix.

By using the Arnoldi relation, this minimization is transformed into solving a small, dense least-squares problem:

$$\mathbf{y}_m = \arg\min_{\mathbf{y}\in\mathbb{C}^m} |\beta\mathbf{e}_1 - \tilde{\mathbf{H}}_m\mathbf{y}|_2, \quad \text{where } \beta = |\mathbf{r}_0|_2. \tag{4.2}$$

This least-squares problem is solved efficiently at each iteration. A sequence of Givens rotations is applied to transform the Hessenberg matrix $\tilde{\mathbf{H}}_m$ into an upper triangular form, which is equivalent to performing an incremental QR factorization. This allows the residual norm to be updated with minimal cost at each step without explicitly computing the solution vector until convergence. The overall optimality property guarantees that the residual norm is monotonically decreasing, ensuring progress toward the solution.

While GMRES offers theoretical robustness, its practical performance for challenging problems is often insufficient. The convergence rate is highly dependent on the spectral properties of $\mathbf{A}$. For matrices with eigenvalues scattered widely in the complex plane, such as those from the Helmholtz equation, GMRES may require many iterations to converge. This presents two drawbacks: increased solution time and, more critically, high memory consumption. GMRES must store the entire basis $\mathbf{V}_m$, meaning its memory and computational costs per iteration grow linearly with m. To manage this, a restarted variant, GMRES(m), is often used, where the process is restarted every m iterations. However, restarting can lead to convergence stagnation.

For these reasons, unpreconditioned GMRES is rarely practical for the problems considered here. The standard approach is to employ preconditioning [5]. The goal is to transform the original system into an equivalent one with more favorable spectral properties. This can be done via left preconditioning:

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{4.3}$$

or right preconditioning: $\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b}$, where $\mathbf{x} = \mathbf{M}^{-1}\mathbf{y}$. In either case, the preconditioner $\mathbf{M}$ is an easily invertible approximation of $\mathbf{A}$.

An effective preconditioner clusters the eigenvalues of the preconditioned operator around 1, allowing GMRES to find an accurate solution in a small number of iterations and drastically reducing both solution time and memory footprint.

4.2.2 The Challenge of the Helmholtz Equation

While preconditioning is an essential ingredient for any effective iterative solver, for a problem such as the Helmholtz equation, a standard preconditioner is often not sufficient on its own. The Helmholtz operator presents a number of numerical challenges that challenges conventional methods:

- It is indefinite, especially at high frequencies, meaning its eigenvalues are scattered on both sides of the origin [29].
- Its eigenvalues are not easily clustered by simple preconditioners (like or Jacobi), which are typically designed for definite, “Laplacian-like” operators.
- The problem suffers from the “pollution effect,” where the numerical solution requires a much finer discretization than the physical wavelength to maintain accuracy, leading to very large linear systems [30].

4.3 Ineffectiveness of Standard Preconditioning for Helmholtz

While preconditioning is essential for the practical application of iterative solvers like GMRES, the unique challenges of the Helmholtz equation render most standard, “off-the-shelf” preconditioners

ineffective [29]. The failure is not due to a minor inefficiency but stems from a fundamental mismatch between the nature of the preconditioner and the physics of the underlying operator, a mismatch that is exacerbated as the frequency increases.

4.3.1 Approximating the Wrong Operator

Consider the matrix arising from a finite element discretization of the Helmholtz equation, which has the general form:

$$\mathbf{A} = \mathbf{K} - k^2\mathbf{M} \tag{4.4}$$

where $\mathbf{K}$ is the stiffness matrix (approximating the Laplacian operator, ∇^2) and $\mathbf{M}$ is the mass matrix. The stiffness matrix $\mathbf{K}$ is symmetric positive-definite and captures the local, elliptic part of the operator. Standard preconditioners are good at approximating $\mathbf{K}$.

Let's assume we have a standard preconditioner $\mathbf{M}_P$ such that $\mathbf{M}_P \approx \mathbf{K}$. When applied to the Helmholtz system, the preconditioned operator becomes:

$$\mathbf{M}_P^{-1}\mathbf{A} = \mathbf{M}_P^{-1}(\mathbf{K} - k^2\mathbf{M}) \approx \mathbf{K}^{-1}(\mathbf{K} - k^2\mathbf{M}) = \mathbf{I} - k^2\mathbf{K}^{-1}\mathbf{M} \tag{4.5}$$

The ideal preconditioner would cluster the eigenvalues of the preconditioned operator around 1. However, a spectral analysis of this operator reveals that the eigenvalues μ_j of the generalized eigenproblem $\mathbf{K}\mathbf{x}_j = \mu_j\mathbf{M}\mathbf{x}_j$ are all real and positive. The eigenvalues of the preconditioned operator are therefore approximately $1 - k^2/\mu_j$. As the wavenumber k increases, these eigenvalues become large negative numbers, moving far away from 1 on the real axis. An operator with such a distributed, negative spectrum is difficult for GMRES to handle, leading to an increase in iteration count.

The preconditioner is effectively approximating only the elliptic part of the operator while ignoring the increasingly dominant, indefinite part. Consequently, any effective preconditioning strategy for the Helmholtz equation must explicitly incorporate the wavenumber k to approximate the full operator, not just its Laplacian component. The limitations of standard iterative methods are demonstrated in Figure 4.1, which compares the convergence of standalone GMRES and GMRES with a basic $ILU(p = 1)$ preconditioner (introduced later Section 4.5.3). The unpreconditioned GMRES solver fails to converge within the 1000-iteration limit. Although the **ILU**-preconditioned solver eventually converges, it requires a substantial number of iterations to do so, highlighting the need for a more robust strategy.

4.4 The Shifted Laplacian Preconditioner

To overcome the scaling limitations of iterative solvers for the Helmholtz equation, another strategy is to employ a preconditioning operator based on a modified, or "Shifted", Laplace operator. This approach aims to create a preconditioner that is spectrally close to the original operator, making it effective at clustering eigenvalues, while also being significantly easier to invert, particularly with robust methods like algebraic multigrid (AMG).

The Helmholtz operator is given by:

$$\mathcal{L} = -(\Delta + k^2) \tag{4.6}$$

where Δ is the Laplacian operator and k is the wavenumber. The general form of the Shifted Laplacian preconditioner, $\mathcal{P}$, is defined by introducing a complex shift into the wavenumber term:

$$\mathcal{P} = -(\Delta + (\alpha + i\varepsilon)k^2) \tag{4.7}$$

where α and ε are real-valued shifting parameters. The selection of these parameters has been the subject of extensive research, leading to a rich body of literature that informs our approach.

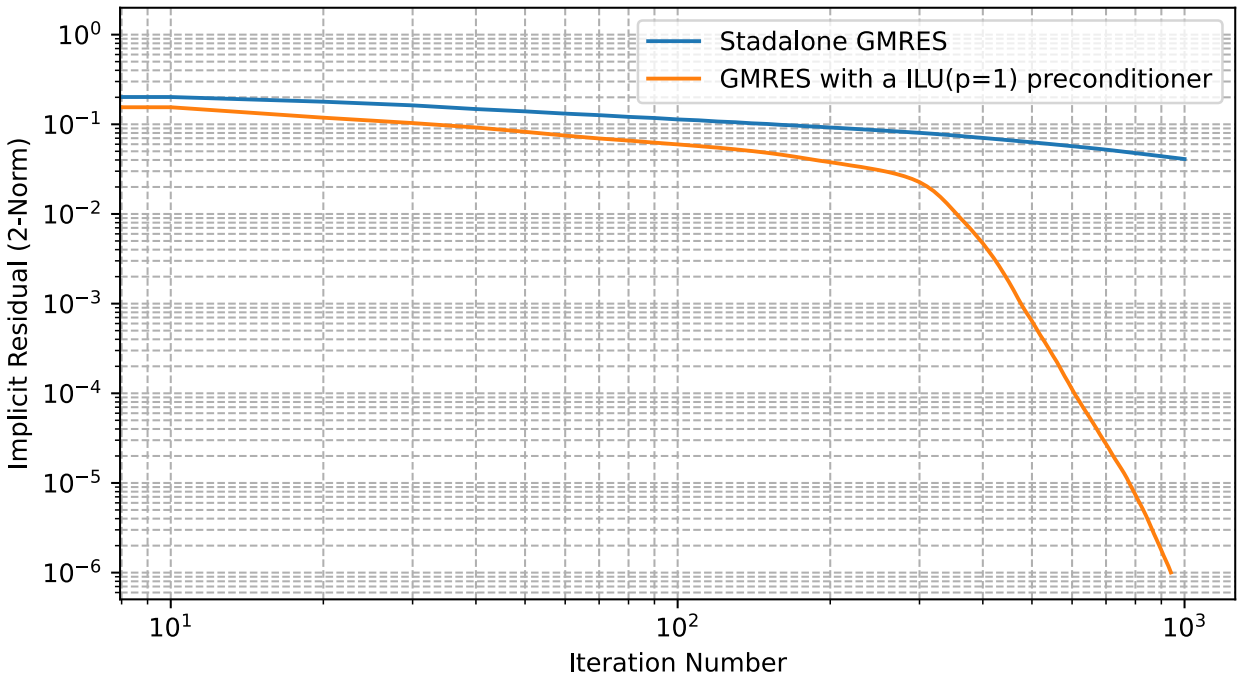


Figure 4.1: Implicit residual of the standalone GMRES algorithm and a preconditioned GMRES with a **ILU**(p=1)

4.4.1 Evolution of the Shifted Laplacian Preconditioner

The concept of using a Shifted operator as a preconditioner is well-established [31]. Early work focused on real shifts, such as using $(\Delta - k^2)$ as a preconditioner [32]. However, subsequent research demonstrated the significant benefits of introducing an imaginary component.

A major advancement came from Erlangga et al., who investigated a purely imaginary shift, effectively using $(\Delta + ik^2)$ [33, 34]. Through a 1D analysis of the preconditioned operator, they found that the ratio of the largest to smallest eigenvalues was minimized when the imaginary shift was proportional to $\pm k^2$, providing a strong theoretical motivation for this choice [33, 29].

More recent and highly successful formulations utilize a complex shift where the real part is close to the original operator ($\alpha \approx 1$) and a small imaginary part is added for stability. This leads to a preconditioner of the form $(\Delta + (1 + i\varepsilon)k^2)$. This approach has proven particularly effective when the preconditioner inverse, $\mathcal{P}^{-1}$, is approximated using a multigrid method [35, 36](multigrids will be introduced in Section 4.5.4). The imaginary term acts as a form of artificial damping, making the otherwise indefinite system amenable to multigrid techniques, which traditionally struggle with oscillatory, high-frequency error components.

4.4.2 The Critical Choice of the Imaginary Shift ε

A central theme in the literature is the optimal scaling of the imaginary part of the shift. Different studies, motivated by different goals, have arrived at different conclusions:

- **For Multigrid Convergence:** Many works leveraging multigrid to approximate $\mathcal{P}^{-1}$ have found that a shift proportional to the square of the wavenumber, $\varepsilon \sim k^2$, is necessary for the multigrid solver itself to converge robustly [35, 37, 38, 39].
- **For Eigenvalue Clustering:** In contrast, Ernst and Gander showed through a 1D finite-difference

analysis that to cluster the eigenvalues of the preconditioned system $\mathcal{P}^{-1}\mathcal{L}$ near $(1, 0)$ in the complex plane, a smaller shift of $\varepsilon \sim k$ is required [40].

- **Other Contexts:** Other variations have also been explored, such as $\varepsilon \sim k$ in the context of sweeping preconditioners [41].

This highlights a crucial trade-off: the optimal shift for making the inner multigrid solver converge is not necessarily the same as the optimal shift for making the outer GMRES solver converge in fewer iterations.

4.4.3 Limitations of Previous Analysis

As summarized by Gander and an der Heiden [42], the majority of the theoretical analyses in the literature share two common traits: they are based on a simplified 1D Helmholtz problem with Dirichlet boundary conditions, and their primary focus is on the eigenvalue distribution of the preconditioned matrix $\mathcal{P}^{-1}\mathcal{L}$.

Crucially, while eigenvalue analysis provides valuable insight, its predictive power for the convergence of iterative solvers like GMRES or BiCGStab is limited for non-Hermitian systems. The convergence rate of GMRES depends on the entire field of values of the matrix, not just its spectrum. As noted in [35], even when the original Helmholtz operator $\mathcal{L}$ is Hermitian (e.g., with Dirichlet boundary conditions), the shifted preconditioner $\mathcal{P}$ is inherently non-Hermitian due to the complex shift. Therefore, the preconditioned system is always non-Hermitian, and its eigenvalue distribution alone is insufficient to guarantee fast GMRES convergence.

Based on this review, we adopt the following strategy. To ensure the preconditioner remains a good approximation of the Helmholtz operator, we set $\alpha = 1$ and will seek values of ε to minimize the time-to-solution.

4.4.4 Achieving Wavenumber-Independent Convergence with Multigrid

A primary objective when designing preconditioners for the Helmholtz equation is to achieve a GMRES iteration count that is independent of, or grows only mildly with, the wavenumber k. The work of Gander et al. [42] provides a crucial analytical framework for understanding when this might be possible with a Shifted Laplacian preconditioner whose inverse is approximated by a multigrid method.

This setup constitutes a two-level preconditioning scheme. At the outer level, the original system $\mathbf{Ax} = \mathbf{b}$ is left-preconditioned by an approximation of the shifted operator $\mathbf{A}_\varepsilon^{-1}$. Let us denote this practical, inexact preconditioner as $\mathbf{B}_\varepsilon^{-1}$, where the inverse is computed using a multigrid. The system solved by GMRES is therefore:

$$\mathbf{B}_\varepsilon^{-1}\mathbf{A}x = \mathbf{B}_\varepsilon^{-1}\mathbf{b} \tag{4.8}$$

The performance of GMRES is known to be favorable when the preconditioned operator, $\mathbf{B}_\varepsilon^{-1}\mathbf{A}$, is clustered around 1 in the complex plane. To understand the requirements for this, we can decompose the deviation from the identity matrix as follows:

$$\mathbf{I} - \mathbf{B}_\varepsilon^{-1}\mathbf{A} = (\mathbf{I} - \mathbf{B}_\varepsilon^{-1}\mathbf{A}_\varepsilon) + (\mathbf{B}_\varepsilon^{-1}\mathbf{A}_\varepsilon)(\mathbf{I} - \mathbf{A}_\varepsilon^{-1}\mathbf{A}) \tag{4.9}$$

As shown in [42], for the norm of the left-hand side to be small, the norms of the two terms on the right-hand side must also be controlled. This decomposition naturally leads to two distinct and fundamental conditions that must be simultaneously satisfied:

Proposition 1 (P1) The ideal Shifted Laplacian operator, $\mathbf{A}_\varepsilon^{-1}$, must be a good preconditioner for the original Helmholtz operator, A. This corresponds to making the term $\|\mathbf{I} - \mathbf{A}_\varepsilon^{-1}\mathbf{A}\|$ small.

Proposition 2 (P2) The multigrid approximation, $\mathbf{B}_\varepsilon^{-1}$, must be a good approximation of the ideal preconditioner, $\mathbf{A}_\varepsilon^{-1}$. This corresponds to making the term $\|\mathbf{I} - \mathbf{B}_\varepsilon^{-1}\mathbf{A}_\varepsilon\|$ small.

Herein lies the central dilemma of this preconditioning strategy. To satisfy **Proposition 1**, one desires a small shift parameter ε. In the ideal limit where $\varepsilon \to 0$, $\mathbf{A}_\varepsilon^{-1}$ becomes the exact inverse of the Helmholtz operator, making it a perfect preconditioner. Conversely, to satisfy **Proposition 2**, the multigrid method must converge efficiently for the shifted system $\mathbf{A}_\varepsilon$. This typically requires a somewhat large value of ε to introduce enough damping to make the indefinite shifted problem tractable for multigrid.

The analysis in [42] focuses exclusively on the conditions under which **Proposition 1** holds, providing valuable insights into the spectral properties of the ideal preconditioned system. To achieve a complete understanding of the two-level method, this analysis must be combined with the conditions for **Proposition 2** to hold. A successful strategy for achieving wavenumber-independent performance must therefore find a delicate balance, choosing a shift ε that is large enough for the inner multigrid solver to be effective, yet small enough for the outer GMRES iteration count to remain low.

