RICE UNIVERSITY

Accelerated Plane-wave Discontinuous Galerkin for Heterogeneous Scattering Problems

by

Thomas Reid Atcheson

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree
Doctor of Philosophy

APPROVED, THESIS COMMITTEE:

Timothy Warburton
Professor of Computational and Applied Mathematics

William W. Symes
Noah Harding Professor of Computational and Applied Mathematics

Danny C. Sorensen
Noah Harding Professor of Computational and Applied Mathematics

Vivek Sarkar
E.D. Butcher Chair of Engineering, Professor and Chair of Computer Science

Houston, Texas
May, 2015
ABSTRACT

Accelerated Plane-wave Discontinuous Galerkin for Heterogeneous Scattering Problems

by

Thomas Reid Atcheson

This thesis considers algorithmic and computational acceleration of numerical wave modelling at high frequencies. Numerical propagation of linear waves at high frequencies poses a significant challenge to modern simulation techniques. Despite the fact that potential practical benefits led a great deal of attention to this problem, current research has yet to provide a general and performant method to solve it. I consider finite element as a possible solution because it can handle geometric complexity of heterogeneous domains, but unfortunately it suffers from the “pollution” effect which imposes a prohibitively large memory requirement to handle high frequencies. One recent step towards enabling the finite element method to solve high frequency wave propagation in the frequency domain involves using a plane-wave basis rather than the standard polynomial basis. This allows highly compressed representations of scattering waves but otherwise appeared to limit users to nearly-homogeneous problems. This thesis explores the use of plane-waves in a discontinuous Galerkin method (PWDG) for highly heterogeneous problems possibly containing a point source. The low-memory nature of PWDG and the fact that its expressions can be computed in an entirely symbolic manner without quadratures furthermore permits an efficient graphics processing unit (GPU) implementation such that problems with very high
frequencies can be solved on a single workstation. This thesis includes computational results demonstrating results for frequencies in excess of 100 hertz on the Marmousi model, solved using only a single GPU.
Acknowledgements

I would like to thank my advisor Dr. Timothy Warburton for his guidance throughout my time as a graduate student. It was through his constant feedback that I learned what I know today, and without that resource the quality of my understanding could not be what it is now. The value of having such a resource can not be overstated.

A tremendous amount of intuition for my research problems comes from linear algebra and its associated algorithms, and for this I would like to thank Dr. Dan Sorensen and Dr. Mark Embree. Through their courses, seminars, and personal communications I was able to develop a strong sense for what should work and what should not and relate this intuition to computable quantities such as eigenvalues or polynomials. The linear algebraic way of thinking is not a formula that can be written into a paper and published, therefore seeing it personally in action gave me a new and valuable perspective.

I would also like to thank Dr. William Symes for our helpful communications on a wide range of topics. Most importantly to these is the practical industry experience that balances out theoretical points, which can sometimes become disproportionately represented in a mathematical work.

Additionally I would like to thank Dr. Vivek Sarkar who made many helpful points on parallel programming from more formal computer science point of view. These kinds of observations are useful to me because it is impossible to be an expert on everything, and being in an applied mathematics department means I focus more heavily on the mathematics and less so on the computer science. Therefore it is valuable to have someone willing to spend their time to pass some of their expertise to me.
A final individual I would like to acknowledge would be my undergraduate advisor Dr. Jonathan Lewin. It was through his instruction that I gained respect for careful mathematical arguments, and indeed grew to love that side of mathematics. I could not have continued past the undergraduate level without that instruction, where I learned my most important lessons.

As a final point I would like to acknowledge that a PhD is never completed alone in a vacuum. In addition to the incredible amount of work done on the individual level, there is the support network which happens behind the scenes. I would like to thank everyone who helped make this a reality for me, they are too numerous to include specifically.
## Contents

Abstract ii  
List of Illustrations x

1 Introduction 2
   1.1 Overview 2
   1.2 Literature Review 4  
      1.2.1 Addressing Pollution in Finite Element Methods 6
   1.3 Discontinuous Finite Element Methods 9  
      1.3.1 Plane-wave Based Methods 10
      1.3.2 Schur Complement Strategies 12
   1.4 Locally Refined Meshes 14
   1.5 Iterative Solution 16

2 Formulation 19  
   2.1 Helmholtz Equation 21
   2.2 Plane-wave discontinuous Galerkin 23  
      2.2.1 Source Extraction 29
      2.2.2 Perfectly Matched Layer 32
      2.2.3 Numerical Stability and Efficient Formulas 34
      2.2.4 Choice of Local Basis 37
   2.3 Additive Schwarz Domain Decomposition 39  
      2.3.0.1 Homogeneous Box Test Case 46
      2.3.0.2 Heterogeneous Box Case 46
      2.3.0.3 Box With PML Test Case 49
3 Matrix-Free Solution Process

3.1 Restarted GMRES with Right Preconditioning .............................................. 55
3.2 Results ........................................................................................................... 59
3.3 Conclusion ..................................................................................................... 67

4 GPU Implementation ..................................................................................... 71

4.1 A Brief Introduction to OCCA ................................................................. 74
  4.1.1 Hierarchy of Parallelism in OCCA ......................................................... 74
  4.1.2 Memory Spaces of OCCA ..................................................................... 77
  4.1.3 Scheduling Kernels using Julia and OCCA.jl ................................... 79
4.2 PWDG Operator Evaluation ...................................................................... 82

5 Results .......................................................................................................... 86

5.1 Performance Metrics of GPU Implementation ......................................... 86
5.2 Dependence of Error on Source Extraction Patch Size ......................... 94
  5.2.1 Point Source Case $\kappa = 11$ .............................................................. 96
  5.2.2 Point Source Case $\kappa = 31$ .............................................................. 96
  5.2.3 Point Source Case $\kappa = 51$ .............................................................. 97
5.3 Error Behavior for Known Analytic Solutions ...................................... 100
  5.3.1 Convergence Through Mesh Refinement ........................................... 104
    5.3.1.1 The MIE Cylinder Case for $\kappa = 11$ ........................................ 105
    5.3.1.2 The MIE Cylinder Case for $\kappa = 21$ ........................................ 105
    5.3.1.3 The MIE Cylinder Case for $\kappa = 41$ ........................................ 108
    5.3.1.4 Heterogeneous Box Case for $\kappa = 11$ ...................................... 109
    5.3.1.5 Heterogeneous Box Case for $\kappa = 21$ ...................................... 109
    5.3.1.6 Heterogeneous Box Case for $\kappa = 41$ ...................................... 111
  5.3.2 Impact of Local Basis ............................................................................ 112
5.4 Solutions to Select Benchmark Problems ............................................. 116
  5.4.1 Marmousi Model .................................................................................. 117
5.4.2 Scattering “U” Shape .................................................. 120
5.4.3 Scattering Square ...................................................... 122

6 Conclusions and Future Work ........................................... 131

Bibliography ................................................................. 134
# List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>27</td>
<td></td>
<td>28</td>
<td></td>
<td>44</td>
<td></td>
<td>56</td>
<td></td>
<td>57</td>
<td></td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Illustrations

2.2.1 Example mesh $\mathcal{T}$ of triangles such that the domain $\Omega$ the Helmholtz equation is to be solved on is represented exactly, i.e. $\Omega = \bigcup \mathcal{T}$. 

2.2.2 Diagram illustrating source extraction. Inside the gray shaded elements the Helmholtz equation Green’s function is subtracted from the equation in order to make it homogeneous, and therefore PWDG becomes a better candidate for solving it.

2.2.3 Diagram illustrating absorbing layer. A larger box encloses the computational domain and its wavenumber is modified so that incoming waves are captured (not reflected at the boundary between the two boxes) and then gradually dissipated.

2.3.1 Example PWDG spectrum. Note that the eigenvalues are approximately distributed evently which GMRES has issues handling. To compute the eigenvalues using the PWDG algorithm I first obtain a linear operator $A$ by defining $Au = \text{PWDG}(\mathcal{T}, \kappa, N_w, u)$. From this one can obtain eigenvalues of $A$ using an iterative algorithm, but in this case I “assemble” $A$ into a dense format matrix by defining $B_i = Ae_i$, where $e_i$ is the $i$-th column of the identity matrix.

2.3.2 Example mesh of a U-shaped scattering domain. This mesh was generated using GMSH [59].

2.3.3 Example partitioning of mesh using $L=116$ partitions. This partition was generated by making repeated calls to MGridGen [102] and stopping once the number of partitions was below a threshold.
2.3.4 Diagrams illustrating test cases. On the left is a homogeneous box where $\kappa = 11$ and the boundary conditions are purely Robin. In the middle is a box with piecewise constant wavenumber. The top half of the box uses one wavenumber $\kappa_1$ and the bottom half uses another $\kappa_2$. The third box contains a smaller box cavity such that the boundary of the smaller box uses Dirichlet boundary conditions equal to 0, corresponding to a “sound hard” scatterer. The outer boundary uses PML boundary conditions. 

2.3.5 Homogeneous box test case Schwarz levels. The mesh was generated using the GMSH software [59], and the partitions were generated by repeated calls to MGridGen [102]. The top figure is the original mesh, the middle figure is $L = 400$ partitions, and the bottom figure is $L = 8$ partitions.

2.3.6 Eigenvalues of homogeneous box case with and without Schwarz preconditioning. The top spectrum corresponds to the unpreconditioned PWDG operator, the middle spectrum corresponds to the PWDG operator preconditioned by a Schwarz operator based off of $L = 400$ partitions, and the bottom spectrum corresponds to $L = 8$ partitions. Here $\kappa = 11$ and $N_{w}^{D} = 4$ for all $D \in \mathcal{T}$.

2.3.7 Piecewise constant wavenumber test case Schwarz levels. The mesh was generated using the GMSH software [59], and the partitions were generated by repeated calls to MGridGen [102]. The top mesh corresponds to no partitioning, the middle mesh corresponds to $L = 63$ partitions and the bottom mesh corresponds to $L = 16$ partitions.
2.3.8 Eigenvalues of the piecewise constant wavenumber box case with and without Schwarz preconditioning. The top spectrum is obtained from the case with no preconditioning, the middle case corresponds to $L = 400$ partitions, and the bottom case corresponds to $L = 8$ partitions. In this case $\kappa_1 = 31, \kappa_2 = 71, \text{and } N^D_w = 16$ for all $D \in \mathcal{T}$. Figure 2.3.4 defines $\kappa_1$ and $\kappa_2$ and $\mathcal{T}$ is illustrated in 2.3.7.

2.3.9 Schwarz levels for scattering box cavity with PML boundary conditions. The mesh was generated using the GMSH software [59], and the partitions were generated by repeated calls to MGridGen [102]. The top mesh corresponds to no Schwarz partitioning, the middle mesh corresponds to $L = 381$ partitions, and the bottom mesh corresponds to $L = 26$ partitions.

2.3.10 Eigenvalues of the scattering box cavity case with PML boundary condition. This is with and without Schwarz preconditioning. The top spectrum corresponds to an unpreconditioned operator, the middle corresponds to an operator preconditioned using $L = 381$ partitions, and the last corresponds to $L = 26$ partitions. In this case $\kappa = 11,$ and $N^D_w = 4$ for all $D \in \mathcal{T}$ where $\mathcal{T}$ is illustrated in 2.3.9.

3.2.1 Box with PML boundary conditions containing an interior point source.

3.2.2 The mesh $\mathcal{T}$ corresponding to the problem diagrammed in figure 3.2.1 to be used for GMRES convergence studies. The maximum edge length is $h \approx 1/2.$
3.2.3 FGMRES iterations required to reach a fixed relative residual of $10^{-5}$. These results were computed by taking $b$ to be the source term defined by the box point source problem represented by figure 3.2.1 and the mesh $T$ given by figure 3.2.2. Then the number of evaluations of $Ax$ required to solve

$$\text{FGMRES}(A, b, F, 10^{-5}, 1, 10000)$$

where $A, F$ are as in (3.2.1), and the parameters for $F$ are chosen to be

$$\text{reltol} = 0, \text{maxit} = 10, \text{restart} = 1, nparts = L$$

where $L$ is the number of Schwarz partitions and $L = 1$ is defined to be the unpreconditioned PWDG case, i.e. $F = I$.

3.2.4 These results were computed by taking $b$ to be the source term defined by the box point source problem represented by figure 3.2.1 and the mesh $T$ given by figure 3.2.2. Then the number of evaluations of $Ax$ required to compute $\text{FGMRES}(A, b, F, 10^{-5}, 1, 10000)$ where $A, F$ are as in (3.2.1), and the parameters for $F$ are defined to be

$$\text{reltol} = 0, \text{maxit} = i, \text{restart} = 1, nparts = L$$

where $L$ is the number of Schwarz partitions, and $i$ ranges from 10 to 40 on the x-axis of the above figure.

3.2.5 Refined mesh for GMRES convergence studies. The maximum edge length is $h \approx 1/4$.  

64
3.2.6 FGMRES iterations required to reach a fixed relative residual of $1e - 5$. These results were computed by taking $b$ to be the source term defined by the box point source problem represented by figure 3.2.1 with wavenumber $\kappa = 51$ and the mesh $\mathcal{T}$ given by figure 3.2.2. Then the relative cost is calculated by counting the number of evaluations of $Ax$ plus the number of evaluations of $Gx$ required to solve FGMRES($A, b, F, 1e - 5, 1, 10000$) and dividing by the number of evaluations of $Ax$ to solve FGMRES($A, b, I, 0, 1, \infty$) where $A, F,$ and $G$ are as in (3.2.1), and the parameters for $F$ are defined to be $\text{reltol} = 0, maxit = 10, \text{restart} = 1, nparts = L$ where $L$ is the number of Schwarz partitions and $L = 1$ is defined to be the unpreconditioned PWDG case, i.e. $F = I$.

3.2.7 FGMRES iterations required to reach a fixed relative residual of $1e - 5$. These results were computed by taking $b$ to be the source term defined by the box point source problem represented by figure 3.2.1 with wavenumber $\kappa = 51$ and the mesh $\mathcal{T}$ given by figure 3.2.5. Then the relative cost is calculated by counting the number of evaluations of $Ax$ plus the number of evaluations of $Gx$ required to solve FGMRES($A, b, F, 1e - 5, 1, 10000$) and dividing by the number of evaluations of $Ax$ to solve FGMRES($A, b, I, 0, 1, \infty$) where $A, F,$ and $G$ are as in (3.2.1), and the parameters for $F$ are defined to be $\text{reltol} = 0, maxit = 10, \text{restart} = 1, nparts = L$ where $L$ is the number of Schwarz partitions and $L = 1$ is defined to be the unpreconditioned PWDG case, i.e. $F = I$. 

---

66

68
3.2.8 FGMRES iterations required to reach a fixed relative residual of $10^{-5}$. These results were computed by taking $b$ to be the source term defined by the box point source problem represented by figure 3.2.1 with wavenumber $\kappa = 51$ and the mesh $\mathcal{T}$ given by figure 3.2.2. This study is similar to that of figure 3.2.6, however instead of taking $C = 1$ from (2.2.33) it takes the more accurate $C = 2$. Then the relative cost is calculated by counting the number of evaluations of $\mathbf{Ax}$ plus the number of evaluations of $\mathbf{Gx}$ required to solve $\text{FGMRES}(A, b, F, 1e - 5, 1, 10000)$ and dividing by the number of evaluations of $\mathbf{Ax}$ to solve $\text{FGMRES}(A, b, I, 0, 1, \infty)$ where $A$, $F$, and $G$ are as in (3.2.1), and the parameters for $F$ are chosen to be $\text{reltol} = 0$, $\text{maxit} = 10$, $\text{restart} = 1$, $\text{nparts} = L$ where $L$ is the number of Schwarz partitions and $L = 1$ is defined to be the unpreconditioned PWDG case, i.e. $F = I$.

4.0.1 A simplified diagram of an Nvidia Streaming Multiprocessor (SMM) in a Maxwell class GPU. Each SMM contains four warps, which play a similar role as vector units in CPUs, though they are a little more general.

4.0.2 A simplified diagram of a warp within an Nvidia Streaming Multiprocessor. The warp serves as a coarse level of parallelism. All warps may proceed along more or less independent code path, but within each warp the floating point units (called “threads” in Nvidia documentation) must generally all operate on the same path, and in most cases where two paths are possible (for example in an “if/else” block) then the two possible paths are executed sequentially.
5.1.1 Floating point operations of PWDG operator evaluation kernel.
Complex number multiplication is counted as four FLOPs, and addition as two. All values are the result of timing 400 repeated applications of operator evaluation, counting the total number of floating point operations, and dividing by total time. Multiple runs were used in order to rule out startup costs of offloading to the GPU. The results suggest that doubling the number of waves per element has the effect of doubling the floating point cost. Study was performed on a mesh of 16976 triangular elements.

5.1.2 Special function evaluations per second (all scaled by 1e9). This is computed similarly to figure 5.1.1, that is averaged over 400 runs. Each evaluation of the special functions \( \sin, \cos, \exp \) are counted as a single special function evaluation. Many evaluations are required because of stabilized evaluations of the expression \( \exp(z) - 1 \) for complex \( z \). Study was performed on a mesh of 16976 triangular elements.

5.1.3 Measurement of memory bandwidth of the PWDG operator evaluation on the GPU. Bandwidth is computed by measuring the total number of bytes that the PWDG kernel loads from GPU global memory. In some cases a byte contribution is scaled by the length of the OCCA inner-for loop in cases where accesses are uncoalesced. This count is then divided by the total time of execution. The bandwidth value here is low compared to other DG methods because very little data is necessary beyond mesh information and the solution vector, suggesting that the PWDG method has floating point arithmetic as a bottleneck, rather than a memory bandwidth bottleneck. All values are obtained by averaging over 400 runs, and on a mesh of 16976 triangular elements.
5.1.4 Measurement of the speedup of evaluating the PWDG operator on GPU by targeting CUDA and then OpenMP using the OCCA API. The OpenMP code was run first using 8 threads and then 16 threads to observe if hyperthreading improved the time, and then the fastest time was used to compute speedup. The compiler flags used for OpenMP code were the most aggressive vectorization and function inlining options available in gcc as well as the standard -O3 option. These values were obtained by averaging over 400 runs on a mesh of 16976 triangular elements.