4.5 Constructing the Preconditioner $B_{\varepsilon,h}^{-1}$

The practical application of the Shifted Laplacian preconditioner requires the repeated solution of a linear system involving the shifted operator, $\mathbf{A}_{\varepsilon,h}$. For the discrete Helmholtz system $\mathbf{A}_h\mathbf{x}_h = \mathbf{b}_h$, the preconditioner is based on the operator:

$$\mathbf{A}_{\varepsilon,h} = -(\mathbf{K} + (\alpha + i\varepsilon)k^2\mathbf{M}) \tag{4.10}$$

where K and M are the stiffness and mass matrices, respectively, arising from the finite element discretization. During each step of an outer iterative method like GMRES, applying the preconditioner to a residual vector $\mathbf{r}_h$ necessitates the computation of $\mathbf{z}_h = \mathbf{A}_{\varepsilon,h}^{-1}\mathbf{r}_h$, which is equivalent to solving the linear system:

$$\mathbf{A}_{\varepsilon,h}\mathbf{z}_h = \mathbf{r}_h \tag{4.11}$$

The overall efficiency of the preconditioned method hinges on two competing factors:

1. The **effectiveness** of the preconditioner, determined by how well $\mathbf{A}_{\varepsilon,h}^{-1}$ improves the spectral properties of the preconditioned matrix $\mathbf{A}_{\varepsilon,h}^{-1}\mathbf{A}_h$.
2. The **cost** of computing the preconditioning step, i.e., solving the system in Equation (4.11).

For this "inner" solve, one can choose between an exact or an approximate method, a decision driven entirely by computational feasibility.

4.5.1 Exact Inversion of the Preconditioner

For problems of limited size, it is feasible to solve the preconditioner system (4.11) exactly using a direct solver, such as an LU factorization. This approach, referred to as an 'exact" preconditioner, provides the ideal theoretical benefit of $\mathbf{A}_{\varepsilon,h}^{-1}$ and isolates its effectiveness from any errors introduced by an approximate inner solve. This is a valuable tool for analytical studies, such as the initial investigation of **P1** mentioned previously, as it allows for a pure assessment of the spectral properties dictated by the choice of the shift parameters α and ε.

However, the applicability of this method is severely limited. For large-scale 2D problems, and particularly for 3D problems, the memory and computational cost associated with the dense factors of $\mathbf{A}_{\varepsilon,h}$ become prohibitive. Consequently, for practical, high-performance simulations, an exact inversion is not a viable strategy.

4.5.2 Approximate Inversion of the Preconditioner

For large-scale problems, the preconditioner system must be solved approximately. The objective is to find an operator $\mathbf{B}_{\varepsilon,h}^{-1} \approx \mathbf{A}_{\varepsilon,h}^{-1}$ such that its application is significantly cheaper than a direct solve, while still retaining most of the preconditioning power. Two prominent classes of methods for constructing such an approximation are sparse approximate inverses and multigrid methods.

4.5.3 Incomplete *LU* (ILU) Factorization Preconditioners

The core idea behind **ILU** factorization is to compute sparse approximate factors $\mathbf{L}$ and $\mathbf{U}$ such that $\mathbf{A}_{\varepsilon,h} \approx \mathbf{LU}$. The preconditioner application step, $\mathbf{z}_h = \mathbf{B}_{\varepsilon,h}^{-1}\mathbf{r}_h$, is then approximated by solving the triangular systems $\mathbf{L}\mathbf{y}_h = \mathbf{r}_h$ followed by $\mathbf{U}\mathbf{z}_h = \mathbf{y}_h$. These triangular solves are computationally much cheaper than solving the original system directly. The factorization process itself is based on Gaussian elimination, but with a rule for dropping entries that would normally be introduced (fill-in) to maintain sparsity [5].

The effectiveness of an **ILU** preconditioner depends on how well $\mathbf{LU}$ approximates $\mathbf{A}_{\varepsilon,h}$ and the sparsity of $\mathbf{L}$ and $\mathbf{U}$. Different variants of **ILU** factorization employ different dropping rules, leading to a trade-off between the quality of the approximation and the cost of computing and applying the factors.

ILU(0)

The simplest form is **ILU**(0), which allows no fill-in. During the Gaussian elimination process, any entry that would be created in a position where the original matrix $\mathbf{A}_{\varepsilon,h}$ had a zero is simply dropped. The resulting factors $\mathbf{L}$ and $\mathbf{U}$ have the same sparsity pattern as the lower and upper triangular parts of $\mathbf{A}_{\varepsilon,h}$ respectively as illustrated in Figure 4.2 As mentioned previously, for challenging problems like the Helmholtz equation at high frequencies, **ILU**(0) often provides a crude approximation of $\mathbf{A}_{\varepsilon,h}$ and leads to poor preconditioning performance.

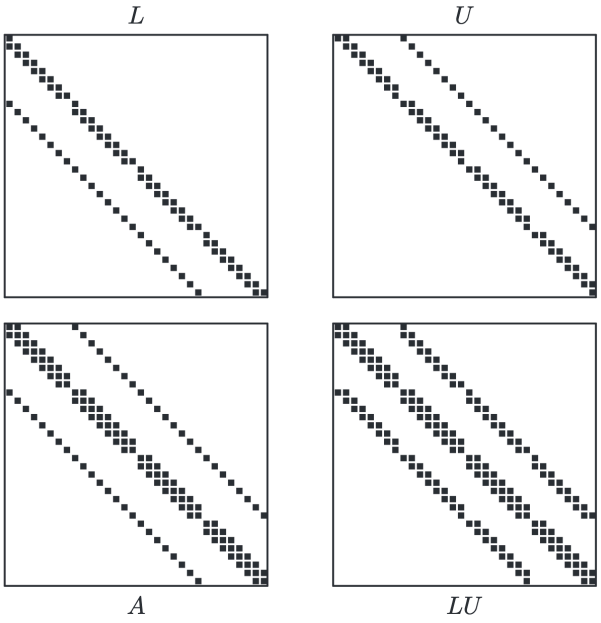


Figure 4.2: The **ILU**(0) factorization for a five-point stencil matrix. (Extracted from [5])

ILU(p)

To obtain a better approximation, the level-of-fill incomplete factorization allows a limited amount of new non-zero entries (fill-in), controlled by an integer parameter known as the level-of-fill. In this method, each potential non-zero entry is assigned a level. Original matrix entries start at level zero, and the level of any new entry created during the factorization is determined by a recursive rule. An entry is kept if its calculated level is less than or equal to the user-specified level-of-fill parameter, and dropped otherwise. Increasing this parameter allows more fill-in, resulting in denser factors, a more accurate approximation of the original matrix, and potentially fewer iterations for the outer solver, but at the cost of increased computational and memory cost. Figure 4.3 shows the resulting sparsity patterns of the factor matrices for a factorization with a level-of-fill of one. The figure illustrates how increasing the level-of-fill parameter from zero to one allows for more fill-in, leading to denser and more accurate factors.

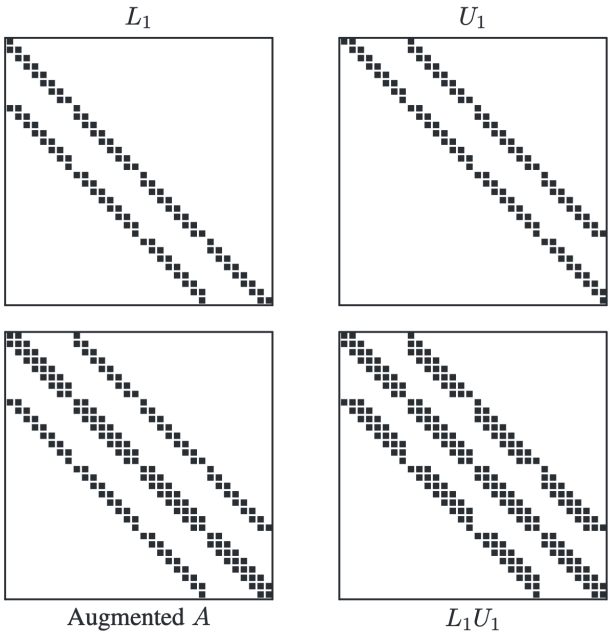


Figure 4.3: The $ILU(p = 1)$ factorization for a five-point stencil matrix. (Extracted from [5])

ILUT (ILU with Threshold)

ILUT (Incomplete LU with Threshold) offers a more flexible dropping strategy compared to **ILU**(p). Instead of relying on the sparsity pattern, **ILUT** drops entries based on their numerical magnitude relative to some measure of the row or matrix norm. This is often combined with a limit on the maximum number of entries allowed per row. The **ILUT** process involves:

1. Initializing **L** and **U** from $\mathbf{A}_{\varepsilon,h}$.
2. For each row i from 1 to n:
   (a) Perform elimination using previously computed rows $k < i$.
   (b) During or after elimination, apply a dropping rule based on a threshold τ.
   (c) Additionally, a limit on the maximum number of non-zeros per row in **L** and **U** can be applied.
   (d) Store the remaining non-zero entries in the appropriate positions in **L** and **U**.

4.5.4 Multigrid as a Preconditioner using the Shifted Laplacian

Multigrid methods represent a class of hierarchical algorithms that are among the most efficient for solving linear systems arising from PDEs [43, 44, 5]. Their power stems from addressing error components across different spatial scales by employing a hierarchy of problems on grids of varying coarseness (or, in the algebraic setting, a hierarchy of matrices of decreasing size). They are known to achieve optimal $O(N)$ complexity for many elliptic problems, where N is the number of degrees of freedom [45].

The fundamental principle of multigrid relies on the distinct effects of basic iterative methods (smoothers) on different error frequencies. Simple iterative methods, such as Jacobi or Gauss-Seidel, possess a smoothing property: they are effective at rapidly reducing high-frequency error components but are inefficient at reducing low-frequency or smooth error components. Multigrid brilliantly exploits the fact that smooth error on a fine grid appears more oscillatory on a coarser grid. By transferring the problem to coarser grids, these smooth error components can be efficiently reduced before interpolating the correction back to the fine grid.

A standard multigrid cycle, often called a V-cycle when applied recursively, consists of the following steps for a fine level h and a coarse level H:

1. **Pre-smoothing:** Apply ν_1 iterations of a smoother to the system $\mathbf{A}_h\mathbf{z}_h = \mathbf{r}_h$. This step damps the high-frequency components of the error.
2. **Compute Residual:** Calculate the residual $\mathbf{r}_h^{(1)} = \mathbf{r}_h - \mathbf{A}_h\tilde{\mathbf{z}}_h$, which corresponds to the error equation $A_h e_h = r_h^{(1)}$.
3. **Restrict Residual:** Transfer the residual to the coarse grid using a restriction operator $\mathbf{I}_h^H$, yielding $\mathbf{r}_H = \mathbf{I}_h^H\mathbf{r}_h^{(1)}$.
4. **Solve Coarse-Grid Problem:** Solve the coarse-grid error equation $\mathbf{A}_H\mathbf{e}_H = \mathbf{r}_H$. The coarse-grid matrix A_H is typically formed via the Galerkin projection: $\mathbf{A}_H = \mathbf{I}_h^H\mathbf{A}_h\mathbf{I}_H^h$, where $\mathbf{I}_H^h$ is the prolongation (interpolation) operator.
5. **Prolongate Error:** Interpolate the coarse-grid error correction back to the fine grid: $\tilde{\mathbf{e}}_h = \mathbf{I}_H^h\mathbf{e}_H$.
6. **Correct Fine-Grid Solution:** Update the fine-grid solution: $\mathbf{z}_h^{\text{corr}} = \tilde{\mathbf{z}}_h + \tilde{\mathbf{e}}_h$.
7. **Post-smoothing:** Apply ν_2 iterations of the smoother to remove high-frequency errors introduced by the prolongation step.

This process is illustrated in Figure 4.4 and can applied recursively to solve the coarse-grid problem in step 4 until a coarsest level is reached where a direct solve is feasible.

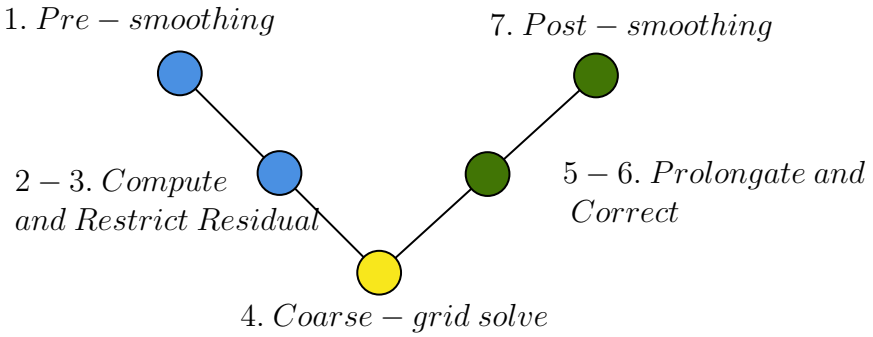


Figure 4.4: Compact multigrid V-cycle with corrected annotation spacing and pointers.

A key achievement of Algebraic Multigrid (AMG) is its ability to construct the entire hierarchy, particularly the transfer operators $\mathbf{I}_H^h$ and $\mathbf{I}_h^H$, without explicit geometric information, using only the entries

of the matrix $\mathbf{A}_h$ [46]. For this work, we employ a Smoothed Aggregation (SA) based AMG. The SA algorithm constructs the prolongator $\mathbf{I}_H^h$ in two stages. First, it partitions the fine-grid nodes into small disjoint sets called 'aggregates" by identifying nodes that are 'strongly connected" based on the magnitude of the matrix entries. These aggregates define the coarse-grid points. Second, a "tentative" prolongator based on these aggregates is smoothed to improve its approximation properties, resulting in the final operator $\mathbf{I}_H^h$.

While standard multigrid fails for the Helmholtz equation, the complex shift in $\mathbf{A}_{\varepsilon,h}$ introduces damping that makes the operator definite and more 'elliptic-like", rendering it amenable to multigrid techniques [29].

In this context, the preconditioner application, $\mathbf{z}_h = \mathbf{B}_{\varepsilon,h}^{-1}\mathbf{r}_h$, is realized by performing one or more multigrid cycles to approximately solve the system $\mathbf{A}_{\varepsilon,h}\mathbf{z}_h = \mathbf{r}_h$. This approach forms the basis of the two-level preconditioning scheme discussed in Section 4.4.4. The choice of the shift parameter ε becomes a critical balancing act: it must be large enough to ensure the multigrid solver for $\mathbf{A}_{\varepsilon,h}$ is efficient (satisfying **Proposition P2**), yet small enough that $\mathbf{A}_{\varepsilon,h}^{-1}$ remains an effective preconditioner for the original Helmholtz operator $\mathbf{A}_h$ (satisfying **Proposition P1**).

Using multigrid without the Shifted Laplacian leads not only to failure, but can even cause divergence of the method, as illustrated in Figure 4.5 where the error is amplified after each cycle.

4.6 Classical Iterative Methods as Smoothers

Before the advent of more sophisticated approaches such as incomplete factorizations or multigrid techniques, a family of fundamental iterative methods based on matrix splittings formed the cornerstone of solving sparse linear systems. While these methods are often insufficient as standalone solvers for challenging systems like the Helmholtz equation, they remain highly relevant as components within more advanced algorithms, particularly as *smoothers* in multigrid methods. The primary role of a smoother is not to solve the linear system accurately, but rather to efficiently damp high-frequency (oscillatory) components of the error [5].

Let the system matrix $\mathbf{A}$ be decomposed as

$$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}, \tag{4.12}$$

where $\mathbf{D}$ is the diagonal part of $\mathbf{A}$, $\mathbf{L}$ is the strictly lower-triangular part, and $\mathbf{U}$ is the strictly upper-triangular part. The following classical iterative methods can be derived by rearranging the system $\mathbf{Ax} = \mathbf{b}$ using this splitting.

4.6.1 Jacobi Method

The Jacobi method is the simplest iterative technique. It is derived by rearranging the system equation to solve for each component x_i in terms of the others. Starting from the splitting (4.12), the system $\mathbf{Ax} = \mathbf{b}$ can be written as:

$$(\mathbf{D} - \mathbf{L} - \mathbf{U})\mathbf{x} = \mathbf{b} \implies \mathbf{Dx} = (\mathbf{L} + \mathbf{U})\mathbf{x} + \mathbf{b}$$

This algebraic rearrangement directly inspires an iterative scheme where the next approximation, $\mathbf{x}^{(k+1)}$, is computed using values from the previous one, $\mathbf{x}^{(k)}$:

$$\mathbf{Dx}^{(k+1)} = (\mathbf{L} + \mathbf{U})\mathbf{x}^{(k)} + \mathbf{b}$$

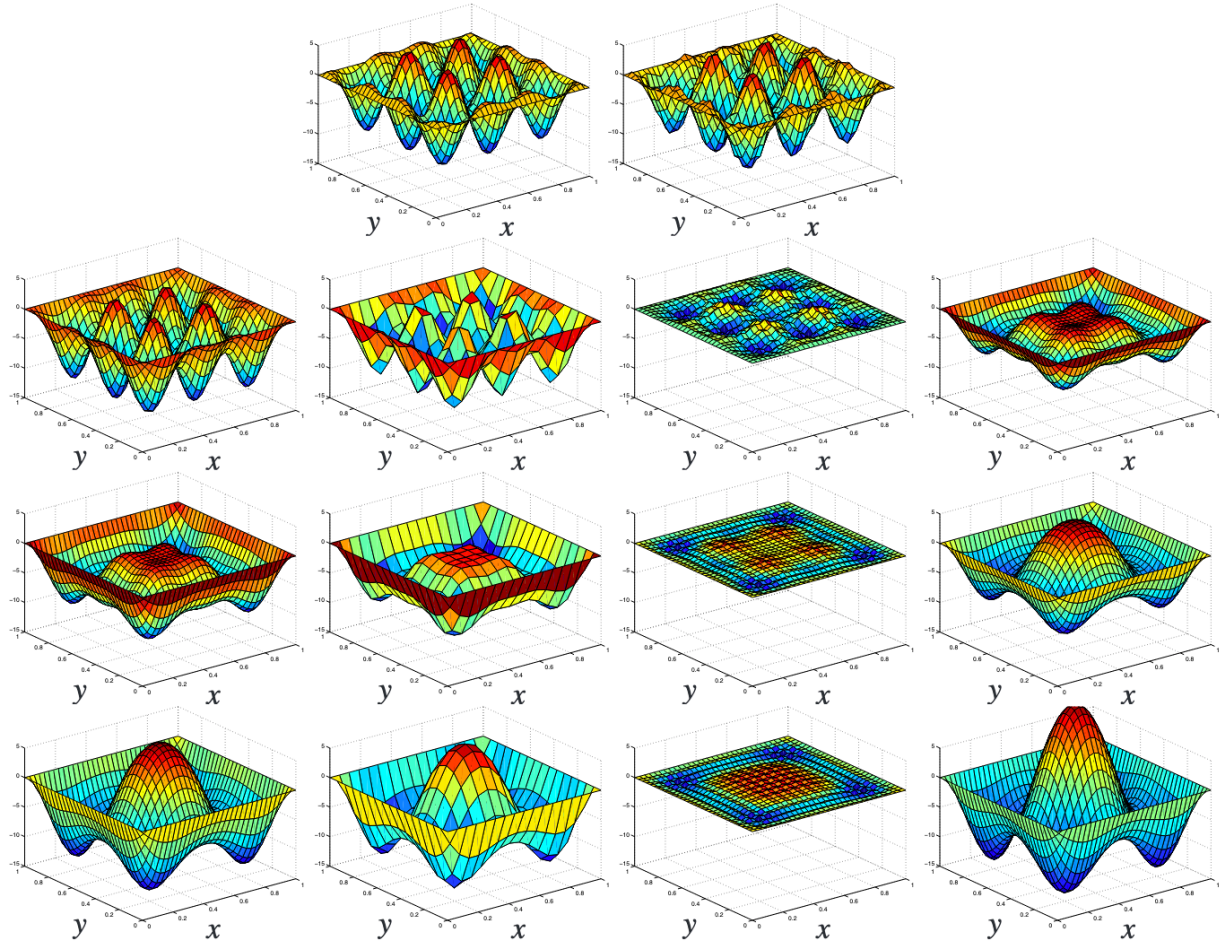


Figure 4.5:

"Failure of the smoother when multigrid is applied to the Helmholtz equation. Top row: target solution and initial error. Next three rows: error after presmoothing, coarse-grid correction to be subtracted, error after coarse-grid correction, and error after postsmoothing, shown for three consecutive iterations. The smoother clearly amplifies the error."[40]

Since $\mathbf{D}$ is a diagonal matrix, its inverse is trivial to compute. Solving for $\mathbf{x}^{(k+1)}$ gives the Jacobi iteration formula:

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}\left((\mathbf{L}+\mathbf{U})\mathbf{x}^{(k)} + \mathbf{b}\right).$$

This method updates all components of the solution vector simultaneously. In practice, a relaxation factor ω can be applied to form the *weighted Jacobi* iteration:

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^{(k)} + \omega\mathbf{D}^{-1}\big((\mathbf{L}+\mathbf{U})\mathbf{x}^{(k)} + \mathbf{b}\big).$$

As a preconditioner, the Jacobi method corresponds to using only the diagonal of the matrix, $\mathbf{P}_{\text{Jac}} = \mathbf{D}$. The method is highly parallelizable since each component update is independent, but its convergence is typically slow [5].

4.6.2 Gauss-Seidel (GS) Method

The Gauss-Seidel method improves upon Jacobi by using the most recently updated components within the same iteration. Its iteration can be expressed as

$$(\mathbf{D}-\mathbf{L})\mathbf{x}^{(k+1)} = \mathbf{U}\mathbf{x}^{(k)} + \mathbf{b}.$$

As a preconditioner, this corresponds to $\mathbf{P}_{GS} = \mathbf{D} - \mathbf{L}$. Relaxation can also be applied, yielding the *Successive Over-Relaxation (SOR)* method:

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^{(k)} + \omega\,\hat{\mathbf{x}}_{\text{GS}}^{(k+1)}.$$

This allows tuning the damping of error components, potentially accelerating convergence. The forward-substitution nature of GS introduces data dependencies between rows, making it inherently sequential and less amenable to parallelization than Jacobi [43].

4.6.3 Multi-Color Gauss-Seidel (MCGS)

To overcome the sequential limitations of Gauss-Seidel on parallel architectures, the Multi-Color Gauss-Seidel method partitions the nodes of the matrix graph into sets, or "colors", such that no two nodes of the same color are adjacent. Updates are performed in parallel for all nodes of a single color, sweeping sequentially through each color. For example, in a red-black ordering, all "red" nodes are updated simultaneously, followed by a simultaneous update of all 'black" nodes. Relaxation can be applied in the same manner as for GS:

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^{(k)} + \omega\,\hat{\mathbf{x}}_{\text{MCGS}}^{(k+1)}.$$

This restores a high degree of parallelism while retaining most of the convergence benefits of standard Gauss-Seidel [47].

4.7 The Additive Schwarz Method as a MPI-aware Smoother

Within the multigrid algorithm, the smoother is responsible for damping high-frequency error components. To perform this task efficiently on distributed-memory architectures, a parallel smoother is required. For this purpose, we employ a **Domain Decomposition (DD) method**, specifically the Additive Schwarz method [48, 5], which is designed explicitly for parallel computing.

The fundamental idea of DD is to 'divide and conquer": the global problem domain, Ω, is decomposed into a set of p smaller, potentially overlapping subdomains, $\{\Omega_i\}_{i=1}^{p}$, where each subdomain is typically assigned to a single MPI process. The original large problem is then replaced by a collection of smaller problems on these subdomains, which can be solved in parallel.

4.7.1 Mathematical Formulation

Let the global linear system on the fine grid be $\mathbf{Ax} = \mathbf{b}$. To formalize the domain decomposition, we introduce a set of restriction operators. For each subdomain Ω_i, the restriction operator $\mathbf{R}_i$ is a boolean matrix that selects the degrees of freedom belonging to that subdomain from a global vector. Applying $\mathbf{R}_i$ to the global solution vector $\mathbf{x}$ yields the local solution vector on that subdomain, $\mathbf{x}_i = \mathbf{R}_i\mathbf{x}$.

The system matrix for each subdomain, $\mathbf{A}_i$, is then defined via the Galerkin projection:

$$\mathbf{A}_i = \mathbf{R}_i\mathbf{A}\mathbf{R}_i^T$$

where $\mathbf{R}_i^T$ is the extension (or prolongation) operator that injects a local vector from subdomain Ω_i back into a global vector, padding with zeros.

The core of the Schwarz method is to approximate the inverse of the global matrix $\mathbf{A}^{-1}$ with a combination of approximate inverses of the smaller, local subdomain matrices, $\mathbf{A}_i^{-1}$. Let us denote the local subdomain solver in our case, an **ILU** factorization by $\mathbf{B}_i^{-1} \approx \mathbf{A}_i^{-1}$.

4.7.2 The Restricted Additive Schwarz Smoother

The smoother used in this work is based on the Restricted Additive Schwarz (RAS) method [49], a highly practical and widely used domain decomposition technique. When used as a smoother within multigrid, its role is to compute an approximate solution to the error equation, $\mathbf{A}\delta\mathbf{x} = \mathbf{r}$, where $\mathbf{r}$ is the current residual.

The process for one application of the smoother is as follows:

1. **Restrict Residual:** Each process i takes the global residual vector $\mathbf{r}$ and restricts it to its local overlapping subdomain: $\mathbf{r}_i = \mathbf{R}_i\mathbf{r}$. The overlap is crucial for providing correct boundary information for the local solve.
2. **Solve Local Problems (in parallel):** Each process solves its local error equation using its subdomain solver: $\delta\mathbf{x}_i = \mathbf{B}_i^{-1}\mathbf{r}_i$. This is where the local **ILU** factorization is applied.
3. **Extend and Update Restricted Corrections:** Each process extends its local correction $\delta\mathbf{x}_i$ back to a global-sized vector, but the update is restricted to only the non-overlapping part of its subdomain. This ensures that each global degree of freedom is updated by exactly one process.

Mathematically, the action of the RAS preconditioner, $\mathbf{M}_{\text{RAS}}^{-1}$, is often written with a simplified notation:

$$\delta\mathbf{x} = \mathbf{M}_{\text{RAS}}^{-1}\mathbf{r} = \sum_{i=1}^{p} \mathbf{R}_i^T\mathbf{B}_i^{-1}\mathbf{R}_i\mathbf{r} \tag{4.13}$$

While this is the standard representation, it is important to note that the extension operator $\mathbf{R}_i^T$ on the left implicitly restricts the update to the non-overlapping part of the subdomain, and is therefore not the strict transpose of the restriction operator $\mathbf{R}_i$ on the right.

One step of the smoothing iteration is then an update of the form $\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} + \delta\mathbf{x}$. The parallel nature of the method is evident, as the local solves for each subdomain i are completely independent and can be executed concurrently.

4.7.3 Summary of the Complete Preconditioning Strategy

The final solution strategy is a multi-level, nested algorithm where each component is chosen to address a specific challenge of the parallel Helmholtz problem. This hierarchical *solver-within-a-solver* approach can be understood by starting from the outermost layer and working inwards:

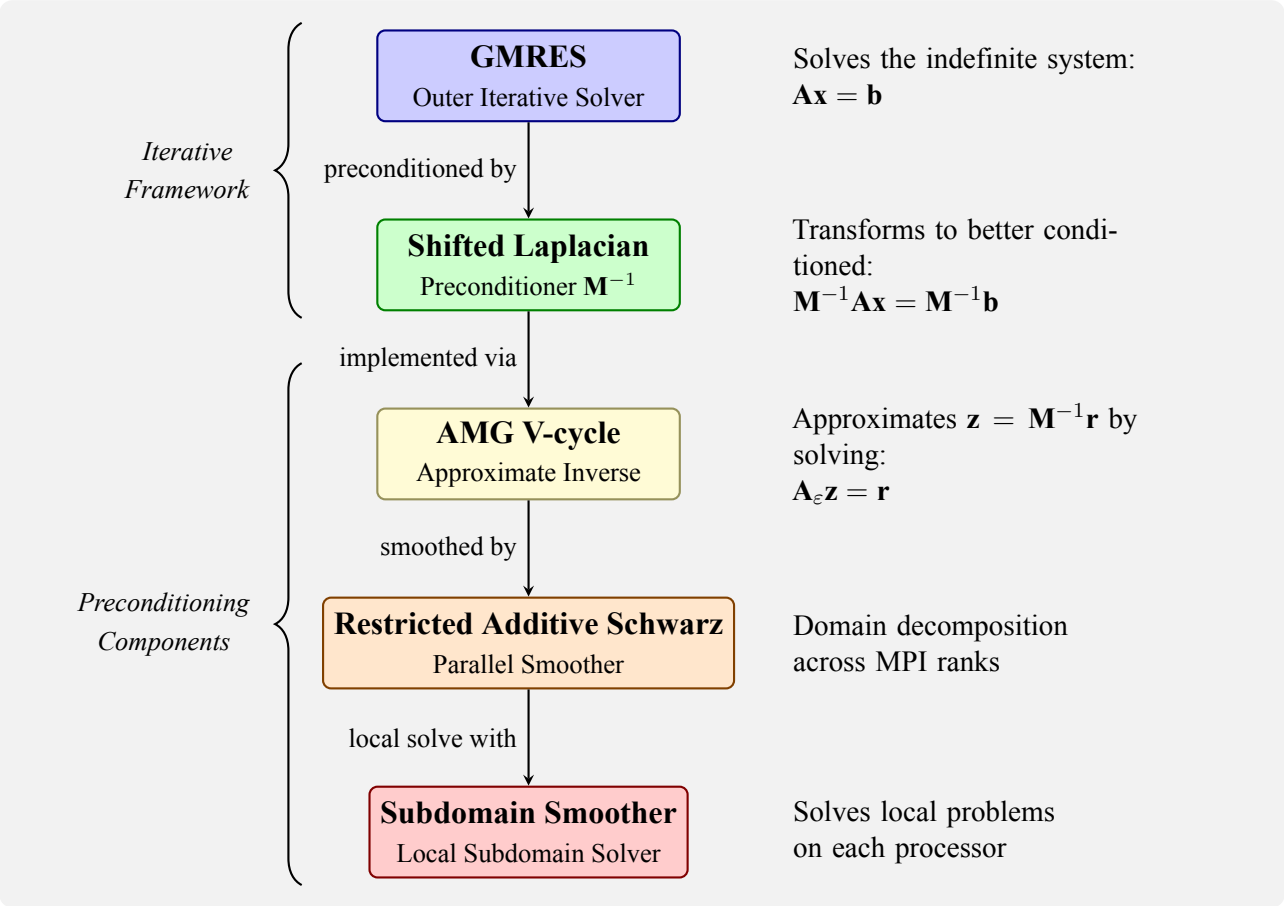


Figure 4.6: Illustration of the Nested Strategy to solve Helmholtz problem using iterative solvers.

1. **Outer Solver (GMRES):** At the highest level, the **GMRES** iterative method is used to solve the original, indefinite Helmholtz system, $\mathbf{Ax} = \mathbf{b}$.

2. **Preconditioner (Shifted Laplacian):** To accelerate convergence, GMRES is left-preconditioned by an operator $\mathbf{M}^{-1}$ based on the **Shifted Laplacian**, $\mathbf{A}_\varepsilon$. This transforms the problem into one that is more amenable to standard solution techniques.

3. **Approximate Inverse (Multigrid):** The action of this preconditioner, $\mathbf{z} = \mathbf{M}^{-1}\mathbf{r}$, is not computed exactly. Instead, it is **approximated by applying a single V-cycle of an algebraic multigrid (AMG) method** to the shifted system, $\mathbf{A}_\varepsilon\mathbf{z} = \mathbf{r}$.

4. **Parallel Smoother (Restricted Additive Schwarz):** For this multigrid V-cycle to operate efficiently on distributed-memory architectures, a parallel smoother is essential. We employ the **Restricted Additive Schwarz method** for this purpose. During each smoothing step, the problem is decomposed across MPI ranks, and each rank solves a local problem in parallel.

5. **Local Subdomain Solver:** Finally, the local problem on each subdomain within the Schwarz smoother is solved using an **smoother**.

In summary, a single iteration of the outer GMRES solver triggers a call to the multigrid V-cycle. This, in turn, involves several applications of the parallel Schwarz smoother, which itself requires a local **ILU** solve on each processor as illustrated in Figure 4.6

4.8 Parameter Search and Optimization

The effectiveness of the sophisticated multigrid preconditioner described in the previous sections is not automatic; it is highly sensitive to a wide range of algorithmic parameters. Unlike a ‘black-box” direct solver, this hierarchical method requires careful tuning to achieve optimal performance for a specific problem class like the Helmholtz equation. The large number of interacting parameters makes a systematic search essential for balancing robustness, parallel scalability, and overall solution time.

All parameter tuning experiments were conducted on a single compute node equipped with an **Intel Xeon Gold 6126 CPU** and an **NVIDIA TITAN V GPU**.

4.8.1 Tunable Parameters

The parameters controlling the solver can be grouped into three main categories: those governing the structure of the multigrid hierarchy, those defining the behavior of the smoother on each level, and those controlling the outer Krylov solver.

Table 4.1: Summary Table of the key tunable parameters.