5.2.1 Relative error using formula (5.2.1) for $r = 1/2$ and $\kappa = 11$. In this study all runs were reasonably successful except for the case of $N_w = 4$ which is in the preasymptotic regime.

5.2.2 Relative error for $r = 3/2$ and $\kappa = 11$. Despite the bigger element patch for the source function, the results are similar to the case $r = 1/2$ and the same wavenumber.

5.2.3 Relative error (using formula (5.2.1)) for $r = 1/2$ and $\kappa = 31$. Only the last two cases observe convergence for mesh refinement, as the higher wavenumber increases the preasymptotic behavior.

5.2.4 Relative error for $r = 3/2$ and $\kappa = 31$. Despite the larger point source patch, the convergence behavior is no better (and actually, slightly worse) than the corresponding case with a smaller patch.

5.2.5 Relative error (computed using formula (5.2.1)) for $r = 1/2$ and $\kappa = 51$. As in the previous study only the last two cases converge, but the case of $N_w = 32$ converges faster than the case of $N_w = 16$.

5.2.6 Relative error for $r = 3/2$ and $\kappa = 51$. The results for this run are identical to the corresponding study for $r = 1/2$.

5.3.1 Sequence of meshes for mie cylinder scattering test case.

5.3.2 Sequence of meshes for piecewise constant wavenumber test case.
5.3.3 Relative error for $\kappa = 11$ computed using the formula (5.2.1) taking $u_{\text{exact}}$ to be $u$ in the analytic formula (5.3.1). The mesh sequence for this study is given in figure 5.3.1. The error actually grows in most cases because of the small wavenumber compared to the mesh spacing, therefore the poor conditioning of these problems quickly overcomes any possible convergence.

5.3.4 Relative error for $\kappa = 21$ computed using the formula (5.2.1) taking $u_{\text{exact}}$ to be $u$ in the analytic formula (5.3.1). The mesh sequence for this study is given in figure 5.3.1. Only the case $N_w = 32$ appears to exhibit convergence, possibly because the other cases are still preasymptotic.

5.3.5 Relative error for $\kappa = 41$ computed using formula (5.2.1) taking $u_{\text{exact}}$ to be $u$ in the analytic formula (5.3.1). This uses the mesh sequence from figure 5.3.1.

5.3.6 Relative error for $\kappa = 11$ and $\theta_{\text{inc}} = \pi/4$. The relative error is computed using formula (5.2.1) taking $u_{\text{exact}}$ to be $u$ in the analytic solution (5.3.2). The parameters are chosen as $\eta_1 = 1, \eta_2 = 4$.

5.3.7 Relative error for $\kappa = 11$ and $\theta_{\text{inc}} = \pi/3$. The relative error is computed using formula (5.2.1) taking $u_{\text{exact}}$ to be $u$ in the analytic solution (5.3.2), the parameters are chosen as $\eta_1 = 1, \eta_2 = 4$.

5.3.8 Relative error for $\kappa = 21$ and $\theta_{\text{inc}} = \pi/4$. The relative error is computed using formula (5.2.1) taking $u_{\text{exact}}$ to be $u$ in the analytic solution (5.3.2) the parameters are chosen as $\eta_1 = 1, \eta_2 = 4$.

5.3.9 Relative error for $\kappa = 21$ and $\theta_{\text{inc}} = \pi/3$. The relative error is computed using formula (5.2.1) taking $u_{\text{exact}}$ to be $u$ in the analytic solution (5.3.2), the parameters are chosen as $\eta_1 = 1, \eta_2 = 4$. 
5.3.10 Relative error for $\kappa = 41$ and $\theta_{inc} = \pi/4$. The relative error is computed using formula (5.2.1) taking $u_{exact}$ to be $u$ in the analytic solution (5.3.2), the parameters are chosen as $\eta_1 = 1, \eta_2 = 4$.

5.3.11 Relative error for $\kappa = 41$ and $\theta_{inc} = \pi/3$. The relative error is computed using formula (5.2.1) taking $u_{exact}$ to be $u$ in the analytic solution (5.3.2), the parameters are chosen as $\eta_1 = 1, \eta_2 = 4$.

5.3.12 MIE cylinder scatter problem with local refinements. The difference in mesh element sizes requires that full PWDG penalties are used and that a $C$ from the local basis criterion (2.2.33) is carefully selected to balance between both convergence and ill-conditioning.

5.3.13 Relative errors for choice of local basis functions using the criteria (2.2.33) for different values of $C$. Analytic solution for error computation is given by equation (5.3.1). Small values of $C$ put the resulting solution in the preasymptotic, where large values result in a problem with no achievable accuracy due to conditioning.

5.4.1 Image of the Marmousi model, the color represents wavenumber for the case of $100\,Hz$ frequency. This image was constructed from the regular grid definition of Marmousi, but there is an extra layer around the model itself which is to serve as a PML in later computations but otherwise does not have a relationship to the model.

5.4.2 Marmousi model projected to piecewise constant velocity on regular mesh. There are some smearing artifacts near regions with sharp interfaces.

5.4.3 Marmousi model projected to piecewise constant velocity, showing the regular mesh. Note that although Marmousi itself is defined on a regular grid, producing a regular mesh from that grid and projecting the wavenumber to the space of piecewise constant functions introduces smearing artifacts.
5.4.4 Marmousi model projected to piecewise constant velocity on mesh with local refinement. The model appears much closer to the original Marmousi model.

5.4.5 Marmousi model projected to piecewise constant velocity. This shows how automatic mesh adaptivity has picked out the interfaces in order to capture the sharp changes in velocity.

5.4.6 Point source on Marmousi with frequency $f = 120\, Hz$. The point source location is $x = 1500\, m$ and $y = 53\, m$ and the source patch is defined to be all elements within $r = 50\, m$ of the point source location.

5.4.7 Point source on Marmousi with frequency $f = 150\, Hz$. The point source location is $x = 1500\, m$ and $y = 53\, m$ and the source patch is defined to be all elements within $r = 50\, m$ of the point source location. This is zoomed out to see global features of solution. This problem involved more than $20,000,000$ degrees of freedom.

5.4.8 Point source on Marmousi with frequency $f = 150\, Hz$. The point source location is $x = 1500\, m$ and $y = 53\, m$ and the source patch is defined to be all elements within $r = 50\, m$ of the point source location. This image is zoomed into the point source to see finer scale features of solution. This problem involved more than $20,000,000$ degrees of freedom.

5.4.9 Diagram of U shape model with point source. The cut out portion of the internal reflecting object causes incident waves to produce internal reflections with a lot of local variation, these are believed to be difficult to solve with PWDG method except on refined meshes.

5.4.10 Mesh of U shape model.
5.4.11 A U-shape scattering an incident wave originating from a point source. The point source location is \( x = 3, y = 0 \), the wavenumber is \( \kappa = 25 \) and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.

5.4.12 A U-shape scattering an incident wave originating from a point source. The point source location is \( x = 3, y = 0 \), the wavenumber is \( \kappa = 45 \) and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.

5.4.13 A U-shape scattering an incident wave originating from a point source. The point source location is \( x = 3, y = 0 \), the wavenumber is \( \kappa = 65 \) and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.

5.4.14 Mesh of square scatterer. The interior square boundary uses pure Dirichlet boundary conditions, and the outer boundary is a PML layer combined with Robin outgoing conditions.

5.4.15 A square scattering an incident wave originating from a point source. The point source location is \( x = 3.0, y = 0.1 \), the wavenumber is \( \kappa = 25 \) and the internal box takes Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.

5.4.16 A square scattering an incident wave originating from a point source. The point source location is \( x = 3.0, y = 0.1 \), the wavenumber is \( \kappa = 45 \) and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.
5.4.17 A square scattering an incident wave originating from a point source. The point source location is \( x = 3.0, y = 0.1 \), the wavenumber is \( \kappa = 65 \) and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.
Nomenclature

\( \alpha \)  
Penalty parameter for approximate gradient.

\[ \{u\} = \frac{1}{2}(u^{D+} + u^{D-}) \]  
Average of solution on elements \( D^+, D^- \in \mathcal{T} \) which share an edge

\( \beta \)  
Penalty parameter for approximate amplitude.

\( \delta \)  
Penalty parameter used on Robin boundaries.

\( \Gamma_D \)  
Sound-hard scattering boundary. Note: \( \partial \Omega = \Gamma_D \cup \Gamma_R \)

\( \Gamma_R \)  
Approximate outgoing boundary. Note: \( \partial \Omega = \Gamma_D \cup \Gamma_R \)

\( [u] = u^{D+} - u^{D-} \)  
Jump of solution on elements \( D^+, D^- \in \mathcal{T} \) which share an edge

\( \kappa \)  
Wavenumber. Linearly proportional to wave frequency.

\( \mathbf{n}_D \)  
Outward pointing normal for domain \( D \)

\( \mathcal{T} \)  
Mesh of triangles such that \( \Omega = \bigcup \mathcal{T} \)

\( \mathcal{T}_l \)  
Schwarz partition for \( l = 1, \ldots, L \).

\( \Omega \)  
Domain on which the Helmholtz equation is solved.

\( \partial D \)  
Boundary of a domain \( D \)

\( f \)  
Volume source term to the Helmholtz equation

\( g \)  
Dirichlet boundary source term to the Helmholtz equation

\( h \)  
Maximum edge length in a mesh \( \mathcal{T} \)
$h_D$ Edge length function for the triangle $D \in \mathcal{T}$.

$L$ Number of partitions used for additive Schwarz preconditioner.

$s \in [0,1]$ Interpolates between UWVF and PWDG such that $s = 0$ corresponds to UWVF and $s = 1$ to full PWDG
Chapter 1

Introduction

1.1 Overview

The goal of this thesis is to derive an efficient finite element algorithm for solving wave propagation problems in the frequency domain. In achieving this goal, a variety of difficulties must be overcome. These difficulties may all be traced back to what Zienkiewicz termed the “short-wave problem” [140], a problem which remains essentially unresolved for modern polynomial based finite element methods [27]. For high wave frequencies, finite element methods generally require a prohibitively expensive degree of approximation before they yield a physically meaningful result. The general strategy for overcoming this is to use high order approximations in combination with stabilization techniques which, when in the asymptotic regime, can drastically reduce phase errors inherent in finite element methods [2,20]. They can furthermore ensure unconditional stability with respect to the mesh [50] (a desirable property in the context of finite element methods involving locally refined meshes). A method which satisfies these requirements and furthermore naturally scales well in parallel computer architectures is the discontinuous Galerkin (DG) method [14], the primary topic of this thesis.

Despite many of the benefits of using DG methods, they still tend to not yield satisfactory algorithms at higher wave frequencies. Therefore I will adopt the strategy of Cessenat and Despres in [32] and piecewise plane-waves instead of piecewise
polynomials. This technique was later generalized into the plane-wave discontinuous Galerkin method [58]. The Plane-wave discontinuous Galerkin method has the potential to reduce the number of degrees of freedom necessary required to achieve a desired accuracy in the specific case of the Helmholtz equation. However, further work is necessary to make this technique more broadly useful. The use of plane-waves for example limits the wave frequency to be piecewise constant on the computational mesh, and so until recently has not been seriously explored to handle very heterogeneous models (see e.g. [39], where a “smoothed” Marmousi model is used). Therefore in this thesis I consider the benchmark non-smoothed Marmousi model as a test case, demonstrating that the plane-wave DG method can handle strong heterogeneities. To handle these heterogeneities however will require local mesh refinements in order to fit the material model with piecewise constants.

Further difficulties of the plane-wave DG method come when local mesh refinements are employed, which make it impossible to use a globally uniform number of plane waves per mesh element. In this case I introduce a mesh and frequency dependent heuristic for a robust and cheap choice of the number of plane-waves. Previous work has relied on local condition number computations to achieve this effect (e.g. [85]) but this is expensive and does not lend itself to computation in a robust matrix-free manner, since computing the condition number accurately is roughly as difficult as solving the original system.

The issue of solving the resulting linear system from plane-wave DG has also posed difficulty, leading many researchers to try domain decomposition or block Jacobi style preconditioners [25], [12] combined with the GMRES algorithm. While these have led to improvements over the initial approaches taken in by Hutttunen et al. in [85] with Richardson iteration or BiCGSTAB, the results of the domain decomposition
work so far appears to be dependent on full matrix assembly and exact application of the preconditioner, leading to the full benefit of spectral clustering which is known to improve the performance of GMRES [129]. I will show that in terms of work, it is simply a better alternative to increase the number of restarts in the outer GMRES algorithm without preconditioning rather than to include an inner domain decomposition preconditioner which is itself solved approximately via GMRES.

Finally I will present a high performance implementation of these ideas which can run efficiently on multicore CPUs via OpenMP or OpenCL or graphics processing units (GPUs) using OpenCL or CUDA. These three threading models are used through the unified interface OCCA [95], and the host language used was Julia [26] through its OCCA interface, OCCA.jl [110].

1.2 Literature Review

Wave propagation forms the foundation of many modern computationally intensive tasks in industry. These applications can range from the oil industry: simulating material resistivity for logging-while-drilling [8,73], to the biochemical device industry: piezoelectric material simulation for the optimization of biochips [9,10,73]. Difficulties however commonly arise when a wave simulation involves wavelengths of a much smaller scale than the domain under consideration. Industrial applications inevitably involve simulating waves in a spatial domain which can be several orders of magnitude larger than the wavelength of interest, and this difference of scale leads directly to computational difficulty.

When a scale difference of large magnitude exists in wave simulations, computational difficulty arises due to the nature of existing simulation algorithms. Most simulation techniques replace the domain of interest with a discrete approximation
on which the desired waves can then be calculated to a desired accuracy, and a higher quality approximation leads to a more computationally intensive simulation. A common engineering rule-of-thumb for the required quality of this spatial approximation comes from the Nyquist sampling theorem [119], and it suggests a fixed finite number of points per wavelength present. This rule of thumb does appear necessary, as violated conditions of the Nyquist theorem can lead to severe phase errors [123]. Therefore if the spatial approximation does not sufficiently resolve the smallest scale of the simulation, then the accuracy of the entire simulation could be in jeopardy, even the well resolved large scales. This difficulty led researchers in scientific computation to investigate simulation methods which have special properties that, despite the mentioned difficulties inherent to wave propagation, could be exploited to yield more efficient solutions.

In the context of frequency domain wave simulations, where waves are assumed to be periodic in time, there are two major numerical techniques for their computation: boundary element methods [36,91], and finite element methods [34,64,86,126]. Boundary element methods have the advantage that they reduce the dimensionality of computation, can easily incorporate important boundary conditions [66], and permit the use of fast approximate matrix-vector products by way of the fast multipole method [115]. Advantages of the boundary element methods are however somewhat offset by the unstructured nature of the resulting system and difficulties in the inhomogeneous case. Desire for more flexibility regarding the difficulties of the boundary element method led to the proposal of finite element methods for wave propagation problems. Finite element methods yield more structured and sparse systems suitable to well-researched methods in large scale linear solutions, and can furthermore easily incorporate inhomogeneity into their formulation [64,126]. A cost accompanies the
increased flexibility, however. A well-documented “pollution” effect occurs in finite element methods which do not aggressively resolve the smallest scales of the solution; this requires resolution in excess of classical “points per wavelength” requirements, and if unmet the error in small scales destroys accuracy at all scales [21, 65]. Maintaining flexibility in finite element methods while addressing pollution has become a significant topic of research.

1.2.1 Addressing Pollution in Finite Element Methods

When using a classical finite element approach to solve frequency domain wave problems the now well known pollution effect occurs in the high frequency limits, corresponding to scale differences from resolution requirements and the computational domain. In terms of analysis this difficulty is often attributed to loss of definiteness of the Helmholtz operator [87], which corresponds to a loss of coercivity in classical finite element theory for proving well-posedness results. The theoretical difficulty often surfaces as a wavenumber dependent constant in stability results, which ultimately appear in error estimates.

Early investigation into pollution error decomposed error estimates into a wavenumber independent term and a wavenumber dependent term (an “asymptotic” term and a “pre-asymptotic” term). A representative paper is that of Bayliss et al. [21] which showed through such arguments that standard points-per-wavelength requirements do not yield sufficient resolution to avoid pollution errors in classical finite element methods. Later Ihlenberg and Babuska in [87] found a connection between the pre-asymptotic pollution term in error estimates and the tendency of Galerkin methods to produce spurious dispersion, leading to phase errors. These discoveries suggest therefore two potential causes of pollution when using classical finite element meth-
ods: loss of coercivity in the high wavenumber limit (potential ill-posedness), and spurious dispersion in the discrete solution (phase errors); techniques for improving the finite element method therefore often focus on one or both of these problems in the hopes of producing a more efficient numerical algorithm.

Directly targeting dispersion in finite element methods began with under-integrating stiffness matrices with special quadrature rules that can significantly reduce phase errors. An early application of this was done by Mullen and Belytschko in [103], but later successful applications also include that of Guddati and Yue [63] and Thompson and Kunthong [127]. Other techniques instead use carefully defined variational forms to minimize dispersion error. An early modification to the classical Galerkin method called the Galerkin/Least-Squares method was developed by Harari and Hughes [67] and involved adding to the original formulation the residual operator of the governing equations, scaled by a tunable parameter. This additional term served to promote spurious low frequency modes to high frequency spurious modes with smaller energy, and could be optimally chosen according to discrete dispersion analysis.