Parameter	Category	Description / Effect
Multigrid Hierarchy		
`levels`	Multigrid	Controls depth of the V-cycle.
`coarse size`	Multigrid	Size of the coarsest grid (determines the size of the matrix for the direct solve.)
`cycles`	Multigrid	Number of cycles per application.
Additive Schwarz Smoother		
`sweeps`	Smoother	Number of pre- and post-smoothing applications.
`overlap`	Smoother	Overlap level between subdomains.
*Local **ILU** Subdomain Solver*		
`level-of-fill`	**ILU**	Fill-in level for the local **ILU** factorization.
`drop tol`	**ILU**	Drop tolerance for small entries.
Local Relaxation Subdomain Solver		
`relax val`	Relaxation	Relaxation value for Jacobi / Gauss-Seidel.
Outer GMRES Solver (Belos)		
`iterations`	Solver	Maximum GMRES iterations.
`tolerance`	Solver	Relative residual convergence tolerance.
`restart size`	Solver	Number of vectors in the Krylov subspace.

4.8.2 Parameter Tuning on a Simplified Model Problem: *Galeri* Helmholtz2D

Our initial parallel scalability tests on the unstructured Mie problem revealed a critical issue: the number of solver iterations required for convergence increased significantly as the number of MPI

ranks grew. It was unclear whether this poor scaling was caused by the simple row-by-row distribution of the unstructured matrix across processes or if it was a more fundamental limitation of the solver's algorithm itself. Furthermore, tuning the numerous parameters of the preconditioner on this complex case would be a very time-consuming process.

To isolate these factors, we adopted a simplified yet representative model problem for both parameter tuning and scaling analysis: the `Helmholtz2D` problem from the *Trilinos* package. This model discretizes the Helmholtz equation on a structured and uniform grid using a five-point finite difference stencil and features a point-like source in the center of the domain.

For a structured grid, a simple row-by-row data distribution is equivalent to a geometric domain decomposition, ensuring that each MPI rank manages a spatially local block of the problem.

By using this idealized case, we could test a key hypothesis: if the iteration count still increases with the number of MPI ranks even with this optimal data layout, then the scaling problem must be inherent to the solver's communication algorithm (the *Restricted Additive Schwarz* smoother) and not an artifact of a suboptimal matrix distribution. This simplified environment thus enabled a clear diagnosis of the underlying parallel performance limitations while also providing a rapid platform for parameter optimization.

Crucially, despite these geometric and discretization differences, the resulting system matrix shares the same fundamental mathematical properties as the one from our Mie2D problem: it is large, sparse, indefinite, complex-valued and symmetric. By performing the parameter search on this lightweight and well-controlled problem, we can efficiently identify a robust parameter configuration. This configuration then serves as a well-informed starting point for the final simulations on the more computationally intensive Mie2D benchmark cases.

4.8.3 Preconditioner Configuration for the Helmholtz2D Benchmark

The MueLu preconditioner was configured using a customized parameter set, modified from the library defaults to enhance performance for the specific challenges of the Helmholtz equation.

The default parameters provide a general-purpose starting point, but optimization for this problem led to several key changes. Notably, a shallower hierarchy (`levels = 2`) was chosen and a higher fill-level for the local **ILU** solves (`level-of-fill = 10`). The final parameters used for the analysis are summarized in Table 4.4 and will be the basis for further optimizations done in subsequent sections.

4.8.4 The Critical Role of the Imaginary Shift ε

While the previous section outlined a large parameter space for the multigrid preconditioner, one parameter stands out as uniquely critical to the success of the entire solution strategy: the imaginary shift, ε. Unlike other parameters which fine-tune algorithmic performance, the choice of ε governs the fundamental trade-off at the heart of the Shifted Laplacian method and determines whether the two-level solver converges at all.

As established in the theoretical discussion in Section 4.4.4, the performance of the preconditioned GMRES method hinges on satisfying two competing propositions simultaneously. Finding an optimal ε is therefore not a matter of simple tuning, but of enabling the convergence of the two-level method itself.

For this reason, the imaginary shift was not treated as a simple constant during the parameter search.

Table 4.2: Parameters for Helmholtz2D Analysis

Parameter	**Value**
Mesh Parameters	
`Number of elements (`N^2`)`	90000
`Wavenumber (`k`)`	20 π
`Element Size (`h`)`	0.01
`Order of Discretization (`p`)`	2
Shifted Laplacian Operator	
`alpha (`α`)`	1
`epsilon (`ϵ`)`	0.1
Multigrid Hierarchy	
`levels`	2
`coarse size`	5000
`cycles`	1
Aggregation Parameters	
`min aggregation size`	3
`max aggregation size`	8
Additive Schwarz Smoother (type 8)	
`sweeps`	1
`overlap`	0
*Local **ILU** Subdomain Solver*	
`level-of-fill`	10
`drop tol`	1e-10
Local Relaxation Subdomain Solver	
`relax val`	1.
Outer GMRES Solver (Belos)	
`iterations`	1000
`tolerance`	1e-6
`restart size`	1000

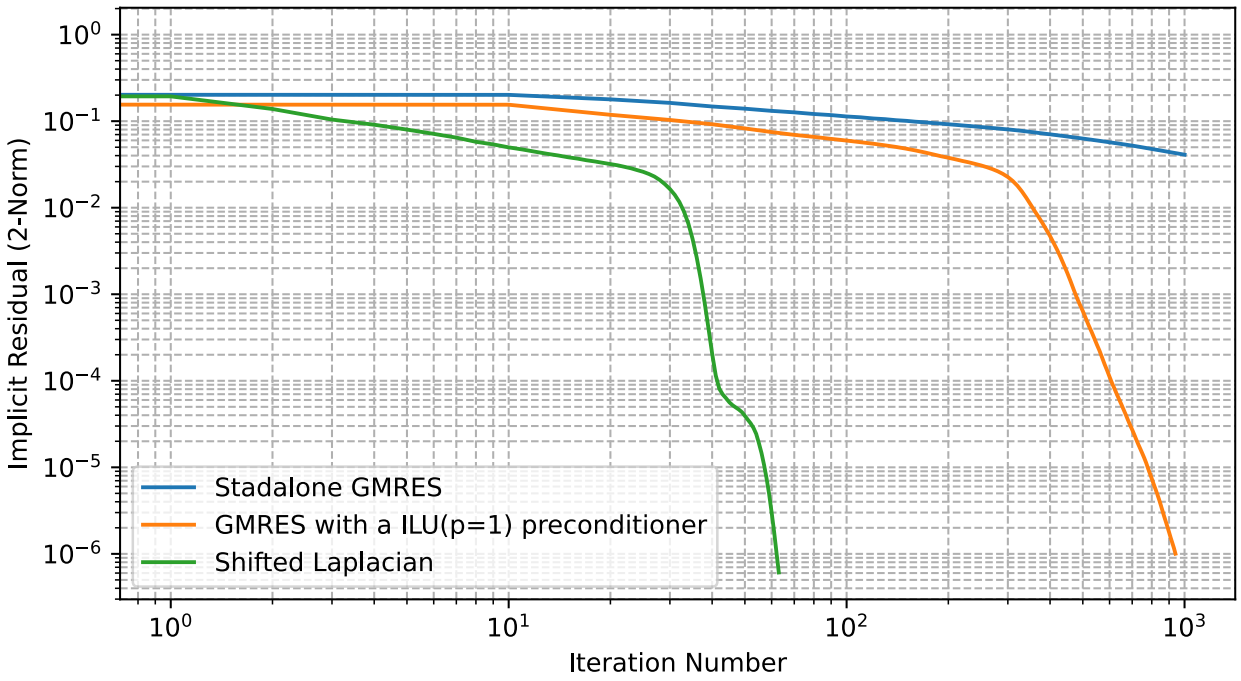


Figure 4.7: Implicit residual of the standalone GMRES algorithm, preconditioned GMRES with a **ILU**(p=1) and the Shifted Laplacian Strategy with the default parameters of Table 4.4 as well their total execution time : 251.781[s], 244.425[s] and 9.668[s] respectively.

4.8.5 Choosing ε with ILU

The effectiveness of the **ILU** smoother is critically dependent on two key tuning parameters: the level-of-fill (LOF) and the shift parameter (ε). A fundamental competition exists between them that is central to optimizing the solver performance. The trade-off directly impacts the computational cost per iteration versus the total number of iterations required for convergence:

- **Low Level-of-Fill:** Leads to a sparser, computationally cheaper smoother (faster iterations), but it is a weaker approximation of the inverse. This often requires a larger shift ε to ensure sufficient damping for the multigrid method.
- **High Level-of-Fill:** Results in a denser, more accurate smoother that requires fewer iterations, but each iteration is more computationally expensive and more memory-intensive.

This interplay is a direct manifestation of the theoretical tension between **Proposition P1** and **Proposition P2**. This relationship is quantified and visualized in Figure 4.8. For each fixed LOF, the iteration count forms a convex curve as a function of the shift. For small ε values, the smoother lacks sufficient damping, degrading multigrid performance (violating **P2**) and increasing the outer GMRES iterations. Conversely, for large ε values, the preconditioner becomes a poor approximation of the Helmholtz operator (violating **P1**), which also diminishes its effectiveness. The minimum of each curve thus identifies the optimal shift that balances these two competing effects for a given level-of-fill.

The ultimate goal is to minimize the total solution time, which is a product of the time per iteration and the number of iterations. As shown in Figure 4.8, each level-of-fill has its own optimal ε value that minimizes the number of iterations. However, the configuration that yields the fewest iterations is not necessarily the fastest overall. Therefore, to find the globally optimal performance, it is essential to test all relevant combinations of LOF and ε, as a simple analysis of iteration count alone is insufficient.

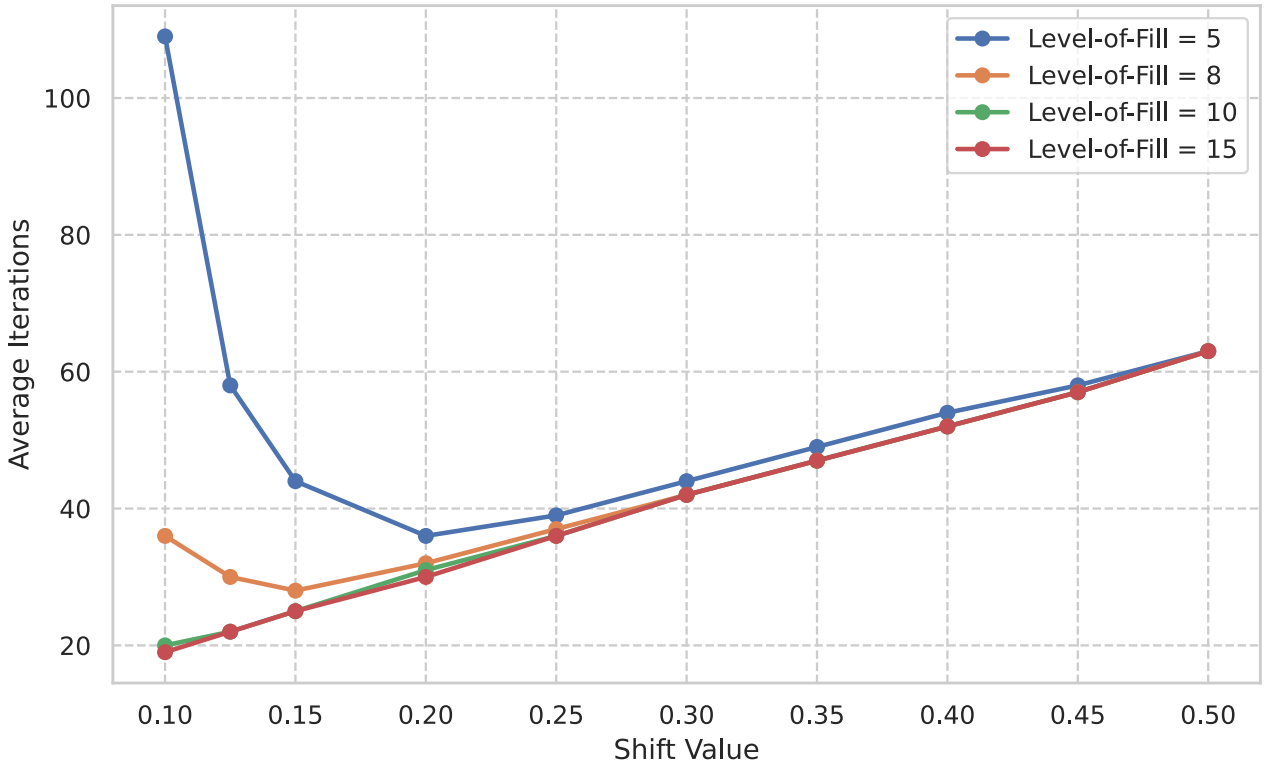


Figure 4.8: Average number of iterations values w.r.t the shift value ε for multiple level-of-fill values in the 2D Helmholtz case with parameters from Table 4.4.

4.8.6 Relaxation value in SOR

Our numerical experiments confirmed the superior convergence rate of the Gauss-Seidel (GS) smoother compared to the Jacobi method. This well-known result stems from the underlying matrix splitting: while Jacobi uses only values from the previous iteration, GS immediately incorporates the most recently updated values within the current iteration. This is equivalent to using the lower triangular part of the matrix as a preconditioner, which accelerates the propagation of corrections through the system [5]. We further optimized convergence by applying relaxation, empirically finding that both Weighted Jacobi and Successive Over-Relaxation (SOR) performed best with a relaxation parameter ω in the range of 0.6 to 0.8, as shown in Figure 4.9

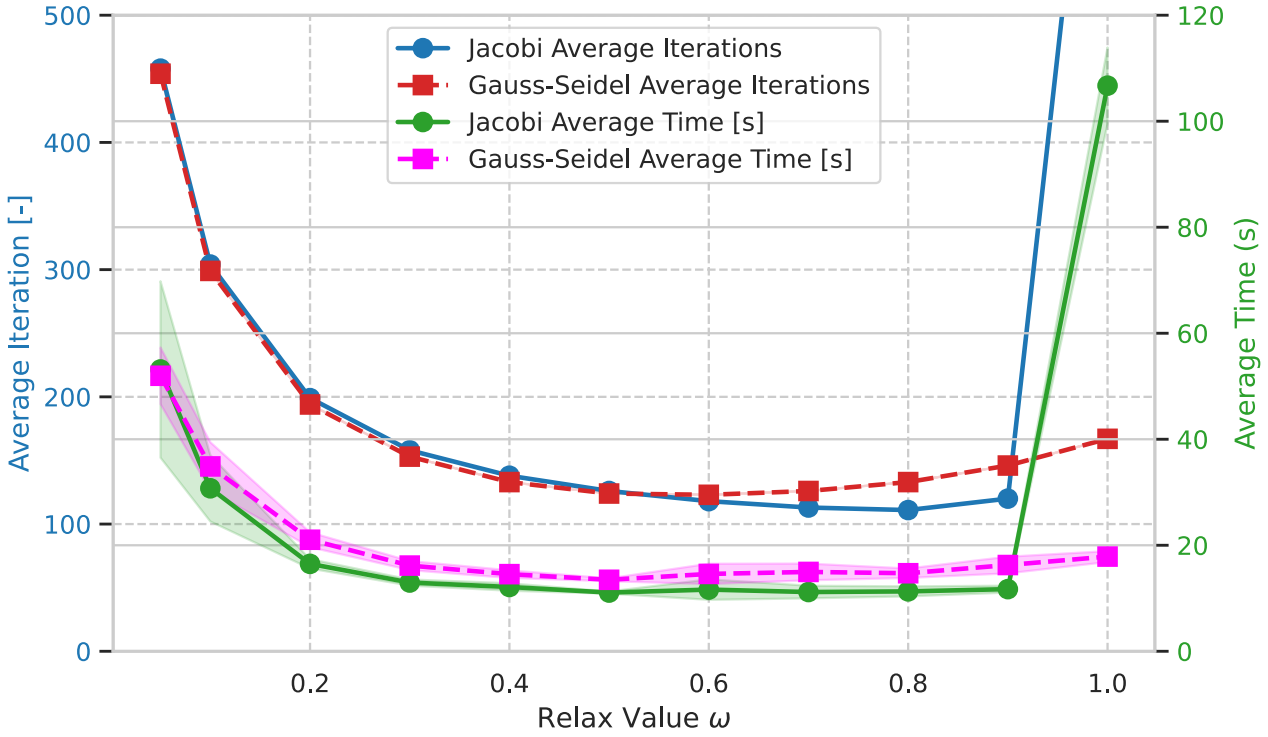


Figure 4.9: Number of iterations and time-to-solution with respect to the relaxation value ω of *Jabobi* and *Gauss-Seidel* (GS) for the Helmholtz2D test case using parameters from Table 4.4.

4.8.7 Optimizing the Number of Smoother Sweeps

Optimizing the number of smoother sweeps is crucial for the efficiency of a multigrid solver. This section investigates this trade-off by analyzing the impact of the pre- and post-smoothing sweeps, denoted by ν_1 and ν_2, on solver performance for the Helmholtz2D test case. For this analysis, the Jacobi relaxation parameter was fixed at $\omega = 0.8$, with other configuration parameters detailed in Table 4.4.

As demonstrated in Figure 4.10, a minimal number of sweeps yields the best performance. Increasing the number of sweeps from one offers no benefit; both the iteration count and the total time-to-solution increase monotonically. This behavior is a fundamental characteristic of multigrid methods, driven by the following principles:

- **Smoother Efficient Role**: The primary purpose of a relaxation-based smoother is to efficiently damp the high-frequency error components on a given grid level. A single sweep is often sufficient to accomplish this task.
- **The Cost of Oversmoothing**: Applying any sweeps beyond the minimum necessary leads to "oversmoothing." In this regime, the smoother begins to ineffectively work on the low-frequency error, a task handled far more efficiently by the coarse-grid correction mechanism. This results in wasted computation.
- **Time-to-Solution as the Metric**: The computational cost of each multigrid V-cycle is directly proportional to the total number of sweeps ($\nu_1 + \nu_2$). As the figure clearly shows, the added cost of each additional sweep is not offset by any reduction in the iteration count, leading to a direct and immediate increase in total solution time.

The analysis in Figure 4.10 confirms that a minimal number of sweeps provides the optimal balance. For this specific problem, using a single sweep ($\nu_1 = \nu_2 = 1$) yields the minimum time-to-solution,

establishing it as the most effective configuration.

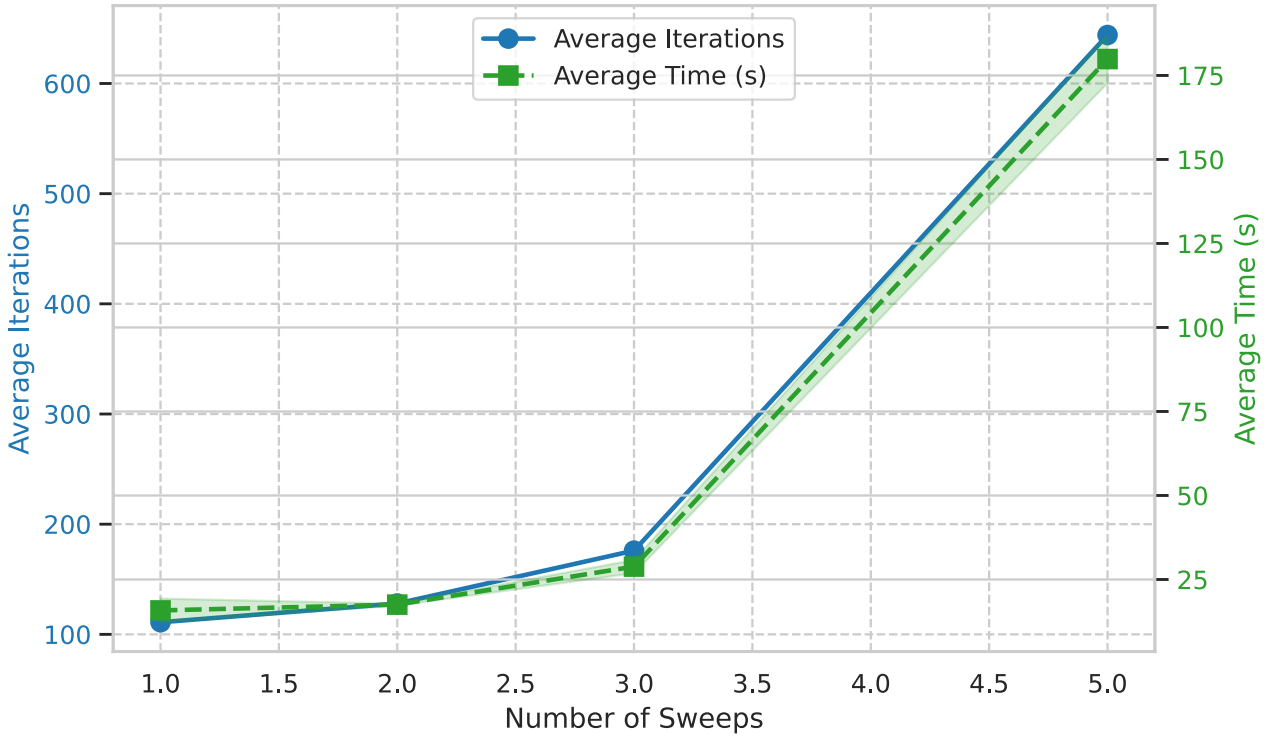


Figure 4.10: Optimal number of iterations and time-to-solution is achieved with a small number of smoother sweeps. The plot shows the total GMRES iterations and time-to-solution as a function of the number of Jacobi sweeps ($\nu = \nu_1 = \nu_2$) for the Helmholtz2D test case. The overall time increases beyond two sweeps due to the cost of oversmoothing. Test parameters are from Table 4.4 with a relaxation value of $\omega = 0.8$.

4.8.8 Level-of-Fill Analysis

The level-of-fill parameter for the **ILU** smoother proved to be a critical factor for convergence. For the Helmholtz2D problem, as illustrated in Figure 4.11, the solver failed to converge for any level-of-fill less than four. At this threshold, robust convergence was achieved. Interestingly, further increasing the level-of-fill yielded almost no reduction in the iteration count, while significantly increasing the total execution time due to the more costly factorization and application of the denser preconditioner. Consequently, a level-of-fill of five was identified as the optimal balance between robustness and computational efficiency.

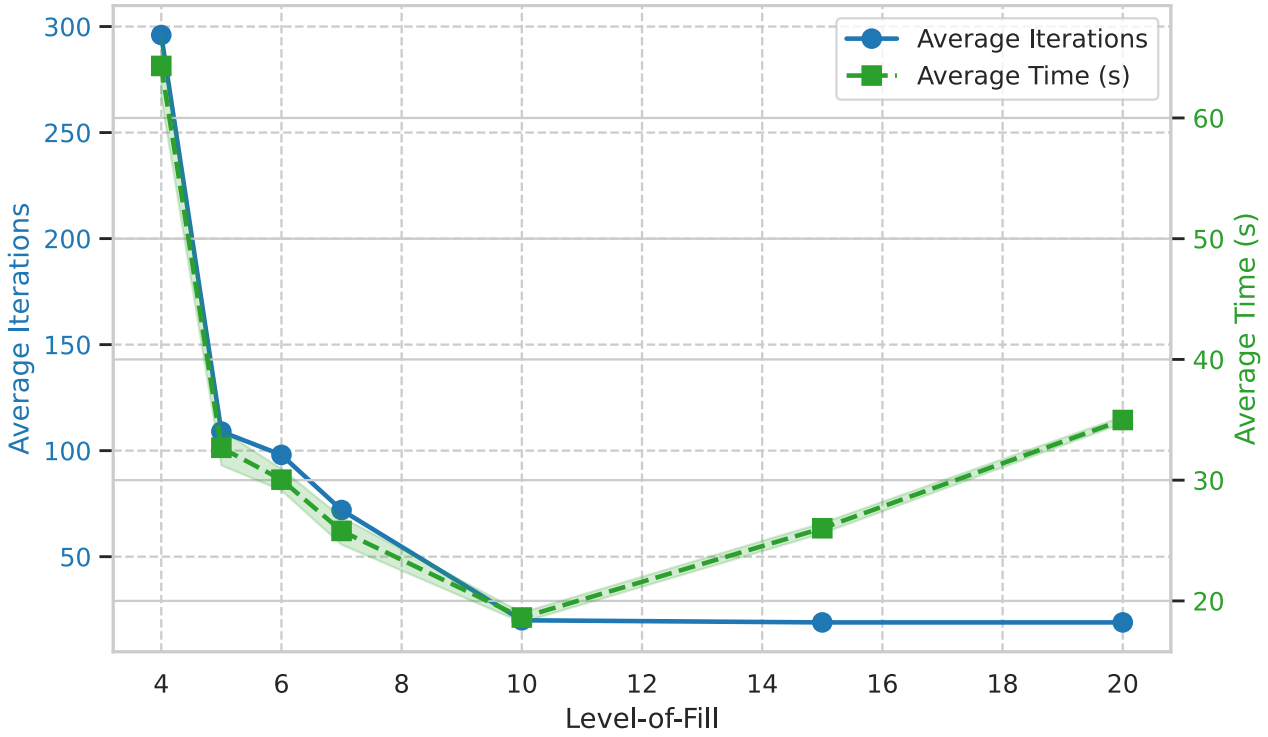


Figure 4.11: Number of iterations and time-to-solution w.r.t the level-of-fill using **ILU**(p) for the Helmholtz2D test case using Table 4.4.

4.8.9 Drop tolerance

The drop tolerance (τ) has little effect on the solver performance, as seen in Figure 4.12. The number of iterations and the time-to-solution remain largely unchanged across a wide range of drop tolerance values. However, if the drop tolerance is set too high, the preconditioner becomes too sparse and loses its effectiveness, leading to the divergence of the solver. This behavior is expected, particularly for structured problems like this Helmholtz2D test case on a regular grid, where the matrix has a very predictable sparsity pattern.

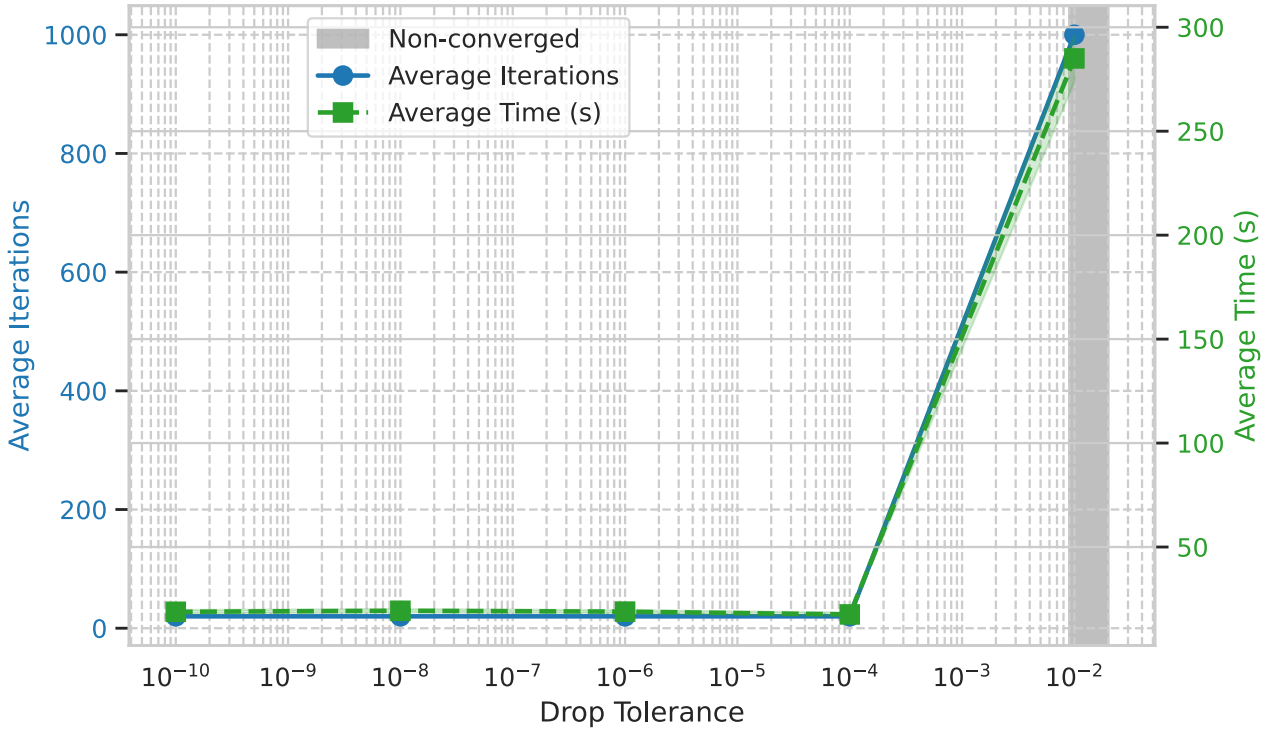


Figure 4.12: Number of iterations and time-to-solution w.r.t the drop tolerance using **ILU**(p, τ) for the Helmholtz2D test case using Table 4.4.

4.9 Optimal Parameter Configuration

Following the detailed analysis of individual solver parameters in the preceding sections, this section consolidates these findings to present the final, optimized configurations that yield the minimum time-to-solution. The optimization process began with a set of reasonable default parameters, as listed in Table 4.4, and systematically refined them through extensive numerical experiments where the total solve time was measured against an increasing number of Degrees of Freedom.

The primary goal was to find the optimal balance between the number of iterations and the computational cost per iteration. The final parameters for both the Helmholtz2D and Mie test cases are presented below.

4.9.1 Optimal Parameters for Helmholtz2D

Starting from the default parameters in Table 4.4, a series of experiments revealed that several key adjustments were necessary to achieve the fastest solve time.

The most impactful changes were:

- **Smoother Choice:** A finding was that using **ILU** subdomain solver was significantly more effective than simpler point-wise smoothers like Jacobi or Gauss-Seidel. For larger problem sizes, the Jacobi and Gauss-Seidel smoothers struggled with robustness and often failed to converge, establishing the **ILU**-based smoother as the only viable option.
- **Shift Parameter (ϵ):** Increased from 0.1 to 0.2. This provided a better balance between preconditioner accuracy (**Proposition P1**) and multigrid damping (**Proposition P2**), ultimately reducing the number of outer GMRES iterations.

- **ILU Level-of-Fill:** Reduced from 10 to 5. While a lower level-of-fill creates a "weaker" **ILU** smoother (potentially increasing iterations), the computational savings from computing and applying a sparser preconditioner in each sweep resulted in a significant net reduction in total time-to-solution.
- **GMRES Parameters:** The maximum iterations and restart size were reduced from 1000 to 100. This adjustment was made primarily for memory efficiency. The restart size dictates the dimension of the Krylov subspace basis stored in memory; reducing it from 1000 to 100 lowers the memory footprint of the outer solver.

The resulting optimal parameter set for the Helmholtz2D problem is detailed in Table 4.3.

Table 4.3: Improved Parameter Set for the Helmholtz2D Problem base on Table 4.2

Parameter	**Value**
Shifted Laplacian Operator	
`epsilon` (ϵ)	0.2
*Local **ILU** Subdomain Solver*	
`level-of-fill`	5
Outer GMRES Solver (Belos)	
`iterations`	100
`restart size`	100

4.9.2 Optimal Parameters for Mie2D

In contrast to the homogeneous case, the Mie scattering problem, which features heterogeneous material properties, proved to be significantly more challenging to solve. As with the Helmholtz2D case, simpler smoothers like Jacobi or Gauss-Seidel were insufficient, and the more robust Additive Schwarz smoother with a local **ILU** solver was necessary for convergence. However, the tuning of this solver required a distinct and more aggressive set of parameters which is expected since the problem is heterogenous and unstructered comparered to Helmholtz2D.

Key differences from the Helmholtz2D configuration include:

- **Shift Parameter (ϵ):** Increased substantially from 0.2 to 0.7. A much larger shift was required to provide sufficient damping and ensure the preconditioner remained effective, indicating the increased indefiniteness of the Mie problem.
- **ILU Level-of-Fill:** Increased from 5 to 7. A denser, more accurate **ILU** smoother was necessary to control the error, reinforcing the conclusion that the heterogeneous problem is numerically harder.
- **GMRES Parameters:** The maximum iterations and restart size were reverted to 1000. The number of iterations grew past 100 which lead to stop before convergence, as such, that change was reverted.

The final optimized parameter set for the Mie2D problem is detailed in Table 4.4.

4.10 Performance Comparison and Scaling Analysis

To benchmark the optimized multigrid preconditioner, its performance and asymptotic complexity are compared against a sparse direct solver (KLU2) on a single CPU rank. The analysis is performed

Table 4.4: Optimized Parameter Set for the Mie2D Problem

Parameter	Value
Shifted Laplacian Operator	
`epsilon` (ϵ)	0.7
*Local **ILU** Subdomain Solver*	
`level-of-fill`	7

under a fixed resolution of 20 degrees of freedom per wavelength, corresponding to a constant product of the wavenumber and mesh size ($k \cdot h \approx \pi/10$). This regime is standard for ensuring the pollution effect is controlled and that the problem difficulty scales consistently with the number of unknowns, N. The multigrid solver uses the optimized parameters identified in Table 4.3.