In one dimension, dispersion analysis based techniques often completely remove the phase errors in classical Galerkin methods [67]. The natural extension to two dimensions, analyzed by Thompson and Pinsky [128], could not fully eliminate phase errors with the residual term however. The extra directions of possible propagation limits in two dimensions how the tunable parameter may be chosen, leading Thompson and Pinsky to minimize dispersion errors along the dominant direction of wave travel [128], still yielding an improved method in many cases [126]. It would later be proved by Babuska and Sauter in [20] that beyond one dimension the pollution phenomena can never be completely eliminated in a finite element method, however Ainsworth in [3] showed that if the order of approximation is permitted to be variable (along
with the mesh), then phase errors can be efficiently minimized. This shows the need for more flexibility in how the numerical solution is represented.

An alternative to directly controlling the discrete dispersion is to permit the use of more oscillatory solution spaces (which as a slight abuse of terminology I will refer to such methods universally as “high order methods”). This permits easier resolution of fine scales inherent in large wavenumber simulations. Developments in the finite element method which use more oscillatory functions to represent the solution have seen reasonable success, and they may be broadly categorized into conforming or nonconforming methods. This categorization corresponds to methods which require high regularity in the numerical solution (conforming) versus those which do not (nonconforming).

The use of conforming high order elements began as early as 1969 by Silvester in [121] for the solution of time harmonic wave problems on complicated domains. The reasoning behind their use was the ability to minimize the degrees of freedom required to obtain a desired accuracy, corresponding to a favorable resolution requirement. Another alternative is the partition of unity method introduced by Babuska and Melenk in [96], where regularity is imposed in a nonstandard way, and it permits a greater flexibility with the approximation. The conforming methods which have been introduced thus far however suffer from a discrete ill-posedness issue which results in ill-conditioning in the resulting linear system. Constraints on approximation properties of the solution space must be met before discrete well-posedness is realized (in low order finite elements this corresponds to a mesh refinement criteria), this restriction has been quantified in generality by Esterhazy et al. in [50] and in the same paper also shows that classical nonconforming methods suffer no such constraint. A potential solution to the discrete ill-posedness issues raised by conforming methods
therefore is to relax the regularity constraint and employ stabilization techniques
to yield automatic mesh independent discrete stability, and the remainder of this
introduction will focus on such nonconforming high order methods.

1.3 Discontinuous Finite Element Methods

Rather than requiring continuity in the numerical solution a common technique in-
volve using an approximation with jump discontinuities and penalizing the jumps
in some way to guarantee stability (consistency with governing equations often is an
automatic feature of these methods). Done correctly, the resulting discrete system
frequently is unconditionally well posed, a feature that is absent in most conform-
ing methods [50]. Despite the improved stability properties of discontinuous finite
element methods, they generally involve an unfavorable increase in the degrees of
freedom for the resulting system [113].

One popular application of discontinuous Galerkin methods to frequency domain
problems arose out of Maxwell’s equations where conforming methods often require
the use of complicated edge elements to ensure curl conformity, or else risk spurious
modes entering the solution [99]. This use of the discontinuous Galerkin method
began with Perugia and Schötzau in [106] where the local discontinuous Galerkin
(LDG) method [41] was applied to low frequency problems for the time-harmonic
Maxwell’s equations. Later, Perugia et al. in [106] and Houston et al. in [74] would
consider high frequency problems but instead using the interior penalty discontinuous
Galerkin (IPDG) [13]. One of the advantages of the penalty based methods like LDG
and IPDG is that the penalty parameter serves to damp out spurious modes which
would otherwise have to be dealt with using a curl conforming basis, this was shown
e.g. numerically by Hesthaven and Warburton in [68] and theoretically by Warburton
and Embree in [134].

One novel application of the discontinuous Galerkin method to the Helmholtz equation was by Alvarez et al. in [6] where a unique choice of parameterized penalty terms led to a method which in their experiments led to a functional points-per-wavelength requirement that did not produce pollution for fixed approximation order, a feature that is absent in most finite element methods past the first dimension [20]. A similar approach was taken by Feng and Wu in [55, 56] where an IPDG method with additional jump terms involving normal derivatives and tangential derivatives were also included in the formulation, yielding again a method with favorable stability properties. The results for IPDG methods were later extended to two LDG methods by Feng and Xing in [57].

The mentioned methods thus far do not attempt to remedy degree-of-freedom issues associated with discontinuous methods. Thus the next subsections will be dedicated to specialized techniques which modify discontinuous finite element methods in such a way as to reduce the required degrees-of-freedom to achieve a desired accuracy.

### 1.3.1 Plane-wave Based Methods

The use of specialized non-polynomial basis in a variational method began as early as 1926 by Trefftz in [130]. Numerical methods based on this idea became known as “Trefftz methods,” for a survey of such methods see [108]. The primary type of basis used in an approximation came from physical motivations rather than mathematical ones. Plane waves for example would be used despite having poorly understood approximation properties [50]. However, the fact that plane waves satisfy an adjoint version of the governing equations (Helmholtz, Maxwell’s, etc) permits the use of specialized arguments to prove exceptional convergence rates in those special cases [50].
An early application to frequency domain wave propagation by Cessenat and Despres in [32] used a plane wave basis applied to the Helmholtz problem, and later the time harmonic Maxwell equation [33]. The plane wave basis applied to frequency domain wave problems has some unique virtues which separates it from a polynomial basis. One benefit of the plane wave basis is that despite the fact it is not complete basis in the full solution space for the Helmholtz equation in general, individual plane waves satisfy an adjoint Helmholtz problem and this fact can be used to derive powerful approximation results for solutions of the homogeneous Helmholtz equation [50]; the approximation result obtained requires significantly fewer degrees of freedom than polynomials would typically require for the same accuracy. Combined with a novel weak form which further reduces the degrees of freedom and permits fast operator assembly, the “Ultra Weak Variational Formulation” (UWVF) of Cessenat and Despres has seen successful applications and research beyond the Helmholtz equation.

In the early publication of Cessenat and Despres [32] several additional advantages of their formulation were noted, but not investigated further. Among these was the potential for utilizing different numbers of plane waves per element in order to control ill-conditioning. This was further explored by Huttunen et al. in [85] where a much tighter control on the conditioning was achieved through various strategies in varying the approximation order in each element. This approach saw further extension to more complex elastic wave problems [80, 82] and the time harmonic Maxwell equations [81]. Further improvements involved translating the UWVF into a discontinuous Galerkin method for the purpose of obtaining a more standard energy norm estimate of the error, done in parallel by Buffa and Monk in [29] and Hiptmair et al. [70, 71, 97], and preliminary work on parallelism [81, 84]. The UWVF is not the only successful formulation to have used plane waves. Developed in parallel
with the UWVF was a least square method [98] and the discontinuous enrichment method [51]. The discontinuous enrichment method however is best explained as specialized formulation amenable to a Schur complement, and so further discussion and comparison with the UWVF is postponed until subsection 1.3.2.

1.3.2 Schur Complement Strategies

An attractive feature of a numerical method is the ability to use its structure to reduce the cost of solving the linear system. A common way this is done in finite element methods, discontinuous or not, is to partition the system in such a way to permit a Schur factorization. If the resulting structure of this system is appropriate, then this decomposition permits the construction of a Schur factor whereby the coupling degrees (usually of much smaller dimension than the non-coupling degrees) can be solved independently of the non-coupling degrees of freedom, and if necessary the non-coupling degrees can be recovered by a cheap block operation. In the context of discontinuous finite element methods for frequency domain problems, there are two main approaches to obtaining this type of structure: Lagrange multipliers, and hybridization.

The Lagrange multiplier approach is typified by the discontinuous enrichment (DE) method of Farhat et al. [51]. It involved a discontinuous polynomial basis of a uniform fixed order, and a plane wave “enrichment” which serves to capture fine scale oscillations that are missed by the polynomial basis (cf. the variational multi-scale method [79]). The discontinuous nature of the approximation however leaves open the question of inter-element boundary conditions. These are imposed by introducing a piecewise polynomial basis on element edges as “Lagrange multipliers” which serve to weakly enforce inter-element continuity and boundary conditions. The
primary difference between DE and the UWVF is in how edge degrees of freedom are introduced into the system. For DE the plane wave basis is an intermediary basis and the dimension of the Lagrange multiplier space is chosen for well posedness, but the plane wave enriched space is statically condensed out by a Schur complement technique. The ultimate solution is expressed in an edge polynomial basis defined on the given mesh. In the UWVF the solution is directly cast in the plane wave basis and dimensionality reduction occurs at the element level by way of a special isometry mapping [32, 101]. DE would later be applied to the Helmholtz equation in [52–54], and extended to three dimensions by Tezaur et al. in [124]. The development and extensions of discontinuous enrichment was accompanied by a rigorous error analysis by Amara et al. in [7], which was followed by a performance comparison done by Wang et al. in [133] showing that in terms of the degrees of freedom count required to obtain similar accuracy, the discontinuous enrichment method compares favorably to the UWVF [32] and partition of unity method [96]. The idea of enriching a solution space will be revisited later in this thesis, but instead in the context of attempting to alleviate the difficulties of using plane waves to solve inhomogeneous problems.

In [40] Cockburn et al. showed that many types of finite element methods can be “hybridized,” meaning their coupling degrees of freedom can be partitioned off in the weak formulation and still result in a well posed method. This is closely related to introduction of mortar variables as done in [24], [23]. In discontinuous Galerkin methods for elliptic problems (close in form to Helmholtz equation because of the Laplace operator) there is a large coupling in the degrees of freedom at boundaries that is necessary to ensure well posedness [14]. Hybridization can treat these boundary “fluxes” independently in a way that they may be solved for efficiently in the same Schur complement fashion as with discontinuous enrichment, and this has been shown by Kirby
et al. to be competitive with classical continuous finite element methods combined with a static condensation approach [90]. Further work hybridizing weak forms for the Helmholtz equation include Monk and Griesmaier in [62], where a hybridized LDG scheme was introduced and the resulting Schur complement system had wavenumber independent conditioning, and Chen et al. in [35]; these two approaches have been generalized by Cui et al. in [45], and the accompanying stability results therein are independent of mesh and polynomial order. Up until this point however the focus has been solely on reducing the degree of freedom constraints inherent in high frequency wave propagation. The final tasks which pose difficulty are adaptivity, preconditioning, and high performance implementation; these are the subjects of the last sections.

1.4 Locally Refined Meshes

In addition to the need for a “sufficiently resolved” solution, due to the pollution effect, there is frequently cases where mesh refinement mesh take place only in highly localized areas in order to resolve areas with large and localized gradients, examples of this in plane wave DG methods can be seen in [71]. This can be done in an automatic fashion through error estimation, a brief overview of this is given below, although this thesis will not deal directly with automatic mesh refinement, it will however handle cases of local refinement which could result from such algorithms.

The desire for efficient estimation of error in the solution process arises from the desire to provide a reliable stopping criteria for mesh refinement so that some degree of confidence may be ascribed to the result [19]. Early work for such effective estimators for elliptic problems began with Babuska and Rheinboldt in [17, 18], but algorithms with proven finite termination for a prescribed error tolerance did not arise until Dörfler with [46] where a marking strategy for elements to be refined was presented,
and is now known as “Dörfler marking.” With reliable error estimation and Dörfler marking combined with a refinement algorithm such as triangle bisection [111, 112] one obtains an efficient and convergent adaptive finite element method with varying degrees of confidence in the final result, where the confidence in the result would naturally depend on constants arising from reliability estimates (upper bounds for the estimator in terms of true error) [5], and certain quasi-orthogonality results which are used in the evaluation of contraction properties for convergence [72, 120]. Difficulty arises in this analysis for the case of wave propagation problems as the wavenumber implicitly is a component of the unknown constant factors which arise out of the analysis, yielding another effect of pollution, except in this case it has yet to be fully quantified [100] but can in some cases be controlled [16, 105].

A posteriori analysis in wave propagation problems exhibiting large frequencies is a difficult problem, however an exhaustive review of the literature is outside the scope of this thesis, which seeks to develop a solver that is robust in the presence of local refinements, and show its applicability in parallel DG codes. Surveys for a posteriori estimates in wave propagation may be found for the acoustic case in [105] or for the electromagnetic case in [100, chapter 13]. For discontinuous Galerkin methods in particular there have been recent advances towards the construction of reliable estimators for elliptic operators, see e.g. [4, 30, 31, 75–77] for a posteriori estimators constructed to solve elliptic problems and to solve the more difficult curl-curl elliptic problems. This thesis will not directly work with mesh adaptivity, but the method is of suitable generality to handle the meshes which are the result of local refinements. Such as the recent work on residual based error estimateion for which there is not yet a significant amount of literature, however residual based estimators have been introduced for HDG applied to curl-curl problems by Cartensen et al. in [31] and for
traditional elliptic problems by Cockburn et al. in [42,43]. Following the construction of the method [31] a convergence analysis was undertaken to its extension for indefinite elliptic problems (in this case Helmholtz’ equation) by Sharma in the thesis [120] and in the paper with Hoppe [72], and the related work extending these ideas specifically to that of plane wave DG by Kapita et al. [89].

1.5 Iterative Solution

Another mathematical challenge in solving frequency domain problems in high frequency limits comes from the discrete system of linear equations. For non oscillatory elliptic problems a highly successful approach has been to further approximate the problem at the discrete level with a sequence of smaller problems, each of which can be solved directly in an efficient way. The full discrete problem is then used to guide convergence of the approximate problems by serving as an exact representation of the resulting residuals of the approximate problems. Convergence is guaranteed for iterative methods like this when the residual evaluation and the approximate sub-problems are sufficiently close to the same problem, see e.g. Xu [137] for a very general presentation of this idea.

Although wave equations in the frequency domain are similar in form to many elliptic problems which have been successfully treated with iterative approaches, their oscillatory nature makes achieving robust convergence challenging [49]. Although a careful mathematical analysis of this difficulty has been only achieved in some simple model problems (see e.g. Lieson and Strakos [94], or Zhang [138]), it can be intuitively understood by observing some of the difficulties already discussed with respect to wave equations: namely the pollution effect.

The pollution effect in approximate solutions to wave equations in effect already
dictates that one must use a (typically excessive) resolution in order to ensure the result is physically meaningful. Therefore an attempt to reduce the discrete system of equations into a sequence of approximate subproblems must still respect this requirement. Many authors have noticed this problem, and in [49] it is described as admitting phase errors which are too large in the approximate problems (which incidentally is the same problem occurring in the pollution effect for finite element methods). Of the iterative methods which have been successful, they have all made heavy use of the underlying physics for their successful operation. One such general technique is the Multigrid method, which relies on smoothers and coarse-grid corrections.

Because of the pollution problem, a robust coarse-grid correction has yet to be developed which works in full generality for wave equations [49]. Typically in this case the coarse grid corrections must be “sufficiently accurate” (see e.g. Bramble et al. in [28] for an analysis in a general setting). Unfortunately “sufficiently accurate” in practice this requires to use a coarse-grid correction which is no cheaper than simply solving the original system in the first place, thus bringing into question whether it makes sense to use a multigrid algorithm at all. This and other difficulties (such as handling heterogeneous wavenumber) disqualified the use of multigrid in this thesis, but recent work suggests that some of these difficulties may be overcome - discussion on these points is postponed until the final chapter.

Another class of preconditioners, the primary one investigated by this thesis, are domain decomposition based preconditioners. These partition a problem into subproblems and solves each of the subproblems in a specified order. Recent work has developed these preconditioners for the PWDG method (see e.g. Antoinetti et al. [12] for Schwarz methods). This thesis examines Schwarz based preconditioners from the point of view of the PWDG formulation in chapter 2 as well as their performance in
a practical matrix-free formulation in chapter 3.
Chapter 2

Formulation

The novel contributions of this chapter are a mathematical derivation of a local basis selection heuristic to control conditioning, the formulation of source extraction for the first time with a plane-wave based method, numerical experiments with Schwarz domain decomposition based preconditioners that show their beneficial spectrum clustering characteristics, and development of a basis stabilization which allows for PML on larger domains without suffering floating point underflow or overflow.

In this chapter I will show the governing equation to be solved throughout the thesis, and I will describe the approximation method used to solve it. The governing equation of interest is the Helmholtz equation, which arises from many applications where acoustic waves are studied over relatively large timescales, such as in seismic applications [107]. The equation arises by starting first with the standard acoustic wave equation and assuming that the solution has become periodic in time (with a fixed frequency). After eliminating the time components the Helmholtz equation is left over, which for a given frequency describes the amplitude of the acoustic wave. This equation is described in section 2.1.

While eliminating the time variable in this fashion might seem like an attractive simplification, the resulting Helmholtz equation tends to be at least as difficult to solve - if not more difficult - than the originating acoustic wave equation, and for a
wide range of numerical solution techniques this difficulty scales with the magnitude of the assumed wave frequency [50].

The low frequency regimes of the Helmholtz equation are already well handled by existing finite element and finite difference technology, as this case is a moderately perturbed Poisson or Laplacian and methods can be nearly directly translated from that literature into a successful low frequency Helmholtz solver. Recent work [125], [133] has extended the manageable frequencies into “medium” frequencies, so-called because standard techniques such as those which already work low frequencies will also be able to yield answers for medium frequencies but specializations of these methods can yields considerable performance improvement. The frequencies of interest in this thesis are “high frequencies.” These are frequencies for which standard solution techniques lead to an intractably large system of equations which are in addition highly resistant to most modern iterative solution techniques [49]. Solving problems in this frequency regime requires highly specialized discretization techniques well beyond what is already done to handle “medium” frequencies.