The results for a single MPI rank and a single OpenMP thread are plotted in Figure 4.13 and offer two insights. First, in terms of absolute time-to-solution, the direct solver is faster for the range of 2D problems tested. This is expected, as the overhead of the multigrid setup and V-cycles can be significant for moderately sized problems.

The second and more important finding comes from the scaling analysis. The empirical complexity of the KLU2 solver was measured to be approximately $O(N^{1.55})$, consistent with the theoretical performance of sparse direct methods on 2D structured problems. In contrast, our optimized shifted Laplacian multigrid preconditioner demonstrates a significantly better, though not perfectly linear, asymptotic complexity of approximately $O(N^{1.20})$.

This deviation from ideal linear scaling is an expected and understood outcome of the shifted Laplacian method. As discussed in Section 4.4.4, perfect wavenumber-independent convergence (implying a constant iteration count) is difficult to achieve in practice due to the trade-offs between the competing requirements outlined in **Proposition 1** and **Proposition 2**. Consequently, as the problem size N and wavenumber k increase, a mild growth in the number of GMRES iterations is observed, leading to the overall $O(N^{1.20})$ complexity.

Despite this, the superior asymptotic scaling of the multigrid approach is a clear validation of its effectiveness. It demonstrates that the preconditioner is successfully controlling the growth in iteration count, keeping it far below the rate that would be seen with a less effective method. While the direct solver is faster in this specific regime, the significant gap in their scaling exponents strongly suggests that for larger problems, the multigrid method will become the more efficient solver. This result suggests that the multigrid strategy is a viable and highly scalable path forward for tackling large-scale 3D and parallel problems, where the super-linear complexity of direct methods becomes computationally intractable.

4.11 Perspectives on Parallel Scalability and Future Work

This section provides a perspective on the challenges of such a parallel implementation and outlines a promising direction for future research, which lies beyond the scope of the current work.

4.11.1 Limitations of Restricted Additive Schwarz for Helmholtz Problems

Our current framework utilizes the Restricted Additive Schwarz (RAS) method, a standard one-level domain decomposition preconditioner. While effective for purely elliptic problems, RAS exhibits fundamental limitations when applied to the Helmholtz equation [50]. Its poor scalability stems from

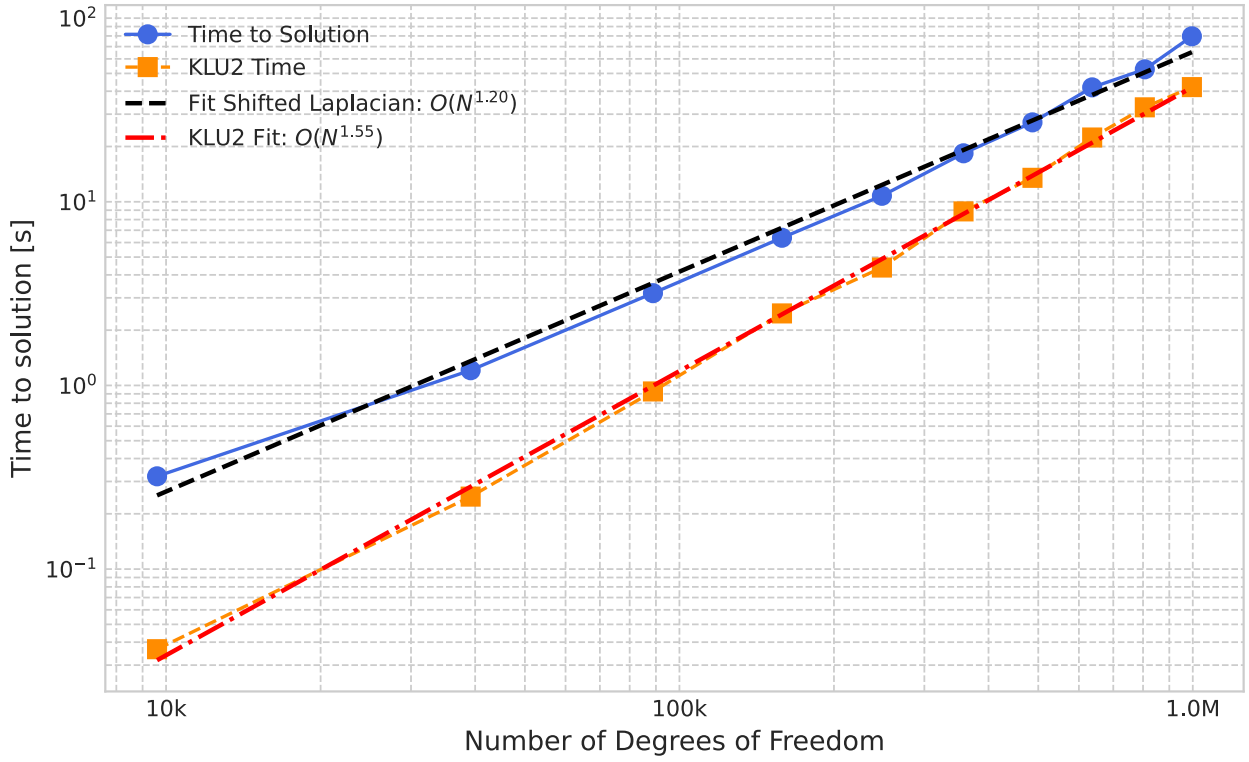


Figure 4.13: Superior asymptotic scaling of the multigrid preconditioner. While the KLU2 direct solver is faster in absolute time for these problem sizes, its $O(N^{1.55})$ complexity is significantly outperformed by the near-linear $O(N^{1.20})$ scaling of the Shifted Laplacian method, demonstrating its better suitability for large-scale problems. The parameters are taken from Table 4.3.

the use of simple Dirichlet transmission conditions at the artificial interfaces between subdomains. For wave propagation problems, these Dirichlet conditions are highly reflective; they do not allow waves to propagate transparently across subdomain boundaries. Instead, waves reflect off these artificial interfaces, pollute the local solutions, and severely hinder the convergence of the global solver. This issue is well-documented in the literature on domain decomposition methods for wave problems [51]. The result is a prohibitive growth in the number of iterations as the number of subdomains (CPUs) or the frequency increases as illustrated in Figure 4.14.

4.11.2 A Path Forward: Optimized Schwarz Methods

To overcome these limitations, a more advanced domain decomposition technique is required. Optimized Schwarz Methods (OSM) [51] have emerged as the state-of-the-art approach for Helmholtz problems. Unlike standard Schwarz methods, OSM replaces the reflective Dirichlet conditions with more sophisticated, physically-motivated impedance conditions (e.g., Robin or higher-order conditions). These optimized conditions are designed to be absorbing, minimizing reflections at the subdomain interfaces and allowing waves to pass through them transparently.

A particularly promising approach is detailed in the work of Gander, Magoulès, and Nataf, who developed Optimized Schwarz Methods without overlap [52]. By incorporating the physics of wave propagation directly into the transmission conditions, their method achieves excellent convergence rates even without the need for overlapping subdomains, which simplifies implementation and reduces communication overhead.

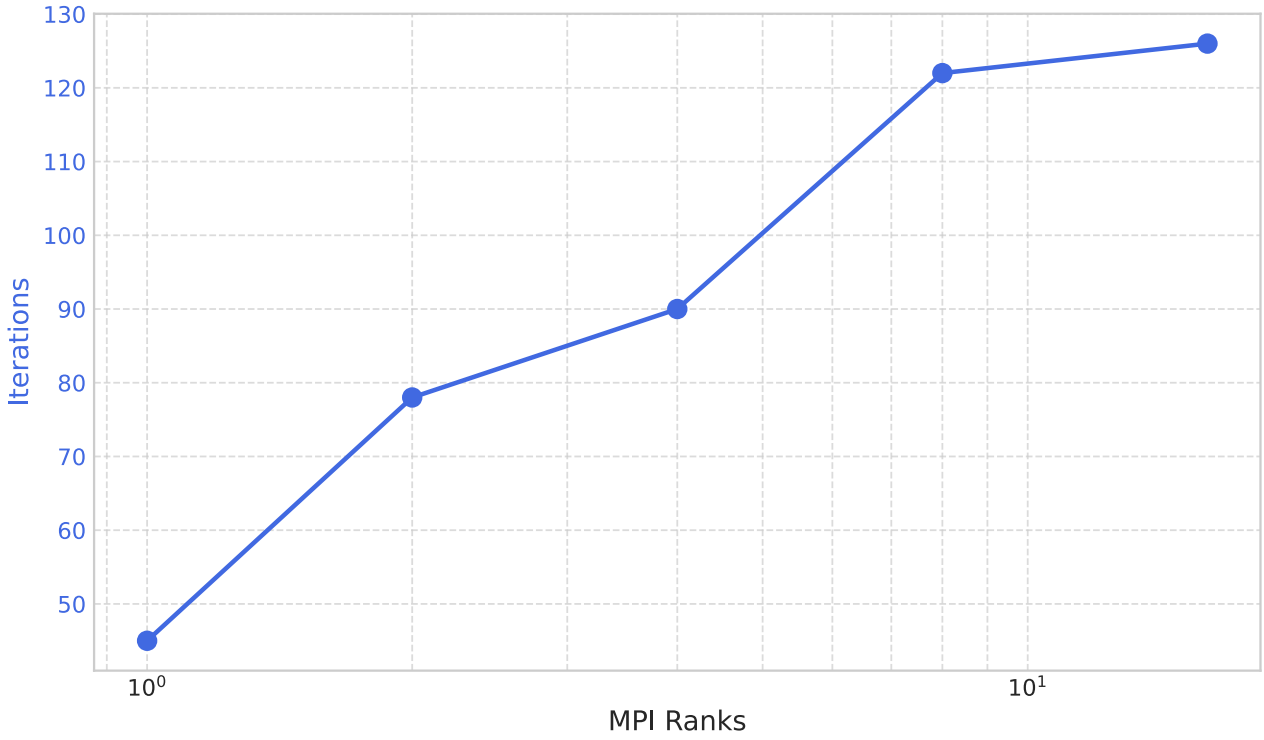


Figure 4.14: Increase of the number of iterations with respect to the number of MPI rank.

4.11.3 Future Direction

In summary, transitioning from a standard RAS preconditioner to a non-overlapping Optimized Schwarz method is identified as the critical path toward achieving true parallel scalability for the Helmholtz solver. The implementation and evaluation of such an advanced domain decomposition method are beyond the scope of the present work but constitute a clear and promising direction for future research.

4.12 GPU Acceleration and Implementation Challenges

The framework supports execution on GPU by leveraging the *Kokkos* performance-portability layer. Achieving GPU acceleration, however, required a multi-step process involving careful parameter selection, debugging of the underlying libraries, and performance analysis.

4.12.1 Smoother Configuration for GPU Execution.

The initial step was to configure the solver to use GPU-native components.

- **Smoother Choice:** The choice of smoother is critical, as not all algorithms are GPU-enabled. For this work, we utilized the `"RILUK"` smoother from the *Trilinos/Ifpack2* library. This smoother is an implementation of a level-of-fill incomplete factorization, **ILU**(k), controlled by the `"fact: iluk level-of-fill"` parameter [53], analogous to the concept shown in Figure 4.3.
- **Triangular Solve Configuration:** To ensure the computationally intensive triangular solves within the `RILUK` smoother were performed on the device, the `"trisolver:type"` parameter was configured to use Kokkos-native kernels like `"KSPTRSV"` and `"KKSPILUK"`.

4.12.2 Debugging the Kernel Dispatch Mechanism.

Initial verification revealed a significant implementation issue that prevented GPU offloading.

- **Problem Identification:** Verification using the *NVIDIA* Nsight Systems profiler showed that despite the parameter settings, the underlying Cuda libraries remained unused. This indicated that the triangular solves were still executing on the CPU.
- **Cause Analysis:** A dive into the *MueLu*/*Ifpack2* source code uncovered an issue in the kernel dispatch logic. A conditional statement was checking the high-level C++ type `scalar_t` (`std::complex<double>`) to select the compute kernel. However, the underlying on-node *Kokkos* implementation uses a custom scalar type (`Kokkos::complex<double>`). The reason is that `std::complex<double>` are not annotated as `__device__`, hence cannot be used on device. The check was therefore failing for complex data types, causing the system to fall back to a CPU-based implementation [54].
- **Implemented Solution:** We corrected in *Trilinos* the dispatch logic to check the `impl_scalar_t` instead. This fix allows the system to correctly identify GPU native kernel. This change ensures the smoother is properly offloaded, enabling a more complete device-resident execution. The successful offload of other key operations, like the Sparse Matrix-Vector product (SpMV), to *NVIDIA cuSPARSE* library (via *Kokkos*Kernels) was also confirmed.

4.12.3 Analysis of the Hybrid CPU/GPU Workflow

The *NVIDIA* Nsight Systems timeline, shown in Figure 4.15, clearly illustrates the hybrid CPU/GPU execution model of the solver during a single preconditioner application. The profiler trace is divided into two sections: the top rows show Cuda API activity, confirming work performed on the GPU, while the lower rows display NVTX ranges that trace the application C++ function calls on the CPU.

The execution trace reveals a clear sequence of operations. The process begins on the CPU with the orthogonalization phase, a core part of the GMRES algorithm managed by *Belos*. The solver then calls the *MueLu* preconditioner `Apply` method, which is the most time-consuming part of the iteration (labeled `Prec * x`).

Within this preconditioner application, significant GPU activity is visible in the Cuda API rows. The nested NVTX ranges clearly expose the structure of the multigrid V-cycle, descending through the hierarchy (*level* $2 \rightarrow$ *level* $1 \rightarrow$ *level* 0) and back up. The GPU activity confirms that the smoothing, restriction, and prolongation operations are successfully offloaded. However, the timeline also shows distinct phases with no GPU activity, such as the initial orthogonalization and the direct solve on the coarsest level (textitlevel 2). This alternation between CPU-bound and GPU-bound tasks highlights the remaining bottlenecks preventing a fully GPU-resident workflow.

Two key operations remain on the host CPU, creating a hybrid workflow:

- **GMRES Orthogonalization:** The orthogonalization step within the iterative GMRES solver, managed by the *Belos* package, is also executed on the CPU but the SpMV/GeMV are performed on the GPU as in Figure 4.18.
- **Coarse-Grid Solve:** The direct solve on the coarsest level of the multigrid hierarchy is performed by the KLU2 solver (See Figure 4.16) via the Amesos2 interface. Currently, *Amesos2* does not provide a path to GPU-accelerated direct solvers like *cuDSS*, which necessitates transferring the coarse-grid system to the CPU for this step as shown in Figure 4.17. However, given that the size of the coarse solve is small compared to the original problem, the impact of this is limited. Moreover, only the residual undergoes a memory transfer since the coarse solve matrix remains on CPU.

Consequently, the overall algorithm operates with the entire multigrid V-cycle running efficiently on the GPU, while the coarse solve and GMRES orthogonalization act as the final bottlenecks to a fully device-resident solution.

Despite the increase of iterations with MPI ranks, the solver's design, built upon the MPI+X parallelism of Trilinos, allows for multi-GPU execution and converge to the correct solution as well.