Higher frequencies have recently become more feasible through the introduction of more specialized discontinuous Galerkin and related methods where polynomials are replaced with or enriched by free-space solutions to the Helmholtz equation, such as plane-waves or radial Bessel functions. In this thesis I will use a plane-wave based discontinuous Galerkin method [58] which I describe in section 2.2. For the specific case of homogeneous solutions to the Helmholtz equation, plane waves can provide very rapid convergence compared to polynomials [97]. The benefit of this rapid convergence is that plane-wave based methods require fewer degrees of freedom to achieve the same accuracy as a polynomial based method, reducing its memory footprint. The downside of plane-wave methods is the lack of robust iterative solvers which do
not rely on sparse matrix assembly or triangular solvers at some level (see e.g. [25] and [12] where domain decomposition preconditioners are applied “exactly” through LU factorization to achieve convergence).

An ideal iterative method would provide robust convergence without the need to assemble a large sparse matrix, allowing larger problems to be solved on more memory constrained computer platforms such as graphics processing units. Additionally such an iterative method should not have large triangular solves built in because these tend to negate the performance benefits of wide vector units now common in modern computing architectures [122]. Triangular solvers would be for example necessary in the cases of [25] and [12] where LU factors of domain decomposition preconditioners are explicitly stored and applied at each step. In this thesis I further build on domain decomposition based preconditioning for plane-wave DG, but I will later provide a means to apply these preconditioners in a completely matrix-free fashion and without the need for triangular solves. The domain decomposition method is mathematically formulated in section 2.3, but details of matrix-free application are deferred until chapter 3.

2.1 Helmholtz Equation

The main form of the Helmholtz equation that will be investigated here can be derived from the acoustic wave equation

\[
\begin{align*}
\Delta \hat{u} - \frac{1}{c^2} \frac{\partial^2 \hat{u}}{\partial t^2} &= \hat{f} \quad \text{in } \Omega \\
\mathbf{n}_\Omega \cdot \nabla \hat{u} - \frac{1}{c} \frac{\partial \hat{u}}{\partial n} &= 0 \quad \text{on } \Gamma_R \\
\hat{u} &= \hat{g} \quad \text{on } \Gamma_D
\end{align*}
\]

(2.1.1)

where \(\mathbf{n}_\Omega\) is the outward pointing normal for the domain \(\Omega\), \(c\) is the speed of sound, the
second equation is a boundary condition which approximates the effect of outgoing waves that are not reflected at the boundary $\Omega$ [48], and the third equation is a Dirichlet boundary condition indicating “sound hard” scattering. It will be understood throughout the thesis that $\Omega = \Gamma_D \cup \Gamma_R$ with the possibilities that either $\Gamma_D = \emptyset$ or $\Gamma_R = \emptyset$.

We assume that $\hat{u}$ and the forcing terms $\hat{f}, \hat{g}$ are periodic in time with a pre-specified frequency. This means that there exists a wavenumber $\kappa$ such that by taking the frequency $\nu = \kappa c$ we get for some amplitude functions $u, f, g$

\[
\hat{u}(t, x) = \exp(i\nu t) u(x) \quad (2.1.2)
\]
\[
\hat{f}(t, x) = \exp(i\nu t) f(x) \quad (2.1.3)
\]
\[
\hat{g}(t, x) = \exp(i\nu t) g(x) \quad (2.1.4)
\]

By taking the expression 2.1.2 and substituting it into 2.1.1 we obtain the Helmholtz equation which describes the amplitude $u$

\[
\begin{cases}
-\Delta u - \kappa^2 u = f & \text{in } \Omega \\
\mathbf{n}_\Omega \cdot \nabla u + i\kappa u = 0 & \text{on } \Gamma_R. \\
u = g & \text{on } \Gamma_D 
\end{cases} \quad (2.1.5)
\]

From the classical theory of partial differential equations a difficulty should be immediately apparent here if the wavenumber $\kappa$ is large, that is the wavenumber scaled term $\kappa^2 u$ shifts the spectrum of the well-behaved Laplacian operator so that the problem becomes indefinite. This complicates theoretical analysis (see e.g. recent work [69] establishing wavenumber explicit stability estimates) , and also has numerical consequences ( see [50] for a review of the state-of-the-art in quantifying this difficulty with respect to the wavenumber parameter $\kappa$). Theoretically the chief
difficulty comes from the fact that the wavenumber $\kappa$ enters the analysis as a constant which can easily dominate all other controlled quantities, making it hard or impossible to draw conclusions about things such as e.g. the stability of a numerical algorithm. The discontinuous Galerkin method to be introduced in section 2.2 overcomes this difficulty somewhat through its penalty terms, which among other things guarantee the resulting system is always solvable, independent of the wavenumber. The chief benefit for the purposes of this thesis however is that the discontinuous Galerkin method permits discontinuous solutions, and therefore can flexibly include non standard approximations such as plane wave expansions. The resulting plane-wave discontinuous Galerkin method is the topic of the next section.

2.2 Plane-wave discontinuous Galerkin

The discontinuous Galerkin method begins with a mesh $\mathcal{T} = \{T \mid \text{is a triangle}\}$ such that the domain $\Omega$ is “well approximated” by its whole union, however to avoid complications that are outside the scope of this thesis it will be assumed that $\Omega = \bigcup \mathcal{T}$.

With a mesh that faithfully represents the geometry. We can further permit additional generality by allowing the wavenumber parameter $\kappa$ to vary on the domain in such a way to be piecewise constant on the mesh, (that is: spatially variable, but when restricted to any triangle in $\mathcal{T}$ is constant). Therefore for this additional generality the notation $\kappa_D \ (D \in \mathcal{T})$ will indicate the piecewise constant wavenumber.

On each mesh element $D \in \mathcal{T}$ we multiply the first equation in (2.1.5) by the complex conjugate of a function $v$ and integrate by parts twice to obtain the equation
The key step in any Trefftz type method is to now assume that \( v \) solves the homogeneous Helmholtz equation (that is, taking \( f = 0 \) in (2.1.5)). By eliminating the volume integral in equation (2.2.1) above, this assumption reduces to

\[
\int_{\partial D} \mathbf{n}_D \cdot \nabla u - \int_{\partial D} (\mathbf{n}_D \cdot \nabla u) \overline{v} = \int_D f \overline{v}
\]  

(2.2.2)

This assumption on the test function \( v \) is satisfied by choosing its representation to be of the form

\[
v(x) = \sum_{j=1}^{N_w} v_j \exp(i \kappa_D \mathbf{d}_j \cdot x)
\]  

(2.2.3)

for \( v_j \in \mathbb{C}, (j = 1, \ldots, N_w) \), \( ||\mathbf{d}|| = 1 \), \( (j = 1, \ldots, N_w) \), and \( i = \sqrt{-1} \).

and a Galerkin method is obtained by choosing the representation of \( u \) to be the same as the test function. The method becomes discontinuous when we furthermore allow
that for each element $D \in \mathcal{T}$ the form of the expansion (2.2.3) can use different $N_w$ and $d_j$. I will now make these ideas more precise.

In each element $D \in \mathcal{T}$ I assume the solution $u$ can be represented as a linear combination

$$u(x) = \sum_{j=1}^{N_w^D} u_j^D \exp(i \kappa_D d_j^D \cdot x)$$

(2.2.4)

where ambiguity may arise I will refer to $u$ as $u^D$ when speaking in the context of a specific element $D$, but in most cases it is clear from context (e.g. by the domain of an integral) which element is under consideration and so this extra notation will be omitted. Furthermore the wave direction vectors $d_j^D$ for each element $D \in \mathcal{T}$ will always be given as equally spaced in the following way

$$d_j^D = \left( \cos \left( \frac{2 \pi j - 1}{N_w^D} \right), \sin \left( \frac{2 \pi j - 1}{N_w^D} \right) \right).$$

(2.2.5)

In order to allow inter-element communication, which equation (2.2.2) does not currently support, I modify that equation and replace occurrences of $u$ and $n_D \cdot \nabla u$ with “hatted” approximations $\hat{u}$ and $\hat{n}_D \cdot \nabla \hat{u}$ which serve to approximate inter-element boundary conditions. With this modification to (2.2.2) we can state the DG formulation as: find coefficients $u_j^D \in \mathbb{C}$ ($j = 1, \ldots, N_w^D$) such that with $u$ expressed as in (2.2.3) and for all $D, (j = 1, \ldots, N_w^D)$ we have

$$\int_{\partial D} \hat{u}(x)n_D \cdot d_j^D i \kappa_D \exp(i \kappa_D d_j^D \cdot x) - \int_{\partial D} \left( \hat{n}_D \cdot \nabla \hat{u} \right) \exp(i \kappa_D d_j^D \cdot x) = \int_D f \exp(i \kappa_D d_j^D \cdot x)$$

(2.2.6)

where, following [71] I define the hatted variables to be
\[
\mathbf{n}_D \cdot \nabla u = \begin{cases}
\{\nabla u\} - \alpha i k [u] & \text{on } \partial D \setminus (\Gamma_D \cup \Gamma_R) \\
\nabla u - (1 - \delta)(\nabla u + i ku) & \text{on } \partial D \cap \Gamma_R \\
\nabla u - \alpha i ku & \text{on } \partial D \cap \Gamma_D 
\end{cases}
\] (2.2.7)

and

\[
\hat{u} = \begin{cases}
\{u\} - \frac{\beta}{ik} [\nabla u] & \text{on } \partial D \setminus (\Gamma_D \cup \Gamma_R) \\
\frac{\delta}{ik} \mathbf{n}_D \cdot \nabla u + u & \text{on } \partial D \cap \Gamma_R \\
g & \text{on } \partial D \cap \Gamma_D 
\end{cases}
\] (2.2.8)

where the jump \([\cdot]\) and average \(\{\cdot\}\) are defined on edges shared by elements \(D^+, D^- \in \mathcal{T}\) by

\[
[u] = u^{D^+} - u^{D^-} \\
\{u\} = \frac{1}{2}(u^{D^+} + u^{D^-})
\] (2.2.9, 2.2.10)

and finally the penalty parameters \(\alpha, \beta, \delta\) are given as

\[
\alpha = (1 - s) \frac{1}{2} + sa \frac{h}{h_D} \\
\beta = (1 - s) \frac{1}{2} + sb \frac{h}{h_D} \\
\delta = \frac{1}{2}
\] (2.2.11, 2.2.12, 2.2.13)

where \(s \in [0, 1]\) is chosen to interpolate between the Ultra-Weak Variational Formulation (UWVF) penalties [32] and more standard DG penalties [71], and \(a, b\) are globally constant variables the size of which determines how much edge length can contribute to the penalties. For the remainder of the thesis the values of \(a, b\) will be fixed at
$a = b = 1/2$, and $h_D$ is the edge length function for the triangle $D \in \text{mesh}$ and $h$ is the largest triangle edge length out of the whole mesh. These choices of $a, b$ reduces the penalties to close approximations of the UWVF method on the largest elements while still giving larger penalties on smaller elements, if they exist in the mesh. The DG penalties are generally expected to provide better convergence behavior in the presence of locally refined regions of a mesh, this was demonstrated by Hiptmair et al. in [109] and [71], but this thesis will explore their impact numerically on much larger test cases than those considered in the literature to date.

The formulation (2.2.6) ultimately results in a system of equations of the form $Ax = b$ where $A$ is a linear operator, the action of which may be computed by algorithm 1

**Algorithm 1** Evaluation of PWDG Operator

```plaintext
procedure PWDG($\mathcal{T}, \kappa, N_w, u$)

Note: hatted variables are defined in (2.2.8),(2.2.7).

Assume: $g = 0$ in numerical fluxes (2.2.8),(2.2.7).

for $D \in \mathcal{T}$ do

for $j = 1, \ldots, N_w^D$ do

$v_j^D = \mathbf{n}_D \cdot \mathbf{d}_j^D \int_{\partial D} (\hat{u}) (x) \frac{i\kappa_D \exp (i\kappa_D \mathbf{d}_j^D \cdot x)}{(\mathbf{n}_D \cdot \nabla u)} (x) \exp (i\kappa_D \mathbf{d}_j^D \cdot x)$

$v_j^D \leftarrow v_j^D - \int_{\partial D} (\mathbf{n}_D \cdot \nabla u) (x) \frac{i\kappa_D \exp (i\kappa_D \mathbf{d}_j^D \cdot x)}{(\mathbf{n}_D \cdot \nabla u)} (x) \exp (i\kappa_D \mathbf{d}_j^D \cdot x)$

end for

end for

return $v$

end procedure
```

note that the assumption $g = 0$ in algorithm 1 is merely to enable stating the system
as a system of linear equations, rather than as a nonlinear system such as $A(u) = 0$.

By linearity of the integrals in (2.2.6) we can separate the boundary data $g$ and source data $f$ into a separate term which becomes the right-hand-side of the linear system, computed by algorithm 2.

**Algorithm 2** Evaluation of PWDG Source Term

```plaintext
procedure SOURCE($\mathcal{T}, \kappa, N_w, v$)

Note: hatted variables are defined in (2.2.8), (2.2.7).

Define $u = 0$.

for $D \in \mathcal{T}$ do

  for $i = 1, \ldots, N_w^D$ do

    $v_j^D = n_D \cdot d_j^D \int_{\partial D} (\hat{u})(x) i\kappa_D \exp(i\kappa_D d_j^D \cdot x)$

    $v_j^D \leftarrow v_j^D - \int_{\partial D} (n_D \cdot \nabla u)(x) \exp(i\kappa_D d_j^D \cdot x)$

  end for

end for

end procedure
```

Although this formulation is consistent and stable, as proven in the earliest work on the subject [32], all work to date has not yet accounted for the fact that although plane-waves have excellent approximation properties when considering solutions of the homogeneous Helmholtz equation (this is the case $f = 0$ in (2.1.5)), the convergence rate degrades rapidly when $f$ contains a function that is not itself a solution of the homogeneous Helmholtz equation (with same wavenumber as the plane waves themselves).

A recent paper by Howarth et al. [39] however explores the technique of *source extraction* in the context of another Trefftz DG method which uses Bessel function
expansions rather than plane-wave expansions. Although Bessel function expansions are not considered in this thesis because their matrix-free implementations require significantly more special function evaluations than plane-waves, the idea of source extraction can be straightforwardly implemented in the context of plane-wave DG and this is done below in section 2.2.1.

2.2.1 Source Extraction

In [39] the observation is made that solving the Helmholtz equation for the case of a nonzero forcing function \( f \) which does not itself satisfy the same Helmholtz equation leads to difficulties in recovering the optimal convergence rates promised by Trefftz-type approximations (which plane-waves belong to). In that paper this is solved by taking the solution \( u \) and writing it as

\[
  u = u^I + u^S
\]  

(2.2.14)

where \( u^S \) solves the homogeneous Helmholtz equation \( (f = 0) \) and \( u^I \) solves the inhomogeneous Helmholtz equation \( (f \neq 0) \), both without any additional boundary conditions. This equation can then be used to re-express the boundary conditions of (2.1.5), which can then be reintegrated into a new DG formulation. In this thesis I will only consider the case of a point source, as this one has caused significant difficulty for Trefftz methods in the past, and also because it tends to be a practically useful source term.

Fortunately the point source term for the Helmholtz equation already has a well understood solution, since it is the Green’s function for the Helmholtz equation - which in two dimensions is given by
where $H_0^1$ is a Hankel function following the notation of Abramowitz and Stegun [1, Ch.9]. The next task is to find a way to incorporate the equation (2.2.14) into the DG formulation (2.2.6) so that it only involves solving an homogeneous Helmholtz equation, with a possible solution recovery step at the end where we add back in $u_I$.

I proceed in the following way. First I assume that the mesh $\mathcal{T}$ will be partitioned into two subsets, $\mathcal{T}_C$ and $\mathcal{T}_{NC}$ where $\mathcal{T}_C$ is a collection of triangles on which we wish to subtract away the free-space inhomogeneous solution $u_I$ and $\mathcal{T}_{NC}$ where we do not (I have named these so that: $\mathcal{T}_C$ contains the point source, and $\mathcal{T}_{NC}$ does not contain the point source). In the case of the point source, subtracting $u_I$ from both sides of the Helmholtz equation (2.1.5) in $\mathcal{T}_C$ has the effect of turning the equation into a homogeneous Helmholtz equation for which one must solve for the unknown $u - u_I$. Since I have assumed the source $f$ is a point source, in the elements from the partition $\mathcal{T}_{NC}$ we also are solving a homogeneous Helmholtz equation, but instead for the unmodified $u$. Therefore the way to incorporate equation (2.2.14) into the DG formulation is the function $u_I$ will enter the jump and average terms given by (2.2.9). From within the partition $\mathcal{T}_C$, when retrieving neighboring element data it will be necessary to subtract $u_I$ to remain consistent with the fact that we are solving for $u - u_I$ inside $\mathcal{T}_C$, and from within the partition $\mathcal{T}_{NC}$ it will be necessary to add $u_I$ to remain consistent with the fact that in $\mathcal{T}_{NC}$ we are solving for $u$ without modifications. These comments are summarized by the following updated formulas for the jumps and averages.
what results from this modification is a purely homogeneous PWDG system with a different source term than if one were to directly apply (2.2.6) with \( f \) being a source term. To recover the correct solution in the partition \( \mathcal{T}_C \) it is enough to simply add the Green function \( u^I \) to the result of solving the source extraction PWDG equations.

\[
\begin{align*}
[u^D^-] &= (u^{D+} - u^I) - u^{D^-} & \text{if } D^- \in \mathcal{T}_C & \text{(2.2.16)} \\
[u^D^-] &= u^{D+} - (u^{D^-} + u^I) & \text{if } D^- \in \mathcal{T}_{NC} & \text{(2.2.17)} \\
\{u^D_-\} &= \frac{1}{2}((u^{D+} - u^I) + u^{D^-}) & \text{if } D^- \in \mathcal{T}_C & \text{(2.2.18)} \\
\{u^D_-\} &= \frac{1}{2}(u^{D+} + (u^{D^-} + u^I)) & \text{if } D^- \in \mathcal{T}_{NC} & \text{(2.2.19)} \\
\end{align*}
\]

Figure 2.2.2 : Diagram illustrating source extraction. Inside the gray shaded elements the Helmholtz equation Green’s function is subtracted from the equation in order to make it homogeneous, and therefore PWDG becomes a better candidate for solving it.

The ability to accurately represent a point source already opens the door to more practical applications of plane wave DG methods, but even a very high resolution of a point source can be polluted by improper handling of boundary conditions. The
next section shows how to carefully handle this with minimal modification to the DG formulation already introduced.

2.2.2 Perfectly Matched Layer

Since a large amount of Helmholtz simulations are in truncated domains primarily because solving on an unbounded domain is impossible, one has to simulate the effect of an unbounded domain by imposing boundary conditions which minimize reflections. Using the first order conditions imposed already in (2.1.5) does help with this somewhat, but still produces reflections. The technique introduced here augments this first order condition with a buffer layer surrounding the computational domain. This buffer layer will have a modified wavenumber $\kappa$ so that the waves which enter it will begin to decay rapidly, therefore once the waves actually reach the first order radiation condition boundary the reflection it causes will be completely negligible.

The technique described above is not new. It is known as the “Perfectly Matched Layer,” which is often derived as a complex coordinate stretching ever since the seminal work by Chew et al. [37], and indeed it is derived this way for the UWVF method by Huttunen et al. in [83]. The approach applied here however will use the limitation of using piecewise constant wavenumbers to provide a simple but effective absorbing layer that requires no change to the existing DG formulation (2.2.6).

I proceed as follows: Enclose the existing truncated domain $\Omega$ in a large box $B$, and generally speaking the larger this box the better the absorbing layer will prevent numerical reflections, but it must also consider practical tradeoffs because of memory limitations. Inside this large box I will extend the wavenumber $\kappa$ from (2.1.5) in the following way:
Figure 2.2.3: Diagram illustrating absorbing layer. A larger box encloses the computational domain and its wavenumber is modified so that incoming waves are captured (not reflected at the boundary between the two boxes) and then gradually dissipated.

\[ \kappa(x) = d(\Omega, x) \kappa_D(x) + i\sigma \frac{1}{\kappa_D(x)} \quad (x \in B \setminus \Omega) \quad (2.2.21) \]

where \( \sigma \) is a “tunable” parameter, which in this thesis is defined to be

\[ \sigma = 1/\text{diam}(\Omega) = 1/\sup\{||x - y|| \mid x, y \in \Omega\} \quad (2.2.22) \]

although there is considerable room for improvement in the selection of \( \sigma \), studied further in e.g. [22], this is not otherwise explored here.

The functions \( d(\Omega, x), D(x) \) are defined respectively to be the distance from \( x \) to the computational domain \( \Omega \) and the element \( D \in T \) that is “nearest” to \( x \). In this thesis, the nearness of elements is determined by using their barycenters and choosing the element with barycenter closest to \( x \).
The next step for incorporating this modified wavenumber into the DG formulation (2.2.6) is to replace $\Omega \leftrightarrow B$, and assume again that $\Omega = \bigcup \mathcal{T}$. Then define

$$\kappa_D = \frac{1}{\int_D 1} \int_D \kappa(x)$$

for all $D \in \mathcal{T}$. In words: approximate the modified wavenumber by taking it to be element-wise averages. The DG formulation given in (2.2.6) may now be used without modification, simply by using this modified wavenumber in place of the original when in the extended PML region. As a remark: a poorly refined mesh may yield a correspondingly bad PML performance, since the piecwise average approximation used in equation (2.2.23) becomes worse in the limit of large mesh elements. This effect is not studied in this thesis since in the cases of highly variable wavenumbers will already require a well refined mesh.

The PML introduced in this section is roughly equivalent to that of Huttunen et al. [83] in the case of the lowest order. This was a deliberate choice so as to allow integrals to still be computable analytically, without expensive quadrature rules. A further complication introduced by this PML which was not addressed in [83] is the fact that the analytic formulas for integrals in the PML region can contain severe underflow and overflow terms, which ultimately result in NaNs propagating throughout the solve and ruining the simulation. I address this with a new basis scaling to slightly modify the solution expansion (2.2.3), and these details are given in the following section.

### 2.2.3 Numerical Stability and Efficient Formulas

In this section I show how all of the integrals from the DG formulation (2.2.6) can be computed analytically without quadratures. This will be an important feature of
this method because quadratures increase the number of special function evaluations significantly.

First note that by the representation of \( u \) as a linear combination of plane-waves, all integrals in (2.2.6) become linear combinations of integrals over edges of the triangles \( D \in \mathcal{T} \). Consider therefore a single edge \( e \) with vertices \( \mathbf{v}_1, \mathbf{v}_2 \) and represented by the parameterization

\[
\gamma (t) = (1 - t) \mathbf{v}_1 + t \mathbf{v}_2 \quad (t \in [0, 1])
\]

\[
\int_e \exp(i \kappa_1 \mathbf{d}_1 \cdot \mathbf{x}) \exp(i \kappa_2 \mathbf{d}_2 \cdot \mathbf{x}) = \int_e \exp \left( i \kappa_1 \mathbf{d}_1 + i \kappa_2 \mathbf{d}_2 \cdot \mathbf{x} \right) \quad (2.2.25)
\]

\[
= h \exp(\mathbf{d} \cdot \mathbf{v}_1) \int_0^1 \exp(\mathbf{d} \cdot (\mathbf{v}_2 - \mathbf{v}_1) t) \, dt \quad (2.2.26)
\]

\[
= h \exp(\mathbf{d} \cdot \mathbf{v}_1) \frac{\exp(\mathbf{d} \cdot (\mathbf{v}_2 - \mathbf{v}_1)) - 1}{\mathbf{d} \cdot (\mathbf{v}_2 - \mathbf{v}_1)} \quad (2.2.27)
\]

where to make the formulas compact I have defined

\[
h = ||\mathbf{v}_2 - \mathbf{v}_1|| \quad (2.2.28)
\]

\[
\mathbf{d} = i \kappa_1 \mathbf{d}_1 + i \kappa_2 \mathbf{d}_2 \quad (2.2.29)
\]

following somewhat the notation for similar derivations in Gittelson’s master’s thesis [38], although the definition of \( \mathbf{d} \) here takes into account possibly different wavenumbers.

Some numerical points must be made now about (2.2.25). First note that if the edge is orthogonal or almost orthogonal to \( \mathbf{d} \) then the expression given above becomes of the form

\[
\frac{\exp(z) - 1}{z} \quad (2.2.30)
\]
for small $z$, the numerator of which is a well-known expression that is difficult to compute to high relative accuracy because it involves subtracting two very similar floating point numbers. The programming language Julia has an accurate implementation of “expm1” which computes this and incurs very little relative error even with the argument $z$ is small. This function however is not implemented in most graphics processing unit programming environments because they generally lack standardized complex numbers, and so to get similar numerical accuracy as the Julia implementations the source code of Julia was consulted to implement “expm1” for the threading model used in this thesis, OCCA, for which more details may be found in chapter 4.

An additional numerical challenge that arises when attempting to use (2.2.25) directly comes from the constant factor of $\exp(d \cdot v_1)$, which if the wavenumber is real over the whole domain can be stably computed because in that specific case the computation reduces to evaluating a bounded periodic function. Problems arise when a PML layer is introduced however because the nonzero imaginary components ultimately make the expression $d \cdot v_1$ have a large and positive real part, leading to overflow. This does not happen in the integrand of (2.2.25) because the possibly large real part of $d$ becomes scaled by $h = ||v_2 - v_1||$ which should already be comensurate to the choice of imaginary part by the PML region, since it depends on the size of the domain. Therefore to achieve a similar effect in the constant factor I introduce a special scaling of the basis.

Suppose that $D \in T$ and that $v_C$ is the barycenter of $D$. Then the function

$$x \rightarrow \exp(i\kappa_D d \cdot (x - (1/2)v_C)) = \exp(i\kappa_D d \cdot x) \exp(-(1/2)i\kappa_D d \cdot v_C) \quad (2.2.31)$$

still solves the homogeneous Helmholtz equation, since it is only a constant factor to another solution of a Helmholtz equation. Therefore a basis scaled in this way can still be used as a Trefftz basis, and if we choose this scaling, then equation (2.2.25)
becomes

\[ h \exp \left( d \cdot (v_1 - \frac{1}{2} (v_{C_1} + v_{C_2})) \right) \exp \left( d \cdot (v_2 - v_1) \right) - 1 \]

(2.2.32)

and now the problematic constant term \(d \cdot (v_1 - \frac{1}{2} (v_{C_1} + v_{C_2}))\) also carries a scaling of \(h\), controlling the entire expression now in terms of the mesh refinement level. Thus overflow can be explicitly avoided by ensuring the PML parameter scales with the inverse of the diameter of the domain.

### 2.2.4 Choice of Local Basis

One final point that is not completely answered in the literature is how to choose \(N^D_w\), the number of plane waves per element. Certain heuristics exist, such as to choose \(N^D_w\) such that the condition number of the mass matrix for the element \(D \in \mathcal{T}\) does not exceed a certain threshold, this is the criteria used by Huttunen et al. in [85], but outside of this heuristic not many others have been explored that are designed to control the condition number of the full operator.

In the context of matrix-free methods, computing the condition numbers of local mass matrices is a challenging task. Therefore I focus on a simple geometric heuristic, namely that for a specific wavenumber \(\kappa\) two plane wave directions

\[ d_1 = (\cos(\theta), \sin(\theta)) \]

\[ d_2 = (\cos(\theta + \tau), \sin(\theta + \tau)) \]

where \(\tau\) represents the shift which would give the “next” plane wave direction. By a Taylor expansion one finds

\[ d_2 = d_1 + \tau b + O(\tau^2) \]
where \( \mathbf{b} = (-\sin(\theta), \cos(\theta)) \). To go further I will assume \( B = [0, h] \times [0, h] \), that \( u(\mathbf{x}) = \exp(i\kappa \mathbf{d}_1 \cdot \mathbf{x}) \), \( v(\mathbf{x}) = \exp(i\kappa \mathbf{d}_2 \cdot \mathbf{x}) \), and compute

\[
||u||_B^2 = h^2 \\
||v||_B^2 = h^2
\]

where \( ||f||_B = \sqrt{\int_B f^2} \). Finally defining the inner product between \( u \) and \( v \) to be \((u, v) = \int_B u \overline{v} \) a straightforward computation shows that by neglecting the higher order term in the taylor expansion of \( \mathbf{d}_2 \) we obtain

\[
||(u, v)|| = \left| \frac{\exp(-i\tau \kappa b_x h) - 1}{\tau \kappa b_x} \right| \frac{||\exp(-i\tau \kappa b_y h) - 1||}{\tau \kappa b_y}
\]

so that the “angle” \( \theta(u, v) \) between \( u \) and \( v \) may be approximated as

\[
\cos(\theta(u, v)) = \frac{||u|| ||v||}{|(u, v)|} = \frac{\exp(z b_x) - 1}{z b_x} \frac{\exp(z b_y) - 1}{z b_y}
\]

where \( z = \tau \kappa h \). Since \( z \to 0 \) as \( \tau \kappa h \to 0 \) the cosine of the angle between \( u \) and \( v \) becomes 1 in this limit, and so this measures how linearly independent \( u \) and \( v \) are from each other. Therefore to maintain good conditioning I will impose

\[
\frac{2\pi}{N^D_w} h_D \kappa_D > K
\]

for some constant \( K > 0 \) and for all \( D \in \mathcal{T} \). Since \( \mathcal{T} \) consists of triangles and not squares, I will choose 
\( h_D = \sqrt{h_1^2 + h_2^2 + h_3^2} \) where \( h_i \) are the edge lengths of the triangle \( D \). Hence, gathering all constants into \( C \) the condition can be expressed

\[
N^D_w = Ch_D \kappa_D. \tag{2.2.33}
\]

The numerical impact of this adaptive choice of \( N^D_w \) is investigated further in chapter 5 section 5.3.2 where different values of \( C \) are investigated.
2.3 Additive Schwarz Domain Decomposition

An additional challenge not yet addressed for the ultimate matrix-free method is that of preconditioning. Since no matrix will be stored for the operator given by equation (2.2.6), it will rest on a Krylov iterative method to solve the resulting system of linear equations. This thesis focuses on the Generalized Minimum Residual [118] (GMRES) method and methods which derive from GMRES, and while it is discussed in further detail in chapter 3, it will be important here to understand some of its basic properties in order to exploit the mathematical formulation of PWDG to yield a matrix-free preconditioner. While current understanding of iterative methods applied to non-Hermitian systems is limited in the general case, it is still known that for normal matrices it is highly beneficial to the GMRES algorithm to have “clustered” eigenvalues, see e.g. the discussion by Nachtigal, Trefethen, and Reddy in [104] demonstrating this and comparisons to other methods. One problem however is that the PWDG method for the Helmholtz equation does not generate a clustered spectrum, one can see this for example in figure 2.3.1

One type of preconditioner that has received attention recently in the literature for PWDG methods are domain decomposition preconditioners (see e.g. [12] and [25] for recent work on this). The basic idea of a domain decomposition preconditioner is to partition the original mesh $\mathcal{T}$ into submeshes $\mathcal{T}_1, \ldots, \mathcal{T}_L$ such that $\mathcal{T} = \bigcup_{l=1}^{L} \mathcal{T}_l$. On each mesh partition separately PWDG is reformulated. What mainly differentiates the different domain decomposition methods is what kind of boundary conditions are imposed between the partitions $\mathcal{T}_l$ and what-if any-order the subdomain problems are solved and reincorporated into the existing global solution on $\mathcal{T}$. The question of how the partitions $\mathcal{T}_1, \ldots, \mathcal{T}_L$ are generated is of importance as well, but it is not further explored by this thesis except to say that the software package MGridGen [102] is
Figure 2.3.1: Example PWDG spectrum. Note that the eigenvalues are approximately distributed evently which GMRES has issues handling. To compute the eigenvalues using the PWDG algorithm I first obtain a linear operator $A$ by defining $Au = \text{PWDG}(\mathcal{T}, \kappa, N_w, u)$. From this one can obtain eigenvalues of $A$ using an iterative algorithm, but in this case I “assemble” $A$ into a dense format matrix by defining $B_i = Ae_i$, where $e_i$ is the $i$-th column of the identity matrix.
used. The type of domain decomposition used here is known as “additive Schwarz” domain decomposition, because all of the subdomains are solved in parallel and not reincorporated to the full solution until they are all complete.

The main purpose of this section will be to formulate the Schwarz domain decomposition operator rigorously and then to follow up with analyzing the spectra of the resulting preconditioned operators. The results will show significant eigenvalue clustering that improves as the number of partitions decrease, corresponding to the fact that fewer partitions yield a closer approximation to the whole operator.

Given a mesh $\mathcal{T}$ (example pictured in figure 2.3.2) an approximation to $u$ computed by way of the PWDG formulation (2.2.6) by first partitioning $\mathcal{T}$ into a sequence of submeshes $\mathcal{T}_1, \ldots, \mathcal{T}_L$ (pictured in figure 2.3.3)

![Figure 2.3.2 : Example mesh of a U-shaped scattering domain. This mesh was generated using GMSH [59].](image)

and then modifying the formulation (2.2.6) to read: find coefficients $u_j^P \in \mathbb{C}$ ($j =$
Figure 2.3.3: Example partitioning of mesh using $L=116$ partitions. This partition was generated by making repeated calls to MGridGen [102] and stopping once the number of partitions was below a threshold.
1, \ldots, N_w^D) such that with \( u \) expressed as in (2.2.3) and for all \( D \in \mathcal{T}, (j = 1, \ldots, N_w^D) \) we have

\[
\int_{\partial D} \hat{u}(x) \mathbf{n}_D \cdot d_j^D i\kappa_D \exp(i\kappa_D d_j^D \cdot x) - \int_{\partial D} \left( \mathbf{n}_D \cdot \nabla u \right) \exp(i\kappa_D d_j^D \cdot x) = \int_{D} f \exp(i\kappa_D D_j^D \cdot x)
\]

(2.3.1)

The change comes from the definition of numerical fluxes, which following the comments on domain decomposition methods for the Helmholtz equation found in [49], will be modified to use Robin boundary conditions on the boundaries of mesh partitions \( \mathcal{T}_l \). Mathematically this means

\[
\mathbf{n}_D \cdot \nabla u = \begin{cases} 
\{ \nabla u \} - \alpha i \kappa [u] & \text{on } \partial D \setminus (\Gamma_D \cup \Gamma_R) \\
\nabla u - (1 - \delta) (\nabla u + iku) & \text{on } \partial D \cap \Gamma_R \\
\nabla u - (1 - \delta) (\nabla u + iku) & \text{on } \partial D \cap \partial \Omega \setminus \Gamma_R \\
\nabla u - \alpha i ku & \text{on } \partial D \cap \Gamma_D 
\end{cases}
\]

(2.3.2)

and

\[
\hat{u} = \begin{cases} 
\{ u \} - \frac{\delta}{i\kappa} [\nabla u] & \text{on } \partial D \setminus (\Gamma_D \cup \Gamma_R) \\
u - \frac{\delta}{i\kappa} \mathbf{n}_D \cdot \nabla u + u & \text{on } \partial D \cap \Gamma_R \\
u - \frac{\delta}{i\kappa} \mathbf{n}_D \cdot \nabla u + u & \text{on } \partial D \cap \partial \Omega_l \setminus \Gamma_R \\
g & \text{on } \partial D \cap \Gamma_D 
\end{cases}
\]

(2.3.3)

where \( \Omega_l = \bigcup \mathcal{T}_l, (l = 1, \ldots, L) \).