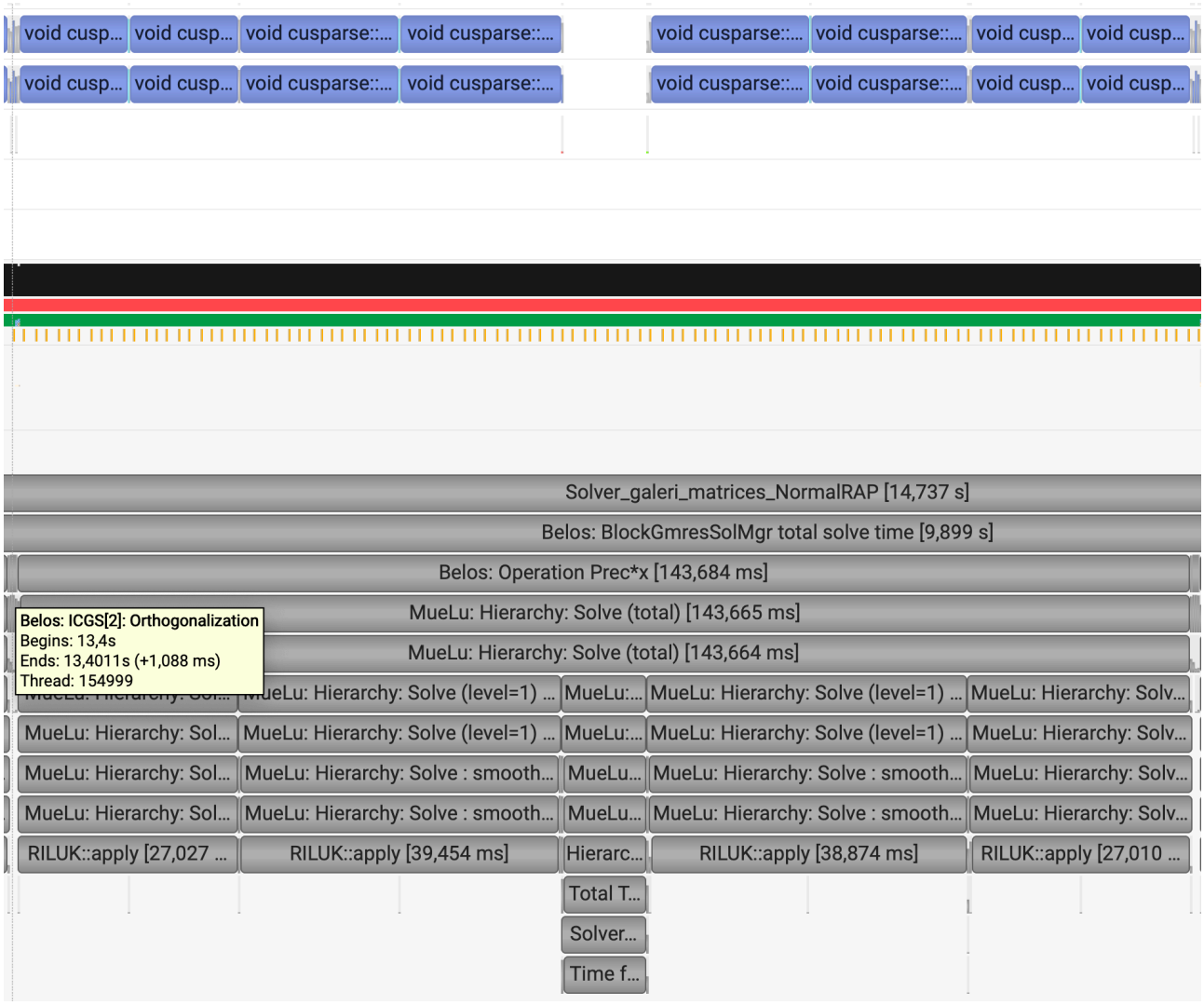


Figure 4.15: A profile of a single preconditioner application captured with *NVIDIA* Nsight Systems. The solution of triangular systems dominates the solution time.

4.12.4 Performance Results and Conclusion.

The performance of the GPU-accelerated solver was evaluated against the CPU implementation, with the results presented in Figure 4.19.

- **Performance Outcome:** Despite the successful GPU execution of the multigrid cycles, the overall time-to-solution was longer than the CPU version for the problem scales tested. This is a common outcome when the computational workload is insufficient to overcome the overheads of CPU-GPU data transfers and kernel launches [55].
- **Scalability Limitation:** The massive parallelism of a GPU is most beneficial for large problems. To achieve a performance advantage, a substantial increase in problem size is necessary.

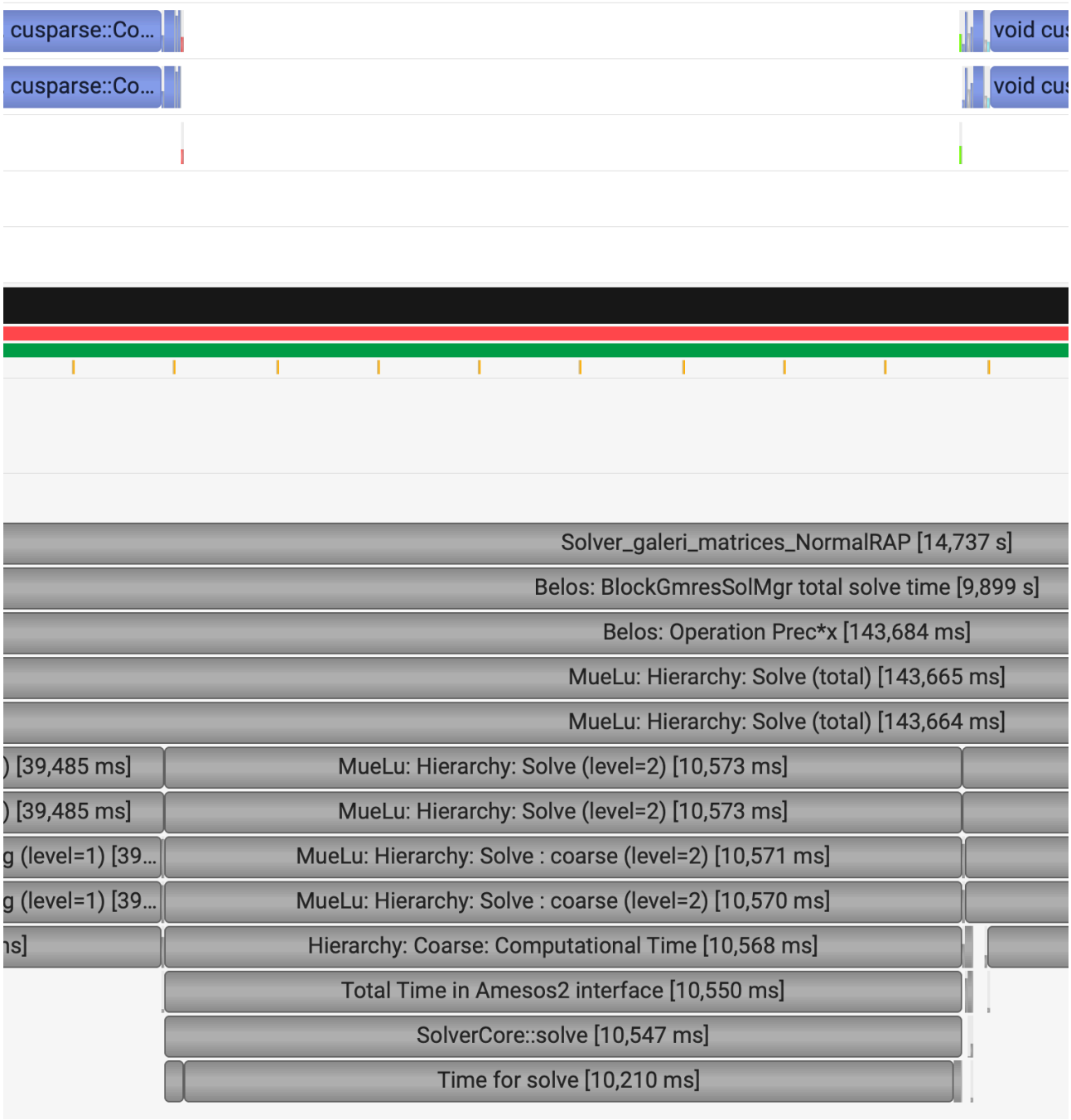


Figure 4.16: Profile of the coarse-grid direct solve (level 2 of the multigrid hierarchy) in *NVIDIA* Nsight Systems. The absence of any activity in the Cuda API rows during this phase confirms that the direct solve is executed entirely on the CPU.

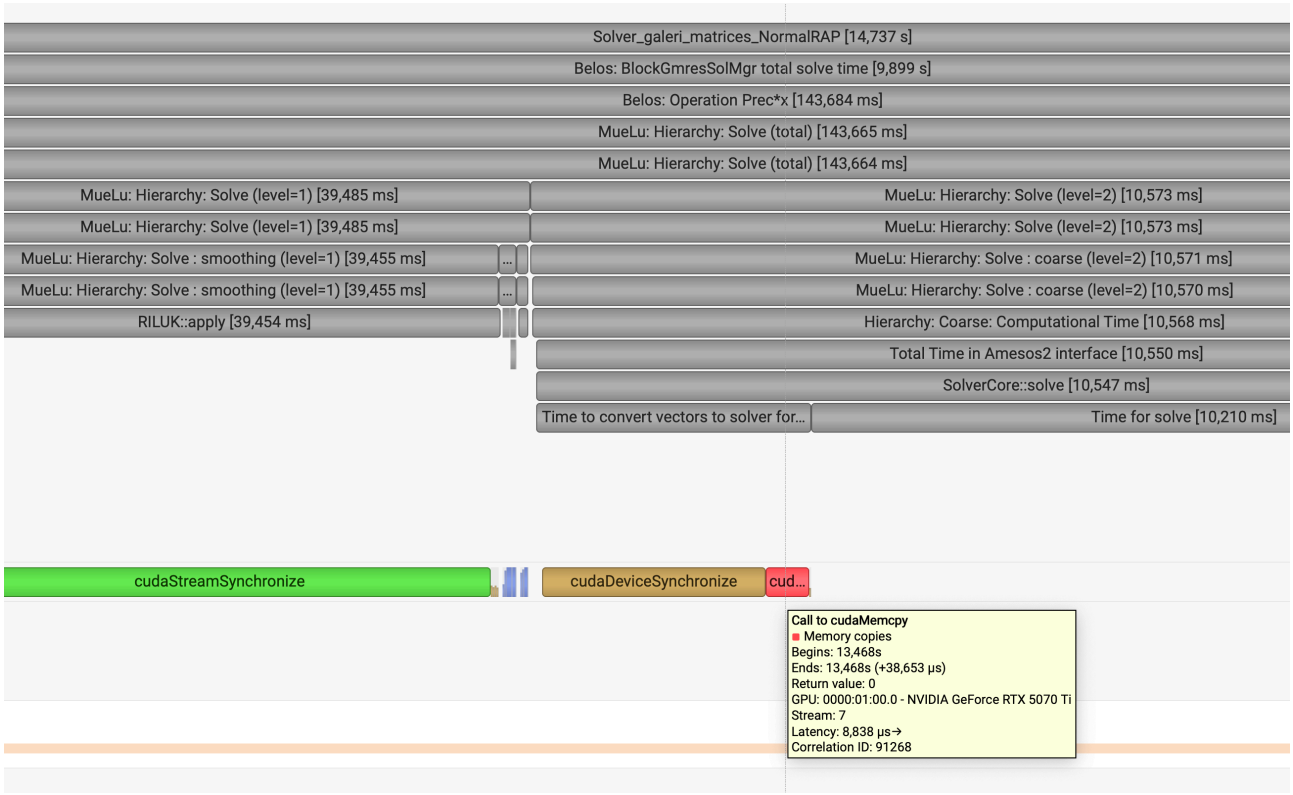


Figure 4.17: Detail of the host-device data transfer of the residual after restriction, preceding the coarse-grid solve, captured in *NVIDIA* Nsight Systems. The highlighted `CudaMemcpy` operation shows the memory transfer of the coarse-grid system from GPU memory to CPU memory.

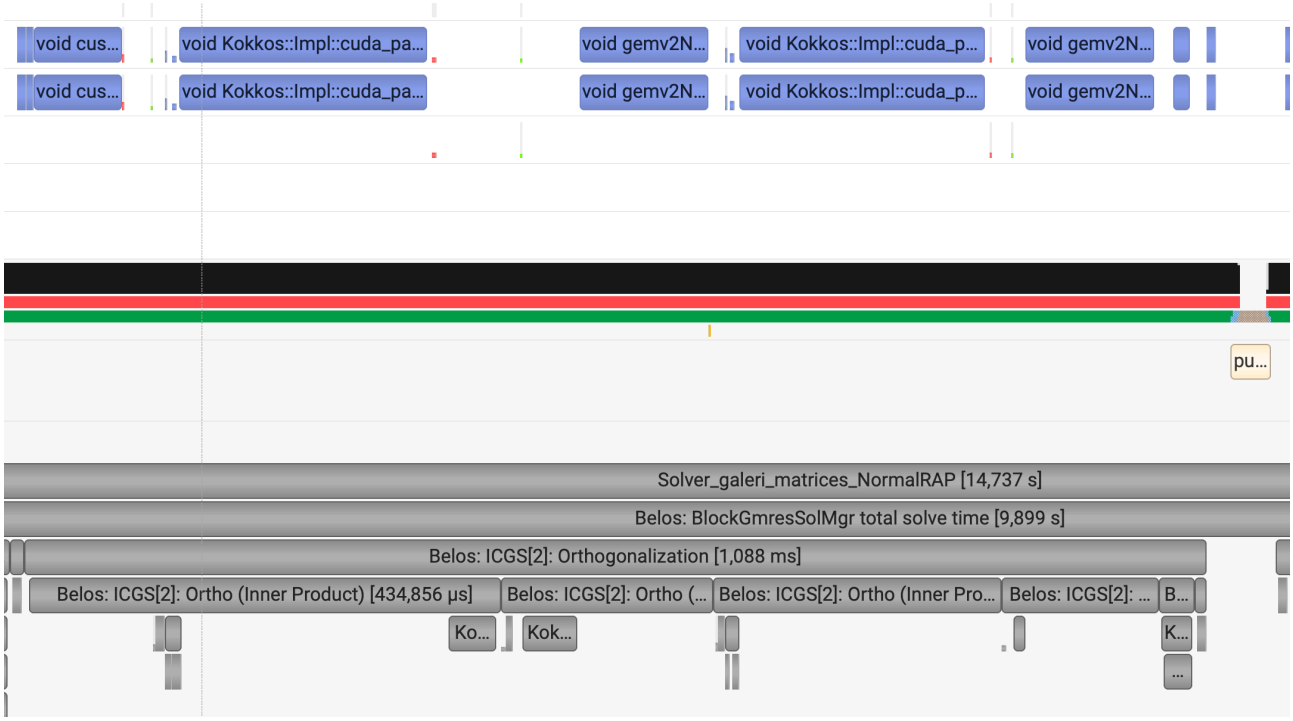


Figure 4.18: Profile of the GMRES orthogonalization phase in *NVIDIA* Nsight Systems. The Cuda API calls during this phase correspond primarily to the underlying BLAS operations required for the orthogonalization process.

However, the poor parallel scaling of the RAS smoother, as discussed previously, prevents us from efficiently creating and solving such large problems in a distributed manner, thus limiting the potential benefits of GPU acceleration in this context.

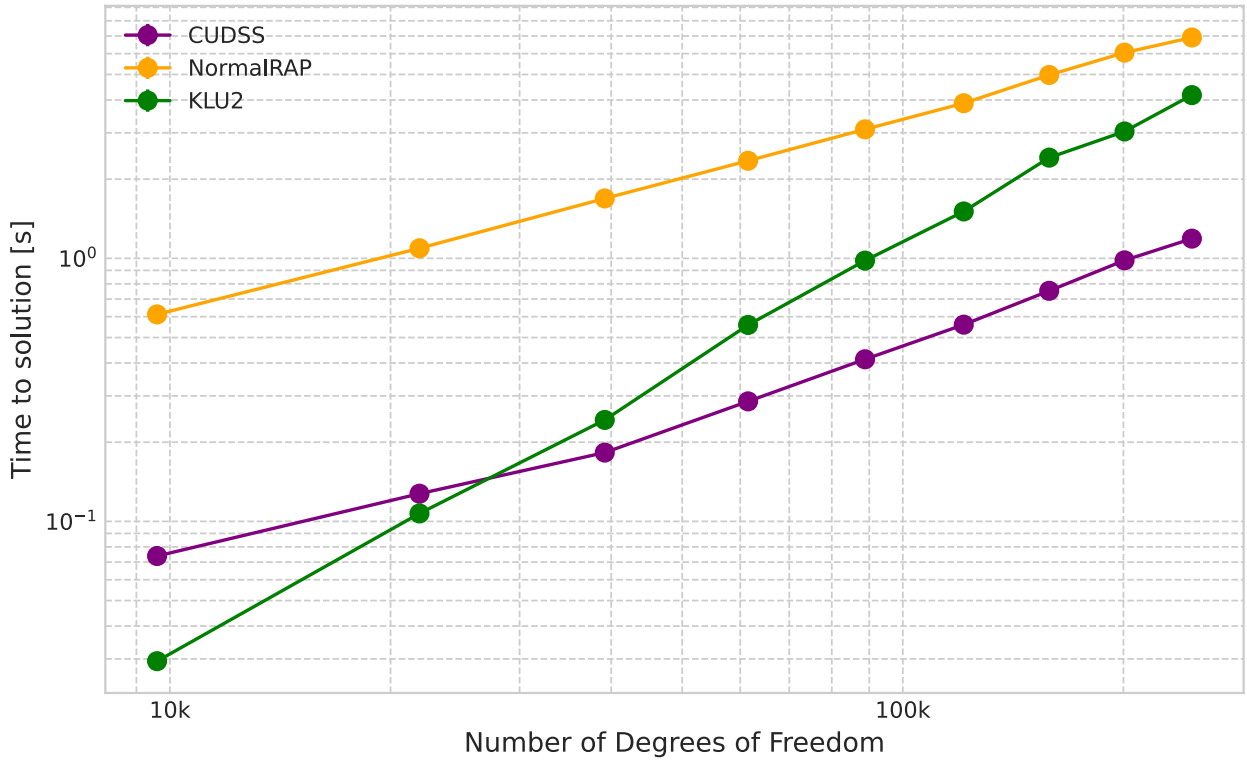


Figure 4.19: Performance comparison of the Shifted Laplacian solver GPU using Shifted Laplacian and *cuDSS* (and *KLU2* for comparison). The plot shows the averaged time-to-solve time over 5 runs as a function of increasing problem size (degrees of freedom).