Although it has not been explicitly described yet, the formulation given by the domain decomposition formulation (2.3.1) can be represented by a system of linear equations \( Gx = b \) where \( b \) collects all boundary and source data. The means of computing \( Gx \) for any vector \( x \) is described in detail in chapter 3 where the linear
solving are also presented in further detail. The main thing important to know here is that if one similarly defines $Ax = b$ to be the linear system arising from the non-domain decomposed system (2.2.6) (and it is this system which is of interest to solve), then the system can be transformed into an equivalent one with a much more favorable spectrum for GMRES by taking $F = G^{-1}$ and then solving for $y$ in

$$AFy = b \quad (2.3.4)$$

from which a solution to $Ax = b$ can be recovered by taking $x = F^{-1}y$. The operator $G$ can be computed with little change from the PWDG operator evaluation algorithm 1, and is given in algorithm 3.

**Algorithm 3 Evaluation of Schwarz PWDG Operator**

```plaintext
procedure SCHWARZPWDG(T, \kappa, N_w, u)

Note: Hatted variables are defined in (2.3.3),(2.3.2)

Assume: $g = 0$ in numerical fluxes (2.3.3),(2.3.2)

for $D \in T$ do

for $j = 1, \ldots, N^D_w$ do

$v^D_j = n^D \cdot d^D_j \int_{\partial D} \hat{u}(x)i\kappa_D \exp \left(i\kappa_D d^D_j \cdot x\right)$

$v^D_j \leftarrow v^D_j - \int_{\partial D} \left(n^D \cdot \nabla u\right)(x) \exp \left(i\kappa_D d^D_j \cdot x\right)$

end for

end for

return $v$

end procedure
```

What follows will be a study of the spectrum produced by the matrix $A$ as compared to the spectrum of $AF$ for different levels of domain decomposition. The ideal
situation is that $AF$ has clustered eigenvalues, and that will be indeed what is observed.

Three cases will be explored below. One case will be a simple box with pure robin boundary conditions and an everywhere constant wavenumber, another case will be a box but two (real) wavenumbers, and the last case will be a scattering problem from a square cavity with internal Dirichlet boundary conditions and PML boundary conditions for radiation. These three cases are illustrated in figure 2.3.4

Figure 2.3.4 : Diagrams illustrating test cases. On the left is a homogeneous box where $\kappa = 11$ and the boundary conditions are purely Robin. In the middle is a box with piecewise constant wavenumber. The top half of the box uses one wavenumber $\kappa_1$ and the bottom half uses another $\kappa_2$. The third box contains a smaller box cavity such that the boundary of the smaller box uses Dirichlet boundary conditions equal to 0, corresponding to a “sound hard” scatterer. The outer boundary uses PML boundary conditions.

next are the different levels of Schwarz subdomains and the associated spectrum for the operator $AF$. Note that for all test cases the first mesh and spectrum corresponds to $A$ unpreconditioned by $F$ for comparison purposes.
2.3.0.1 Homogeneous Box Test Case

The first test case under consideration is one of the simplest, but it is one which often forms the building block to more complicated problem. It is a box that using an everywhere constant wavenumber $\kappa = 11$ and the outer boundary conditions are defined to be Robin, giving a low order approximation of outgoing waves. The mesh $\mathcal{T}$ with its partitions $\mathcal{T}_l, \ldots, \mathcal{T}_L$ are shown in figure 2.3.5. Taking the number of plane-waves per element to be 4 on each element, that is $N^D_w = 4$ for all $D \in \mathcal{T}$ we can assemble a dense matrix form of the PWDG operator by defining $Ax = \text{PWDG}(\mathcal{T}, \kappa, N_w, x)$ and then computing the operator column-by-column by taking $x$ to be the columns of the identity operator. This can be similarly done for the approximation $G$ to $A$ formed by the associated Schwarz levels, after which the eigenvalues of $AG^{-1}$ can be computed using a dense matrix eigenvalue computation routine. The result of this computation is given in figure 2.3.6.

In figure 2.3.6 there is a significant clustering of eigenvalues as the number of mesh partitions decrease (which corresponds to the geometric size of the partitions themselvs increasing). The test case used to generate this however is somewhat simple and furthermore used a small wavenumber, and so many of the problems which are usually evident in preconditioning the Helmholtz equation are not present. This will be addressed in the next test case, the heterogeneous box.

2.3.0.2 Heterogeneous Box Case

To consider a more complicated example I construct again a box, but the top half of the box will use one wavenumber $\kappa_1$ and the bottom half another wavenumber $\kappa_2$. I furthermore investigate a more challenging case where one of the wavenumbers is large, that is I will define $\kappa_1 = 31$ and $\kappa_2 = 71$. The mesh and partition sequence is
Figure 2.3.5: Homogeneous box test case Schwarz levels. The mesh was generated using the GMSH software [59], and the partitions were generated by repeated calls to MGridGen [102]. The top figure is the original mesh, the middle figure is $L = 400$ partitions, and the bottom figure is $L = 8$ partitions.
Figure 2.3.6: Eigenvalues of homogeneous box case with and without Schwarz preconditioning. The top spectrum corresponds to the unpreconditioned PWDG operator, the middle spectrum corresponds to the PWDG operator preconditioned by a Schwarz operator based off of $L = 400$ partitions, and the bottom spectrum corresponds to $L = 8$ partitions. Here $\kappa = 11$ and $N^D_w = 4$ for all $D \in \mathcal{T}$. 
given by figure 2.3.7. Next I look at the eigenvalues for this problem to see if they exhibit the same clustering as the simpler homogeneous box case.

The eigenvalues for the piecewise constant wavenumber test case are given in figure 2.3.8. Despite the larger wavenumber and the piecewise constant wavenumber, it still yields good clustering characteristics, but a last test should include a problem with PML because the imaginary components create a dissipative character that has not been present in the previous two cases.

2.3.0.3 Box With PML Test Case

For the last example I consider again a box with a smaller internal box “cavity.” The cavity will be a square hole inside the domain such that Dirichlet boundary conditions are imposed with \( g = 0 \), which is the case of “sound hard” scattering. Finally an outer layer is made a PML. The mesh for this problem is illustrated in figure 2.3.9. I consider again the smaller wavenumber \( \kappa = 11 \) and compute the eigenvalues of the associated Schwarz problems in figure 2.3.10.

Again as the number of domains decrease, significant eigenvalue clustering is achieved. The three examples studied here cover many of the challenging cases for the Helmholtz equation, but a weakness is that they are computed assuming the matrices \( \mathbf{A} \) and the preconditioner \( \mathbf{F} \) are available in matrix form. So although these results are consistent with the GMRES performance observed in [12] when it was combined with Schwarz preconditioning, it does not completely answer the question of how approximations to \( \mathbf{F} \) will perform. The next chapter, chapter 3 will attempt to answer this question by discussing the GMRES algorithm and closely related variants which will allow \( \mathbf{F} \) to be applied in approximate form.
Figure 2.3.7: Piecewise constant wavenumber test case Schwarz levels. The mesh was generated using the GMSH software [59], and the partitions were generated by repeated calls to MGridGen [102]. The top mesh corresponds to no partitioning, the middle mesh corresponds to $L = 63$ partitions and the bottom mesh corresponds to $L = 16$ partitions.
Figure 2.3.8: Eigenvalues of the piecewise constant wavenumber box case with and without Schwarz preconditioning. The top spectrum is obtained from the case with no preconditioning, the middle case corresponds to $L = 400$ partitions, and the bottom case corresponds to $L = 8$ partitions. In this case $\kappa_1 = 31$, $\kappa_2 = 71$, and $N_w^D = 16$ for all $D \in T$. Figure 2.3.4 defines $\kappa_1$ and $\kappa_2$ and $T$ is illustrated in 2.3.7.
Figure 2.3.9: Schwarz levels for scattering box cavity with PML boundary conditions. The mesh was generated using the GMSH software [59], and the partitions were generated by repeated calls to MGridGen [102]. The top mesh corresponds to no Schwarz partitioning, the middle mesh corresponds to $L = 381$ partitions, and the bottom mesh corresponds to $L = 26$ partitions.
Figure 2.3.10: Eigenvalues of the scattering box cavity case with PML boundary condition. This is with and without Schwarz preconditioning. The top spectrum corresponds to an un preconditioned operator, the middle corresponds to an operator preconditioned using $L = 381$ partitions, and the last corresponds to $L = 26$ partitions. In this case $\kappa = 11$, and $N_{w}^{D} = 4$ for all $D \in \mathcal{T}$ where $\mathcal{T}$ is illustrated in 2.3.9.
Chapter 3

Matrix-Free Solution Process

The novel contributions of this chapter is the combination of recent research on domain decomposition based preconditioners with a flexible variant of GMRES in order to allow the preconditioner to be applied in a purely matrix-free fashion.

In this chapter I present the algorithms which when given a routine to evaluate operator evaluations, will attempt to make progress solving their associated systems with a supplied source term. While there are many choices of algorithms with the property of only requiring operator evaluation (and not a fully assembled matrix), such as for example Richardson iteration or BiCGSTAB [131], both used in early research on PWDG methods [32], [85]. I focus here on the GMRES algorithm [118]. A benefit of GMRES is the fact that it has no tunable parameters like Richardson iteration, and furthermore its residual norm minimizing property grants significant flexibility with preconditioners through a flexible variant [116] which is also explored here.

GMRES has been used in the context of PWDG methods before [12] and in this same work domain decomposition based preconditioners are used and demonstrate significant iteration count improvements when combined with restarted GMRES. This basic form of preconditioned GMRES is presented in section (3.1). Unlike in [12] however, a focus of this thesis is on matrix free methods. The domain decomposition
preconditioners will therefore be solved by additional calls to GMRES. This however raises some difficulties and breaks some of the theoretical benefits of GMRES, since its results depend on the preconditioner being represented by a linear operator. To recover some of these benefits the flexible GMRES (FGMRES) [116] algorithm is presented in section 3.1. FGMRES does not assume the preconditioner to have any additional structure such as linearity. Finally, additional structure of the underlying domain decomposition method can be exploited so that each independent subdomain can have its own separate FGMRES updates, rather than a single global update. A modification to FGMRES is presented in algorithm (6) which accounts for this additional structure. Ultimately the results suggest that although when the domain decomposition preconditioners are applied exactly (say by precomputing its LU factors) it reduces the iteration count significantly of the outer GMRES method, when the preconditioner is applied inexactly however by an inner call to GMRES or FGMRES it can be more challenging to predict its affect the overall cost, and can in fact lead to stagnation of FGMRES if the inner preconditioner solve is too inaccurate.

3.1 Restarted GMRES with Right Preconditioning

In this section I present the restarted GMRES algorithm used in the thesis. Restarted GMRES works by picking a restart parameter \( m \), usually based on memory limitations and the tradeoff of faster convergence rates (for larger \( m \)) versus fewer inner products (smaller \( m \)). Next it generates a basis \( b, Ab, A^2b, \ldots, A^{m-1}b \) which spans the Krylov space for the linear operator vector pair \((A, b)\) and then it seeks to find a linear combination \( x \) of this basis which minimizes the residual \((b - Ax)^*(b - Ax)\). This procedure can then be repeated indefinitely (or until a desired tolerance has been reached) by setting \( b \leftarrow b - Ax \) and then adding the result of re-applying restarted
GMRES back to $\mathbf{x}$. The number of restarted iterations it takes to achieve a desired residual tolerance can be large sometimes, and so a preconditioning operator $\mathbf{F}$ is often applied. Mathematically I use right-preconditioning, which means I solve the system $\mathbf{AFF}^{-1}\mathbf{x} = \mathbf{b}$, and this can be incorporated directly into the GMRES algorithm and is done in algorithm (4).

**Algorithm 4** Restarted GMRES with Right Preconditioning

```plaintext
procedure GMRES(A,b,F,reltol,restart,maxit)
    Assume: $\mathbf{A}, \mathbf{F}$ are invertible linear functions.
    Initialize $\mathbf{x} = 0$
    Initialize $\mathbf{H} \in \mathbb{C}^{\text{restart}+1,\text{restart}}$ such that $h_{j,i} = 0$.
    for $k = 1,\ldots,maxit$ do
        $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$
        if $\sqrt{\mathbf{r}^T\mathbf{r}}/\sqrt{\mathbf{b}^T\mathbf{b}} < \text{reltol}$ then return $\mathbf{x}$
        end if
        $\beta = \sqrt{\mathbf{r}^T\mathbf{r}}$
        $\mathbf{v}_1 = \beta^{-1}\mathbf{r}$
        for $i = 1,\ldots,\text{restart}$ do
            $\mathbf{z} = \mathbf{Fv}_i$
            $\mathbf{w} = \mathbf{Az}$
            for $j = 1,\ldots,i$ do
                $h_{j,i} = \mathbf{v}_j^T\mathbf{w}$
                $\mathbf{w} \leftarrow \mathbf{w} - h_{j,i}\mathbf{v}_j$
            end for
            $h_{i+1,i} = \sqrt{\mathbf{w}^T\mathbf{w}}$
            $\mathbf{v}_{i+1} = \mathbf{w}/h_{i+1,i}$
        end for
        $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{FV}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^*\beta\mathbf{e}_1$
    end for
end procedure
```

In algorithm 4 capital letters such as $\mathbf{V}$ refers to the matrix obtained by taking its columns as $\mathbf{V} = [\mathbf{v}_1,\ldots,\mathbf{v}_{\text{restart}}]$. One issue with the above algorithm is the assumption that the preconditioning operator $\mathbf{F}$ be linear, which among other things means it must be fixed every iteration. A modification that allows $\mathbf{F}$ to change every
iteration, possibly even being itself a further call to GMRES, was presented by Saad in [116], and is summarized in algorithm (5) below:

**Algorithm 5** Flexible GMRES

```
procedure FGMRES(A, b, F, reltol, restart, maxit)
    Assume: A is an invertible linear function.
    Initialize x = 0
    Initialize H ∈ C^{restart+1, restart} such that h_{j,i} = 0.
    for k = 1, . . . , maxit do
        r = b - Ax
        if √r^T r / √b^T b < reltol then return x
        end if
        β = √r^T r
        v_1 = β^{-1} r
        for i = 1, . . . , restart do
            z_i = Fv_i
            w = Az
            for j = 1, . . . , i do
                h_{j,i} = v_j^T w
                w ← w - h_{j,i} v_j
            end for
            h_{i+1,i} = √w^T w
            v_{i+1} = w / h_{i+1,i}
        end for
        x ← x + Z(H^T H)^{-1} H^* β e_1
    end for
end procedure
```

The main change in algorithm (5) over algorithm (4) is that the vectors z, which store the result of preconditioning by F, are stored rather than discarded. Therefore the memory cost of FGMRES (5) is roughly double that of GMRES (4), but the potential gain in iteration count savings could overcome this drawback with a faster solve time.

Before proceeding with discussion and results a further modification will be provided to account for additional structure provided in section 2.3. In that section an
approximation to the full PWDG formulation was provided such that the mesh $\mathcal{T}$ was partitioned, and on each partition a PWDG system formulated independently. Therefore a faster way to solve this partitioned system is to run GMRES concurrently for each partition, each partition using its own update formula.

**Algorithm 6** Partitioned Flexible GMRES

```plaintext
procedure PFGMRES(A,b,F,reltol,restart,maxit,nparts)

Assume: A is an invertible linear function.
Initialize $x = 0$
Initialize $H \in \mathbb{C}^{p,\text{restart}+1,\text{restart}}$ such that $h_{p,j,i} = 0$.

for $k = 1, \ldots, \text{maxit}$ | $p = 1, \ldots, \text{nparts}$ do

$r = b - Ax$
if $\sqrt{r^*r}/\sqrt{b^*b} < \text{reltol}$ then return $x$
end if

$\beta_p = \sqrt{r^*_p r_p}$
$v_{p,1} = \beta_p^{-1} r_p$

for $i = 1, \ldots, \text{restart}$ do

$z_i = F v_i$
$w = A z_i$

for $j = 1, \ldots, i$ do

$h_{p,j,i} = v_{p,j}^* w_p$
$w_p \leftarrow w_p - h_{p,j,i} v_{p,j}$
end for

$h_{p,i+1,i} = \sqrt{w_p^* w_p}$
$v_{p,i+1} = w_p / h_{i+1,i}$
end for

$x_p \leftarrow x_p + Z_p (H_p^* H_p)^{-1} H_p^* e_1$
end for
end procedure
```

With these algorithms in hand I can now investigate the utility of Schwarz based preconditioners when they are applied in a matrix free fashion, this is the topic of the next section.
3.2 Results

In this section I investigate the utility of domain decomposition based preconditioners, which were introduced in chapter 2 section 2.3. As mentioned in that section, the key equation to be solved is of the form $AFx = b$. I have already demonstrated that when $A$ is the PWDG operator and $F$ is the inverse of a Schwarz operator, the operator $AF$ has a favorable spectrum for use in GMRES (algorithm 4). However this kind of analysis only is helpful if $Fv$ can be accurately computed in an efficient manner. This can be challenging to accomplish if $F$ itself involves solving another system of linear equations. Therefore in what follows $F$ is approximated by using the partitioned FGMRES algorithm 6 where the input linear function is the linear function defined by the algorithm 3. It remains to show that this scheme is an improvement over the cost of simply defining $F$ to be the identity operator (that is, no preconditioning).

A common way to measure cost of an iterative linear solver is to count how many times one has to compute $Ax$ before the tolerance (the input reltol in algorithms 4, 5, and 6) has been reached and the solver returns a solution. This measurement is made in figures 3.2.3 and 3.2.4 where large reductions in the number of necessary outer FGMRES iterations are reported. This measurement does not account for the cost of computing approximations to $F$ however, therefore it remains unclear whether or not the same solution could have been obtained using no preconditioning with an overall less cost. In order to carefully account for the additional overhead of $F$, a more detailed presentation is needed of how it is computed.

The solver works with an outer FGMRES iteration (using algorithm (5)) with $A$ supplied as the full PWDG operator, the evaluation of which is done in matrix-free fashion using algorithm 1 by defining $Au = PWDG(T, \kappa, N_w, u)$ for all $u$. The preconditioning function $F$ combines partitioned flexible gmres (algorithm 6) with the
domain decomposition operator presented in section 2.3, again evaluated in a matrix free manner by its corresponding algorithm 3. To be more precise, for every \( x \) define \( G_x = \text{SchwarzPWDG}(\mathcal{T}, \kappa, N_w, x) \), then \( F \) will approximate \( G^{-1} \) by taking \( Fu = \text{PFGMRES}(G, u, I, \text{reltol}, \text{restart}, \text{maxit}, \text{nparts}) \) where \( I \) is the identity operator.

Then the task is to measure the cost of calculating \( \text{FGMRES}(A, b, F, \text{reltol}, \text{restart}, \text{maxit}) \) and then to determine if the overall cost of this computation is at all an improvement over simply computing the same thing with \( \text{GMRES}(A, b, I, \text{reltol}, \text{restart}, \text{maxit}) \) (note \( I \) being used in place of \( F \)).

The problem I will use to measure the efficiency of Schwarz based preconditioning is a box with an interior point source at the origin and PML boundary conditions. A diagram of this problem to be solved is given in figure 3.2.1 and a mesh for it is given in figure 3.2.2.

![Figure 3.2.1](image)

Figure 3.2.1: Box with PML boundary conditions containing an interior point source.

The first study will look at the number of iterations required to achieve a relative
Figure 3.2.2: The mesh $\mathcal{T}$ corresponding to the problem diagrammed in figure 3.2.1 to be used for GMRES convergence studies. The maximum edge length is $h \approx 1/2$.

residual of $1e-5$ as a function of the wavenumber $\kappa$, and here the parameter $N_w^D$ will be defined to be $N_w^D = (1/2)\kappa$ for all elements $D \in \mathcal{T}$. I make this choice because the edge lengths in the mesh in figure 3.2.2 are all approximately $1/2$, this therefore corresponds to a choice of $C = 1$ from the local basis criteria (2.2.33). Two Schwarz levels will be investigated, one with few partitions and one with partitions. The operator with many partitions can be expected to be easier to solve, but as shown in the previous section it does not provide the same quality of eigenvalue clustering as the Schwarz operator corresponding to fewer partitions. The inner P-FGMRES iterations will be fixed at 10 with a restart parameter of 1, and the outer FGMRES iterations will be fixed at 10,000 (which if it reaches this point, it will be declared that the solver did not converge). For comparison purposes, both preconditioned and unpreconditioned will be pictured below. To summarize using earlier notation, I am defining
\( \mathbf{Ax} = \text{PWDG}(\mathcal{T}, \kappa, N_w, \mathbf{x}) \)  
algorithm 1

\( \mathbf{Gx} = \text{SchwarzPWDG}(\mathcal{T}, \kappa, N_w, \mathbf{x}) \)  
algorithm 3  
(3.2.1)

\( \mathbf{Fx} = \text{PFGMRES}(\mathbf{G}, \mathbf{x}, \mathbf{I}, \text{reltol}, \text{restart}, \text{maxit}, \text{nparts}) \)  
algorithm 6

Figure 3.2.3: FGMRES iterations required to reach a fixed relative residual of \( 1e^{-5} \). These results were computed by taking \( \mathbf{b} \) to be the source term defined by the box point source problem represented by figure 3.2.1 and the mesh \( \mathcal{T} \) given by figure 3.2.2. Then the number of evaluations of \( \mathbf{Ax} \) required to solve FGMRES(\( \mathbf{A}, \mathbf{b}, \mathbf{F}, 1e^{-5}, 1, 10000 \)) where \( \mathbf{A}, \mathbf{F} \) are as in (3.2.1), and the parameters for \( \mathbf{F} \) are chosen to be \( \text{reltol} = 0, \text{maxit} = 10, \text{restart} = 1, \text{nparts} = L \) where \( L \) is the number of Schwarz partitions and \( L = 1 \) is defined to be the unpreconditioned PWDG case, i.e. \( \mathbf{F} = \mathbf{I} \).

The convergence study given in 3.2.3 demonstrates a difficulty arising in the Flexible GMRES case, namely the problem of breakdowns. In non-flexible GMRES a breakdown is the situation where one of the scaling factors becomes exactly zero in
the course of the algorithm, leading to a possible division by zero when normalizing the next iteration’s Krylov vector. It is now a well-known fact that in GMRES if such a situation occurs, then the approximate solution at that iteration is in fact the exact solution to the linear system being solved [117] (assuming the system is nonsingular) hence these breakdowns are often called “lucky.” The situation changes for flexible GMRES. Breakdowns can occur, but their relationship to the solution is more complicated [116]. In this case the Schwarz operator corresponding to $L = 8$ partitions should be the better preconditioner based on the numerical studies of chapter 2, at least when used exactly, but when used inexacty it actually performs worse than the preconditioner corresponding to $L = 91$ partitions, because the first iteration causes an “unlucky” breakdown. To investigate this issue further I perform another study where the number of inner iterations (fixed at $maxit = 10$ on this study) is allowed to vary.

The next study will investigate convergence of the Schwarz method as a function of inner P-FGMRES iterations, using the same schwarz levels as the previous study. For this the wavenumber $\kappa$ will be fixed at $\kappa = 51$ and the number of waves fixed at $N_{w}^{D} = 25$ for all $D \in \mathcal{T}$. This study will show how many outer FGMRES iterations it takes to converge when the number of inner Schwarz iterations increases, again keeping $restart = 1$ in both cases. For reference the un-preconditioned iterations will also be included but since there is no inner iterations for this, it is constant. The result of this is given in figure 3.2.4.

Although the results of figure 3.2.4 agree with the positive results of [12] in the limit of exact application of the preconditioner, they do not account for the cost of computing $Fx$. To estimate this cost I will assume that the cost of evaluating $Gx$ is roughly equal to the cost of evaluating $Ax$ (in fact they generally are close
Figure 3.2.4: These results were computed by taking $b$ to be the source term defined by the box point source problem represented by figure 3.2.1 and the mesh $\mathcal{T}$ given by figure 3.2.2. Then the number of evaluations of $Ax$ required to compute $\text{FGMRES}(A, b, F, 1e - 5, 1, 10000)$ where $A, F$ are as in (3.2.1), and the parameters for $F$ are defined to be $\text{reltol} = 0, \text{maxit} = i, \text{restart} = 1, \text{nparts} = L$ where $L$ is the number of Schwarz partitions, and $i$ ranges from 10 to 40 on the $x$-axis of the above figure.
together in total cost). With this in mind and returning to the definition of \( F \) we can define a “total cost” of solving by counting the total number of outer iteration \( n_A \) and then using the assumption that computing \( Gx \) is approximately as expensive as computing \( Ax \) we can multiply \( n_A \) by the number of inner preconditioner iterations, which is \( maxit \), and then to get a relative cost we can divide by the number of unpreconditioned GMRES iterations required to obtain the same solution. This is done for figure 3.2.6, which is a repeat of the result done in 3.2.4 except using this new relative cost measure rather than the number of outer iterations, the newly evaluated cost appears to be lower than the cost of solving without preconditioning suggesting the preconditioner could be an effective one, particularly if many inner iterations are used. Unfortunately the story changes when attempting to scale to larger problems.

Figure 3.2.5 : Refined mesh for GMRES convergence studies. The maximum edge length is \( h \approx \frac{1}{4} \)

For the next test case I consider what happens when attempting to solve a larger problem using the same parameters as in 3.2.4, except instead of using the mesh 3.2.2 I use the refined mesh 3.2.5. These results are given in figure 3.2.7 and in this case
Figure 3.2.6: FGMRES iterations required to reach a fixed relative residual of $1e-5$. These results were computed by taking $b$ to be the source term defined by the box point source problem represented by figure 3.2.1 with wavenumber $\kappa = 51$ and the mesh $\mathcal{T}$ given by figure 3.2.2. Then the relative cost is calculated by counting the number of evaluations of $Ax$ plus the number of evaluations of $Gx$ required to solve FGMRES$(A, b, F, 1e-5, 1, 10000)$ and dividing by the number of evaluations of $Ax$ to solve FGMRES$(A, b, I, 0, 1, \infty)$ where $A$, $F$, and $G$ are as in (3.2.1), and the parameters for $F$ are defined to be $reltol = 0, maxit = 10, restart = 1, nparts = L$ where $L$ is the number of Schwarz partitions and $L = 1$ is defined to be the unpreconditioned PWDG case, i.e. $F = I$. 

```plaintext

Convergence of GMRES

<table>
<thead>
<tr>
<th>Relative cost (smaller is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
</tr>
<tr>
<td>1.5</td>
</tr>
<tr>
<td>1.0</td>
</tr>
<tr>
<td>0.5</td>
</tr>
</tbody>
</table>

Inner Iterations

- $L = 126$
- $L = 8$
- $L = 1$

```
they do not indicate that the preconditioner has helped

In fact the results shown in figure 3.2.7 are more typical. One major point of difference between the results from 3.2.7 and those of 3.2.6 is that the value of $hk$ is smaller, roughly half the size. While I have chosen $C = 1$ for the basis criterion (2.2.33), it will be seen in section 5.3.2 that a better choice may be between $C = 2$ and $C = 2.5$, and $C = 1$ actually performs quite poorly in terms of solution accuracy. Taking $C = 2$ and repeating the study in 3.2.6 tells a more consistent story about the performance, the results of this are given in figure 3.2.8

3.3 Conclusion

The results of this chapter suggest that the Schwarz based preconditioners provide performance improvements only in the preasymptotic regime where the number of waves is chosen to be too small for accuracy, and once the number of waves is chosen according to accuracy considerations it simply becomes cheaper to use standard GMRES without Schwarz preconditioning. The studies could be repeated in many different ways by manipulating the GMRES parameters, for example the inner or outer restart parameter, however the results are essentially the same in all cases. To be used as a successful preconditioner the Schwarz domain decomposition may need to be combined with other techniques, but this is not further explored here. Some possible directions to take this to make it more successful can be found in chapter 6.

The low memory nature of the PWDG method combined with GMRES still yields many benefits without an optimal preconditioner, particularly in handling high frequencies which can not be otherwise handled by the finite element method. The next chapter will show how the PWDG operator evaluation can translate to GPU implementation, and after that the results chapter will demonstrate performance and
Figure 3.2.7: FGMRES iterations required to reach a fixed relative residual of $1e^{-5}$. These results were computed by taking $b$ to be the source term defined by the box point source problem represented by figure 3.2.1 with wavenumber $\kappa = 51$ and the mesh $\mathcal{T}$ given by figure 3.2.5. Then the relative cost is calculated by counting the number of evaluations of $Ax$ plus the number of evaluations of $Gx$ required to solve $\text{FGMRES}(A, b, F, 1e^{-5}, 1, 10000)$ and dividing by the number of evaluations of $Ax$ to solve $\text{FGMRES}(A, b, I, 0, 1, \infty)$ where $A$, $F$, and $G$ are as in (3.2.1), and the parameters for $F$ are defined to be $\text{reltol} = 0, \text{maxit} = 10, \text{restart} = 1, \text{nparts} = L$ where $L$ is the number of Schwarz partitions and $L = 1$ is defined to be the unpreconditioned PWDG case, i.e. $F = I$. 
Figure 3.2.8: FGMRES iterations required to reach a fixed relative residual of $1e^{-5}$. These results were computed by taking $b$ to be the source term defined by the box point source problem represented by figure 3.2.1 with wavenumber $\kappa = 51$ and the mesh $\mathcal{T}$ given by figure 3.2.2. This study is similar to that of figure 3.2.6, however instead of taking $C = 1$ from (2.2.33) it takes the more accurate $C = 2$. Then the relative cost is calculated by counting the number of evaluations of $Ax$ plus the number of evaluations of $Gx$ required to solve FGMRES($A, b, F, 1e^{-5}, 1, 10000$) and dividing by the number of evaluations of $Ax$ to solve FGMRES($A, b, I, 0, 1, \infty$) where $A, F,$ and $G$ are as in (3.2.1), and the parameters for $F$ are chosen to be $\text{reltol} = 0, \text{maxit} = 10, \text{restart} = 1, \text{nparts} = L$ where $L$ is the number of Schwarz partitions and $L = 1$ is defined to be the unpreconditioned PWDG case, i.e. $F = I$. 
accuracy properties of the PWDG method.
Chapter 4

GPU Implementation

Novel contributions of this chapter are the first implementation of a plane-wave discontinuous Galerkin method on a GPU, and an open source Julia interface into the accelerator threading model OCCA which was used to run all results for this thesis.

The purpose of this chapter is to show how the ideas presented thus far can be implemented efficiently on graphics processing units (GPUs) to achieve faster solution times. The threading model used is OCCA [95] through its Julia interface OCCA.jl, which I have written and open sourced on GitHub. OCCA abstracts common features of existing GPU programming models such as CUDA and OpenCL into a single interface which can then output parallel code into multiple backends including CUDA and OpenCL, but also OpenMP.

Similarly to CUDA and OpenCL, OCCA takes a host/device approach to managing the possibly different memory spaces of CPU memory and GPU memory. This means a “host” programming language, Julia in the case of this thesis, sets up and manages scheduling tasks to the “device” code which is compiled and executed upon being called by the host through the OCCA API. This lets high level algorithm details be managed in the host language of choice, while low level details of performance get managed in OCCA kernels. This combined with many of Julia’s novel features such as multiple dispatch, lets algorithms be expressed in a natural way with minimal
explicit management of lower level details once the OCCA kernels are written.

Although over the years and through different vendors the architecture specifics of GPUs change, they all have a common feature of having many independent “cores” which themselves dedicate significant space to arithmetic units, which from the point of view of CPU programming, may collectively be treated as a wide vector unit.

For this thesis all results were run on a Maxwell class Nvidia GTX 980 GPU. The Maxwell class of GPUs consist of a fixed number of Streaming Multiprocessors, which in the Maxwell case are called SMMs. An SMM consists of four warps, each warp consists of 32 single precision floating point units and 8 special function units (for computing e.g. exp, sin, or cos.). A simplified diagram of an SMM is pictured in figure 4.0.1 along with a simplified diagram of a warp in 4.0.2

![Figure 4.0.1: A simplified diagram of an Nvidia Streaming Multiprocessor (SMM) in a Maxwell class GPU. Each SMM contains four warps, which play a similar role as vector units in CPUs, though they are a little more general.](image-url)
Figure 4.0.2: A simplified diagram of a warp within an Nvidia Streaming Multiprocessor. The warp serves as a coarse level of parallelism. All warps may proceed along more or less independent code path, but within each warp the floating point units (called “threads” in Nvidia documentation) must generally all operate on the same path, and in most cases where two paths are possible (for example in an “if/else” block) then the two possible paths are executed sequentially.
The performance profile of this design lends itself to very good memory bandwidth when accesses to memory are structured, and a potentially very high floating point capability when there is a high degree of parallelism. Since DG methods tend to lend themselves both to structured memory access because of their compact nature, and to parallelism in floating point operations because each operation is a dense linear algebra operation such as a matrix-vector product, they have been widely and successfully used on GPUs [61], [92], [47], [60]. In the following sections this work is extended to the case of plane-wave DG using OCCA.

4.1 A Brief Introduction to OCCA

4.1.1 Hierarchy of Parallelism in OCCA

In OCCA the basic hierarchy of parallelism used for this thesis consists of an “outer” loop and an “inner” loop. The purpose of these are to easily map onto the structure of the GPU architecture. The “outer” loop is parallel over streaming cores, whereas the “inner” loop is parallel over the compute units of the streaming core, when the backend is chosen to be OpenMP the outer loop distributes over different CPU cores and the inner loop is unrolled and the backend compiler will attempt vectorization if possible. This parallelism hierarchy is illustrated by the skeleton .OKL file below, which has a C-like syntax:

While this code is very close to C syntax, there are some non-C decorations. “occaKernel” indicates an exported function which can be called from the host language via the OCCA API, “outer0” and “inner0” are decorations to indicate which level in the parallelism hierarchy that loop takes place (the last two decorations may become optional in future versions of OCCA). This code illustrates as well a very typical skele-
Listing 4.1: Minimal illustration of inner and outer loops in OCCA

```c
#define INNER_MAX 16
occaKernel void inner_outer_example(int outer_max){
  for(int outidx=0; outidx<outer_max; outidx++){  
    /*Outer-thread parallel code goes here.*/
    for(int inidx=0; inidx<INNER_MAX; inidx++){      
      /*Inner-thread parallel code goes here.*/
    }
    /*OCCA automatically places a barrier
     between inner loops. The barrier is
     local only to the threads from the
     inner loop.
     */
    for(int inidx=0; inidx<INNER_MAX; inidx++){    
      /*OCCA can handle many inner loops,
       but they must all use the same
       inner loop bound, which must be a
       compile-time known constant.
       */
    }
  }
}
```
ton of OCCA code, one important thing to note for example is that the outer loop bound may be passed in as a function parameter, whereas the inner loop bound must be known at compile time - hence why it is defined using a preprocessor statement. Since the latter parameter is likely to change, the OCCA API offers a programatic way of altering preprocessor definitions from the host language at the cost of kernel recompilation. Furthermore the “inner” loop can be broken and applied a second time, with an automatic barrier placed between them to ensure threads from the previous inner loop have all exited it before proceeding to the second inner loop. This is also a common pattern in OCCA programming, one inner loop may for example perform all computations and the second inner loop may wait on the first to finish and then place the output into host-accessible memory.