4.13 Implementation and Validation in HELM

To validate the revitalized *MueLu* Shifted Laplacian preconditioner and to demonstrate its usage, a comprehensive test suite was developed and contributed to the *HELM* codebase. This contribution serves a dual purpose: it provides a concrete example of how to configure and run the solver for a Helmholtz problem, and it establishes a robust framework for verifying its correctness and performance. This work paves the way for the future full integration of the Shifted Laplacian preconditioner as a primary solver option within *HELM*.

The core of this contribution is a new C++ test and is designed to be modular and flexible, with its key functionalities summarized as follows:

- **Problem Generation:** The test begins by constructing a complete linear system for a 2D Helmholtz problem. It uses the *Trilinos* package to generate all necessary matrices: the Helmholtz matrix (**A**), the Shifted Laplacian preconditioner matrix (**P**), and the underlying stiffness (**K**) and mass (**M**) matrices. A right-hand-side vector (**b**), representing a point source, is also created. This ensures a realistic and reproducible test case. Another way to construct the problem is to import the matrices generated by another test.
- **Solver Abstraction:** A central `setup_and_solve` routine orchestrates the solution process. This function is parameterized to test a wide variety of solution strategies. These include the

Shifted Laplacian preconditioner as well as other established solvers for comparison, such as unpreconditioned and preconditioned GMRES and direct solves with *KLU2* and *cuDSS*.

- **Robust Verification:** Correctness is paramount. The test implements two distinct verification methods to ensure the reliability of the solver:
  1. A reference solution is first computed using *KLU2* direct solver.
  2. For any other solver being tested, its computed solution is directly compared against this reference, and the relative error, $\|\mathbf{x} - \mathbf{x}_{\text{ref}}\| / \|\mathbf{x}_{\text{ref}}\|$, is asserted to be below a tight tolerance.
  3. Additionally, the residual norm, $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$, is calculated and checked to ensure it meets a specified convergence tolerance. This dual-check mechanism provides strong guarantees that the solver is not only converging but converging to the correct solution.

This test fixture acts as a blueprint for using the Shifted Laplacian preconditioner. By providing a clear example and a validation framework, it forms the first steps toward integrating this powerful iterative method into the core *HELM* framework.

CHAPTER 5

CONCLUSION

This thesis embarked on an investigation into the development and analysis of high-performance numerical solvers for the time-harmonic Helmholtz equation, motivated by the demanding simulation requirements of next-generation semiconductor manufacturing. The research targeted a dual focus: the formal derivation of the underlying physical and mathematical models but, also, the optimization and benchmarking of both direct and iterative solution methods on modern, GPU-accelerated hardware.

The work began by establishing the mathematical framework for the target application, formalizing the scattered-field formulation and the implementation of *Perfectly Matched Layers* necessary to accurately model wave scattering from complex, heterogeneous structures like GAAFETs. With this foundation, the research proceeded along two parallel paths: the exploration of GPU-accelerated direct solvers and the modernization and optimization of an iterative solver within the *Trilinos* framework.

Our investigation into direct solvers demonstrated the significant potential of GPU acceleration for moderately sized problems. A comparative analysis of *NVIDIA cuSOLVER* and the newer *cuDSS* libraries revealed *cuDSS* to be the superior choice, offering substantially better memory efficiency that enabled the solution of larger problems on the same hardware. For the indefinite Helmholtz system, the combination of *cuDSS* with an appropriate reordering algorithm, such as SYMAMD or MND, consistently outperformed the reference serial and single-threaded CPU solver (*KLU2*) by approximately an order of magnitude. This confirms that for single-node execution, a well-configured GPU direct solver pipeline provides a substantial performance advantage.

Concurrently, this thesis addressed the more scalable approach of iterative solvers by modernizing an unmaintained Shifted Laplacian preconditioner within the *MueLu* library. This effort involved several code corrections, including resolving data type incompatibilities and fixing a bug in the GPU kernel dispatch logic for complex arithmetic, which ultimately enabled the multigrid V-cycle to execute efficiently on the GPU. An extensive parameter optimization study was conducted, identifying the optimal configuration of the smoother, complex shift and level-of-fill to ensure convergence.

The comparative scaling analysis revealed an interesting trade-off between the two solver classes. While the single-GPU direct solver provided a faster time-to-solution on the 2D test problems, the multigrid iterative solver demonstrated more favorable asymptotic behavior. For this specific test case, we observed a superior asymptotic scaling of $O(N^{1.20})$ for the multigrid method, compared to $O(N^{1.58})$ for the direct solver.

Although these exponents may be problem-dependent, the gap suggests that the iterative multigrid approach could become more efficient for much larger systems, such as those arising from 3D problems.

This observation motivates the continued development of the multigrid strategy as a potentially more scalable path forward for the target applications.

However, the path to a fully scalable, GPU-resident solver is not yet complete. Our analysis revealed two primary limitations. First, the parallel scalability of the iterative solver is fundamentally constrained by the use of a classical *Restricted Additive Schwarz* smoother, whose use leads to a non-negligible increase in iterations. Second, the full potential of GPU acceleration is hampered by remaining host-bound computations, namely the coarse-grid solve within the multigrid hierarchy and the reordering phase of the direct solvers.

Based on these findings, we identify the following critical directions for future work:

- **Integrate a Memory-Distributed Direct Solver:**: A next step is to integrate a memory-distributed direct solver. By partitioning the *LU* factors across multiple cluster nodes, such a solver would overcome the single-GPU memory limitation. Future work should analyze the performance trade-offs between the computational throughput of a single GPU and the larger problem sizes enabled by a distributed CPU-based direct solver.
- **Transition to Optimized Schwarz Methods:** The most crucial next step is to replace the current RAS smoother with a *non-overlapping Optimized Schwarz* method. By designing transmission conditions that are transparent to outgoing waves, such a method would eliminate the inter-process reflections that currently hinder parallel scalability, paving the way for efficient execution on large numbers of MPI ranks.
- **Achieving Full GPU Residency:** To eliminate the final data transfer bottlenecks, future efforts should focus on two key areas:
    1. Integrating a GPU-accelerated direct solver, such as *cuDSS*, to handle the coarse-grid solve directly on the device. This would require extending the *Amesos2* interface within *Trilinos*.
    2. Developing or integrating a GPU-native reordering algorithm to eliminate the need to transfer the matrix structure to the host CPU for this expensive pre-processing step.
    3. **Multi-GPU Solvers:** Evaluating emerging multi-GPU direct solver capabilities as they are released to scale beyond single-GPU memory while remaining fully device-resident.

In summary, this thesis advanced the state of Helmholtz solvers within the *HELM*/*Trilinos* ecosystem through several key contributions. The work began by extending the available physics models, deriving the weak formulation for a general scattering problem to handle complex, heterogeneous media beyond *HELM*'s original scope. Based on this formulation, our primary contributions were a modernized, GPU-accelerated multigrid preconditioner and a thorough performance analysis of direct solvers. A key part of this effort was the development of a new test suite that both validates the Shifted Laplacian preconditioner and exemplifies its use. This work lays the necessary groundwork for a promising future direction: formally integrating this powerful iterative method as a core solver within the *HELM* framework.

BIBLIOGRAPHY

[1] John David Jackson. *Classical Electrodynamics*. Wiley, 3rd edition, 1998.

[2] Keiiti Aki and Paul G. Richards. *Quantitative Seismology*. University Science Books, 2nd edition, 2002.

[3] Olgierd C. Zienkiewicz, Robert L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 7th edition, 2013.

[4] Jian-Ming Jin. *The Finite Element Method in Electromagnetics*. Wiley Series in Microwave and Optical Engineering. Wiley-IEEE Press, 3rd edition, 2015.

[5] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, 2003.

[6] Alain C. Diebold, editor. *Handbook of Silicon Semiconductor Metrology*. Marcel Dekker, Inc., 2001.

[7] Eugene Hecht. *Optics*. Pearson, 5th edition, 2016.

[8] Frank Ihlenburg. *Finite Element Analysis of Acoustic Scattering*, volume 132 of *Applied Mathematical Sciences*. Springer, 1998.

[9] Julien Ryckaert, Anda Mocuta, Imogen De Groote, and Naoto Horiguchi. Scaling CMOS beyond FinFETs: from nanosheets and forksheets to CFETs. *IMEC Magazine*, 12:14–19, 2019.

[10] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall D. Stanley. An overview of the Trilinos project. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):397–423, 2005.

[11] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability for C++ applications. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216, 2014.

[12] Peter Monk. *Finite Element Methods for Maxwell's Equations*. Numerical Mathematics and Scientific Computation. Oxford University Press, 2003.

[13] Jean-Pierre Bérenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 114(2):185–200, 1994.

[14] Guillaume Demésy, Artan Nicolet, and Frédéric Zolla. Open source models for the parametric study of diffraction gratings in 2D/2.5D/3D with ONELAB. arXiv preprint arXiv:2208.05315, 2022.

[15] Constantine A. Balanis. *Advanced Engineering Electromagnetics*. Wiley, 2nd edition, 2012.

[16] Timothy A. Davis and Esmond Palamadai Natarajan. Algorithm 866: KLU, a direct sparse solver for circuit simulation problems. *ACM Transactions on Mathematical Software (TOMS)*, 33(1), 2007.

[17] Timothy A. Davis. *Direct Methods for Sparse Linear Systems*. Software, Environments, and Tools. SIAM, 2006.

[18] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[19] Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software (TOMS)*, 29(2):110–140, 2003.

[20] NVIDIA Corporation. cuSOLVER Library Documentation. `https://developer.nvidia.com/cusolver`, 2023.

[21] NVIDIA Corporation. cuDSS Library Documentation. `https://docs.nvidia.com/cuda/cudss/index.html`, 2024.

[22] Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference (ACM '69)*, pages 157–172, 1969.

[23] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30(3):381–388, 2004.

[24] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[25] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities., journal = nternational journal for numerical methods in engineering 79(11). pages 1309–1331, 2009.

[26] Yousef Saad and Martin H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[27] Chris G. Baker, Michael A. Heroux, Heidi K. Thornquist, Alan Williams, Erik G. Boman, and David M. Day. Belos Users' Guide. Technical Report SAND2012-8646, Sandia National Laboratories, 2012.

[28] Luc Berger-Vergiat, Christian A. Glusa, Graham Harper, Jonathan J. Hu, Matthias Mayr, Peter Ohm, Andrey Prokopenko, Christopher M. Siefert, Raymond S. Tuminaro, and Tobias A. Wiesner. Muelu user's guide. Technical Report SAND2023-12265, Sandia National Laboratories, February 2023. Unlimited Release.

[29] Yogi A. Erlangga. Advances in Iterative Methods and Preconditioners for the Helmholtz Equation. *Archives of Computational Methods in Engineering*, 15(1):37–71, 2008.

[30] Ivo Babuška and Frank Ihlenburg. Dispersion analysis and error estimation of the finite element method for the Helmholtz equation. *International Journal for Numerical Methods in Engineering*, 40(20):3717–3740, 1997.

[31] Alvin Bayliss, Charles I. Goldstein, and Eli Turkel. An iterative method for the Helmholtz equation. *Journal of Computational Physics*, 49(3):443–457, 1983.

[32] Alastair Laird and Mike Giles. Preconditioned iterative solution of the 2D Helmholtz equation. Technical Report NA-02/12, Oxford University Computing Laboratory, 2002.

[33] Yogi A. Erlangga, Cornelis Vuik, and Cornelis W. Oosterlee. On a class of preconditioners for solving the Helmholtz equation. *Applied Numerical Mathematics*, 50(3):409–425, 2004.

[34] Yogi A. Erlangga, Cornelis Vuik, and Cornelis W. Oosterlee. Comparison of multigrid and incomplete LU shifted-Laplace preconditioners for the inhomogeneous Helmholtz equation. *Applied Numerical Mathematics*, 56(5):648–666, 2006.

[35] Yogi A. Erlangga, Cornelis W. Oosterlee, and Cornelis Vuik. A Novel Multigrid Based Preconditioner For Heterogeneous Helmholtz Problems. *SIAM Journal on Scientific Computing*, 27(4):1471–1492, 2006.

[36] Martin B. van Gijzen, Yogi A. Erlangga, and Cornelis Vuik. Spectral Analysis of the Discrete Helmholtz Operator Preconditioned with a Shifted Laplacian. *SIAM Journal on Scientific Computing*, 29(5):1942–1958, 2007.

[37] Siegfried Cools and Wim Vanroose. Local Fourier analysis of the complex shifted Laplacian preconditioner for Helmholtz problems. *Numerical Linear Algebra with Applications*, 20(4):575–597, 2013.

[38] Azahar H. Sheikh, Domenica Lahaye, and Cornelis Vuik. On the convergence of shifted Laplace preconditioner combined with multigrid deflation. *Numerical Linear Algebra with Applications*, 20(4):645–662, 2013.

[39] Jong-Hoon Kimn and Marcus Sarkis. Shifted Laplacian RAS Solvers for the Helmholtz Equation. In Jocelyne Erhel, Martin Gander, Laurence Halpern, G. Pichot, T. Sassi, and Olof Widlund, editors, *Domain Decomposition Methods in Science and Engineering XXI*, volume 91 of *Lecture Notes in Computational Science and Engineering*, pages 151–158. Springer, 2013.

[40] Oliver G. Ernst and Martin J. Gander. Why it is Difficult to Solve Helmholtz Problems with Classical Iterative Methods. In Ivan G. Graham, Thomas Y. Hou, Omar Lakkis, and Robert Scheichl, editors, *Numerical Analysis of Multiscale Problems*, volume 83 of *Lecture Notes in Computational Science and Engineering*, pages 325–363. Springer, 2012.

[41] Bjorn Engquist and Lexing Ying. Sweeping Preconditioner for the Helmholtz Equation: Moving Perfectly Matched Layers. *Multiscale Modeling & Simulation*, 9(2):686–710, 2011.

[42] Martin J. Gander, Ivan G. Graham, and Euan A. Spence. Applying GMRES to the Helmholtz equation with shifted Laplacian preconditioning: what is the largest shift for which wavenumber-independent convergence is guaranteed? *Numerische Mathematik*, 131(3):567–614, 2015.

[43] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A Multigrid Tutorial*. SIAM, 2nd edition, 2000.

[44] Ulrich Trottenberg, Cornelis W. Oosterlee, and Anton Schüller. *Multigrid*. Academic Press, 2001.

[45] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[46] Klaus Stüben. A Review of Algebraic Multigrid. *Journal of Computational and Applied Mathematics*, 128(1-2):281–309, 2001.

[47] Lois M. Adams and Harry F. Jordan. Is SOR color-blind? *SIAM Journal on Scientific and Statistical Computing*, 7(2):490–506, 1986.

[48] Olof Widlund and M Dryja. *An additive variant of the Schwarz alternating method for the case of many subregions*. Technical Report 339, Ultracomputer Note 131. Department of Computer Science, Courant Institute, December 1987.

[49] Xiao-Chuan Cai and Marcus Sarkis. A restricted additive schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21(2):792–797, 1999.

[50] Martin J. Gander. Optimized Schwarz Methods. *SIAM Journal on Numerical Analysis*, 44(2):699–731, 2006.

[51] Victorita Dolean, Pierre Jolivet, and Frédéric Nataf. *An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation*. SIAM, 2015.

[52] Martin J. Gander, Frédéric Magoulès, and Frédéric Nataf. Optimized Schwarz methods without overlap for the Helmholtz equation. *SIAM Journal on Scientific Computing*, 28(3):999–1021, 2007.

[53] Trilinos Community. Ifpack2 User's Guide. `https://trilinos.github.io/pdfs/ifpack2.pdf`, 2023.

[54] F. Distrée M. Arnst, R. Tomasetti. Ifpack2: Ifpack2: local trsv cusparse spec selector missing impl_scalar_type, 2025. `https://github.com/trilinos/Trilinos/pull/13971`.

[55] Romin Tomasetti and Maarten Arnst. Efficiently implementing fe boundary conditions using stream-orchestrated execution on gpu. F.R.S.-FNRS - Fonds de la Recherche Scientifique, 07 March 2024.