Memory in OCCA also works a little differently from standard C. Performance portability requires that OCCA explicitly have many memory spaces which modern CPU architectures and their compilers have otherwise made transparent to programmers today. This is because GPUs do not dedicate much of their device to prediction of code patterns the way CPUs do, the consequence of this is that users generally must manage “cache” memory manually (this is “shared” memory in OCCA). While many applications require very careful optimization of the memory spaces it uses and when it uses them, the PWDG method turned out to be very limited by arithmetic rather than memory performance. Therefore the next section will illustrate simple usage of “global” versus “shared” memory as it is relevant to implementation of the PWDG method.
4.1.2 Memory Spaces of OCCA

In OCCA the device memory space is separated into global, shared, and register memory. The following .OKL file provides an example of the use of all three of these memory spaces and their typical use cases.

Listing 4.2: A typical OCCA pattern using shared memory

```okl
#define INNER_MAX 16
occaKernel void inner_outer_example(int outer_max,
                                        int* offsets,
                                        float* input_data,
                                        float* output_data){
    for(int outer_index=0;outer_index<outer_max;outer_index++){
        /*The "shared" memory here may be thought of
         as a manually managed cache. The shared memory
         bound must be known at compile time like the
         inner loop bound, it is very common for the
         shared memory bound to be the same as the inner
         loop bound.*/
        shared float sin[INNER_MAX];
        shared float sout[INNER_MAX];

        /*Get indices to access a sparse column vector.*/
        /*These are considered to be "register" memory.*/
        int off = offsets[outer_index]
        int size= offsets[outer_index+1]-off;

        /*First inner loop to get data from slow "global"
        memory.*/
        for(int inner_index=0;inner_index<INNER_MAX;
            inner_index++;inner0){
            if(inner_index<size){
                sin[inner_index]=input_data[inner_index+off];
            }
        }
        /*Very common initialization of output.
```
Later code will likely involve many computations of form "sout[inner_index] += F(sin)". */
    sout[inner_index] = 0.0;
}
/* Recall that a barrier is placed here, so we can guarantee that the shared arrays "sin" and "sout" are initialized before proceeding to computation. */
for (int inner_index = 0; inner_index < INNER_MAX; inner_index++) {
    /* The function "some_computations()" below can accept shared memory pointers such as "sin" but if we tried to give it a pointer to global memory, it would be a compile error once the host language invoked OCCA to compile this kernel. */
    sout[inner_index] += some_computations(sin, size);
}
/* Another barrier guarantees above computation completes before populating output array. */
for (int inner_index = 0; inner_index < INNER_MAX; inner_index++) {
    if (inner_index < size) {
        output_data[inner_index + off] = sout[inner_index];
    }
}
/* When kernel exits it amounts to a complete kernel barrier, and at this point it is safe for the host to access the data written to the global memory array output_data. */
In standard C these three memory spaces still exist, but modern compilers and CPUs make their existence entirely transparent to the programmer for most tasks. OCCA forces the programmer to be more explicit about their use. Shared memory in OCCA can be thought of as a manually managed cache, and it has a much lower latency than global memory. Registers have the lowest latency of all, and most “standard” C declarations that do not receive a “shared” OCCA decoration are considered to be register memory, although the compiler has a little more freedom to alter the order of declaration or to avoid it altogether. Aside from the additional details of memory spaces and parallelism hierarchies, .OKL files may be treated almost entirely as if they were vanilla C code. Therefore the next step to setting up and running a kernel is to manage the host code, and a simple case is shown in the next section.

4.1.3 Scheduling Kernels using Julia and OCCA.jl

The host language used for this thesis was Julia [26] because it is dynamically typed like MATLAB or Python, has a strong focus on numerical computing like MATLAB (or NumPy for Python), but unlike MATLAB or Python allows explicit declaration of types. The ability to explicitly declare types as well as Julia’s built in direct C interface through its “ccall” function makes it very easy to interface a package like OCCA and to furthermore pass to it necessary information about type sizes without excess “glue” code, which is often necessary in MATLAB via the MEX interface or with Python.

These features of Julia allowed the quick construction of an OCCA interface for it, OCCA.jl. OCCA.jl is the chief means of offloading data to the GPU for further computation, in this thesis. I will provide a simple example of its usage and highlight where Julia features have been used. Below is provided a simple kernel for adding
vectors together, and an associated Julia script which runs the calculation, reads the data from device global memory, and then verifies that it is correct.

Listing 4.3: Simple OKL kernel for adding two vectors

```c
kernel void vectoradd(int entries,
                        float *a,
                        float *b,
                        float *ab){
    for(int group = 0; group < ((entries + (MAX_ITEMS -1)) / (MAX_ITEMS)); ++group; outer0){
        for(int item = 0; item < MAX_ITEMS; ++item; inner0){
            const int N = (item + ((MAX_ITEMS) * group));
            if(N < entries)
                ab[N] = a[N] + b[N];
        }
    }
}
```

Listing 4.4: Simple Julia script using OCCA.jl

```julia
using OCCA

#deviceID is important
#if there are multiple OpenCL or CUDA enabled devices
info = "mode=OpenCL, deviceID=0";
info = "mode=CUDA, deviceID=0";
info = "mode=OpenMP";
#Sets up OCCA device based on info string.
device = OCCA.Device(info);

#Julia will default to 64 bit ints,
#and the GPU kernel expects 32 bits.
#That is why the otherwise optional declaration of Int32 is here.
entries::Int32=1000;
#Set up host memory.
a=Array(Float32,(entries,));a[:]=1.0;
```
b=Array(Float32,(entries,));b[:,]=-1.0;
ab=Array(Float32,(entries,));
correctvals=zeros(Float32,(entries,));
#Allocates device memory. Automatically
#copies from host memory passed to it.
#Note that no type information is passed.
o_a  = OCCA.malloc(device, a);
o_b  = OCCA.malloc(device, b);
o_ab = OCCA.malloc(device, ab);
#Here a preprocessor definition is added dynamically,
#this allows more flexibility with the inner loops
#since their bound must be a compile-time constant.
max_items=16;
kerninfo = OCCA.KernelInfo();
OCCA.adddefine!(kerninfo,"MAX_ITEMS",string(max_items))
#Build OCCA kernel.
#First string is filename.
#Second string is name of occaKernel to build.
vectoradd = OCCA.buildkernel(device,kerninfo,"vectoradd.okl","vectoradd")
#Run kernel. The exclamation point in Julia is
#the conventional way to indicate that a function
#may mutate one of its arguments, in this case "o_ab."
OCCA.runkernel!(addvectors,entries,o_a, o_b, o_ab)

#Load data from device global memory into host memory.
#This barriers the host and device so that it is safe
#to access ab after the memcpy! call.
OCCA.memcpy!(ab, o_ab)

#Verify correctness
@assert( abs(sum(ab))<1e-5 )

These are the basic ingredients necessary to offload code to the GPU from the Julia programming lanuage. In the next section it is shown how this can apply to
computing the PWDG operator evaluation.

4.2 PWDG Operator Evaluation

The crucial bottleneck in many DG codes is the time it takes to perform an operator evaluation (algorithm 1), followed by low level BLAS routines such as vector additions and dot products which are usually necessary in an iterative linear solver. The overwhelming cost for the problems considered by this thesis however was simply in the operator evaluation, therefore although BLAS routines and BLAS-like routines are run on the GPU for the GMRES solves, their implementation is not novel and so not further discussed. A high level view of the parallel evaluation of the PWDG operator can be given by modifying the pseudocode of algorithm 1 to indicate where “outer” and “inner” loops occur, demonstrated in algorithm 7.

Algorithm 7 Parallel Evaluation of PWDG Operator

procedure PWDG($\mathcal{T}, \kappa, N_w, u$)

Note: Hatted variables are defined in (2.3.3),(2.3.2)

Assume: $g = 0$ in numerical fluxes (2.2.8),(2.2.7)

outerfor $D \in \mathcal{T}$ do

innerfor $j = 1, \ldots, N^D_w$ do

$\hat{v}^D_j = \mathbf{n}_D \cdot \mathbf{d}^D_j \int_{\partial D} \hat{u}(\mathbf{x}) i \kappa_D \exp(i \kappa_D \mathbf{d}^D_j \cdot \mathbf{x})$

$v^D_j \leftarrow v^D_j - \int_{\partial D} \left( \mathbf{n}_D \cdot \nabla u \right)(\mathbf{x}) \exp(i \kappa_D \mathbf{d}^D_j \cdot \mathbf{x})$

end innerfor

end outerfor

return $v$

end procedure
Note that algorithm 7 is essentially the same as algorithm 1, although at a high level it ignores many issues that could arise from managing many memory spaces. Therefore I will provide a simplified .OKL file to give an idea what the toplevel operator evaluation logic looks like in OCCA, it is incomplete because it involves further calls to functions to handle boundary conditions, but the most important and relevant logic relating to parallelism and memory management in OCCA is in the following code:

```c
/*Single precision complex wavenumber PWDG RHS evaluation.
 Define before calling:
 MAXWAVES - maximum number of waves.
 NEDGES - Number of edges per triangle (=3).
 NVERTS - Number of vertices per triangle (=3).
 BC_ROBIN - robin bc flag.
 BC_DIRICHLET - Dirichlet BC flag.
 BC_INTERIOR - interior bc flag.
 */
occaKernel void scpwdg_matvec(
    int nelements,
    scomplex* wavenumbers,
    int* wvoffsets,
    pwvertex* vcs,
    pwvertex* etov,
    int* etoe,
    int* bcflags,
    scomplex* in,
    scomplex* out) {
```
scomplex* out) {
    for (int group = 0; group < nelements; group++) {
        /* Set up shared memory for solution states */

        shared scomplex sharedinm[MAXWAVES];
        shared scomplex sharedinp[MAXWAVES];
        shared scomplex sharedout[MAXWAVES];
        /* Offset for this group’s plane waves */

        int nwavesm = woffs[group+1] - woffs[group];
        int wwoffsm = woffs[group];
        /* This groups mesh element info, and boundary conditions */

        int lbcflags[3] = {bcflags[group*NEDGES+0], bcflags[group*NEDGES+1], bcflags[group*NEDGES+2]};
pwvertex lvs[3] = {etov[group*NVERTS+0], etov[group*NVERTS+1], etov[group*NVERTS+2]};
        int ltoe[3] = {etoe[group*NEDGES+0], etoe[group*NEDGES+1], etoe[group*NEDGES+2]};
pwvertex vcm = vcs[group];
scomplex wvnm = wavenumbers[group];
        for (int item = 0; item < MAXWAVES; item++) {
            scomplex zero;
            zero.re = 0.0;
            zero.im = 0.0;
            sharedout[item] = zero;
            /* Fill input data */
            if (item < nwavesm)
                sharedinm[item] = in[wwoffsm+item];
        }
        barrier(localMemFence);
        /* Loop through local edges and compute relevant BCs */
        for (int led = 0; led < NEDGES; led++) {
            /* Grab the partition neighbor element belongs to */
            int neighbor = ltoe[led];
pwvertex vcp = vcs[neighbor];
scomplex wvnp = wavenumbers[neighbor];
            int wwoffsp = woffs[neighbor];
            int nwavesp = woffs[neighbor+1] - wwoffsp;
scomplex zero;
zero.re=0.0;
zero.im=0.0;

/*Get neighbor element data.*/
for(int item=0;item<MAXWAVES;item++){
    if(item<nwavesp){
        sharedinp[item]=in[woffsp+item];
    }
}
barrier(localMemFence);

for(int item=0;item<MAXWAVES;item++){
    if(lbcflags[led]==BC_INTERIOR){
        /*Evaluate DG fluxes.*/
        sc_pwdg_interior(lvs[led],lvs[(led+1)%NEDGES],vcm,
                         vcp,
                         wvnm,wvnp,
                         nwavesm,nwavesp,
                         item,sharedinm,sharedinp, sharedout);
    }
    else if(lbcflags[led]==BC_DIRICHLET){
        sc_pwdg_dirichlet(lvs[led],lvs[(led+1)%NEDGES],vcm,
                         wvnm,nwavesm,item,sharedinm,sharedout);
    }
    else if(lbcflags[led]==BC_ROBIN){
        sc_pwdg_robin(lvs[led],lvs[(led+1)%NEDGES],vcm,wvnm,
                      nwavesm,item,sharedinm,sharedout);
    }
}

/*Write to output state.*/
for(int item=0;item<MAXWAVES;item++){
    if(item<nwavesm){
        out[woffsm+item]=sharedout[item];
    }
}
Chapter 5

Results

Novel contributions of this chapter are high frequency solutions to the heterogeneous model Marmousi without any smoothing, results demonstrating an effective GPU implementation of the PWDG operator evaluation, and a validation of the source extraction method for a wide range of parameter choices.

5.1 Performance Metrics of GPU Implementation

In this section I demonstrate the performance of the operator evaluation kernel, a crucial bottleneck in the matrix-free solution process utilizing GMRES. On GPUs, workloads are often categorized as either compute-bound or memory-bound. These correspond respectively to the cases where many floating point operations are performed per memory load/store and where few operations are performed per memory load/store, because the optimizations one should perform are different depending on this categorization. Before proceeding to discuss the results of this section I will give a brief overview of the architectures used to produce them, a Maxwell series Nvidia GeForce GTX 980 for GPU results and an eight core Intel i7-5960X for the CPU results.

The Nvidia Maxwell series is divided into multiple Streaming Multiprocessors, for Maxwell these are called SMMs. The SMMs are further partitioned each into four
“warps,” which functions much like a vector unit but with more general capabilities. Each warp on an SMM controls 32 single precision floating point units and 8 special function units. Special function evaluations are an important component of the PWDG method, and since the Maxwell architecture does not match floating point units with special function units, I will measure the total floating point performance and special function performance of the PWDG kernel separately. A considerable portion of the Maxwell is dedicated to floating point performance, this is a common feature of all GPUs. On the other hand, CPUs are more tailored to general purpose computing and so their layout is quite different.

The Intel i7-5960X used for CPU results here is an eight core CPU that has AVX2 enabled on all cores. AVX2 allows computation on 512 bit wide vectors corresponding to 16 single precision floating point numbers, and therefore at any given time there are 128 single precision floating point unit equivalents available. For comparison to the Nvidia GeForce GTX 980: each warp (of which there are four for each SMM) controls 32 single precision floating point units. According to Nvidia documentation there are 16 SMMs in the GTX 980 making for a total of 2048 floating point units. Concerning special function units the Intel CPU does not have any, although it does have dedicated instructions for computing trigonometric functions, the inputs to these functions must be resident in a floating point register and therefore they take the place of possible floating point operations rather than being computed in addition to them. With these architectural differences in mind I now present the computational results.

Since this thesis focuses on the PWDG method as a matrix-free method, recomputing integrals in operator evaluation rather than saving them in a sparse matrix format, the resulting kernel is highly compute bound. Therefore it achieves a high measurement of floating point operations per second, shown in figure 5.1.1, but a
much lower memory transfer rate than typical DG kernels, since most of the time is spent computing integrals rather than in processing new data. The trade-off of this approach is that very low memory implementations allow large problems to be solved on a single workstation, but it has a possibly suboptimal runtime performance where some compute/memory middle-ground could obtain optimal runtime performance at the expense of solving smaller problems (or distributing them over many workstations).

Floating Point Performance of GPU PWDG Operator Evaluation

Figure 5.1.1: Floating point operations of PWDG operator evaluation kernel. Complex number multiplication is counted as four FLOPs, and addition as two. All values are the result of timing 400 repeated applications of operator evaluation, counting the total number of floating point operations, and dividing by total time. Multiple runs were used in order to rule out startup costs of offloading to the GPU. The results suggest that doubling the number of waves per element has the effect of doubling the floating point cost. Study was performed on a mesh of 16976 triangular elements.
An additional component to the PWDG method comes from the need to compute integrals involving oscillatory special functions such as sin and cos. These functions can be very expensive to use in practice and a significant body of work exists in reducing their cost specifically in the case of integrals, see [88] for a review. In the PWDG method I use analytic integration formulas and retain common factors wherever possible to reduce the number of special function evaluations. Nevertheless it is in the nature of the method that there be many such evaluations, I measure below how many special functions per second are necessary. I measure specifically sin, cos, and exp evaluations as special functions, the results of this measurement are given in figure 5.1.2. There exists little in the literature to benchmark this kind of result against, but to get an idea of the quality of this measurement we can note that the Maxwell architecture has 8 special function units for every 32 floating point unit, therefore if a code is intensive both on floating point cost and special function cost then we should expect the special function evaluations per second to be 8/32 = 1/4 that of the FLOP/s measurement, which appears to be the case here for all choices of $N_w$ except $N_w = 64$, possibly because at this level the GPU runs out of register memory to hold all of the operands. On the point of memory it is also usually important to measure memory bandwidth of a GPU kernel alongside its floating point performance, this will be the next point I discuss.

Memory bandwidth is the measure of how much data is transferred to and from global memory over the course of an entire run (or possibly many averaged runs). The PWDG method however does not do much data movement. This is in contrast to other DG methods where the basis functions themselves depend heavily on the underlying mesh, and therefore simply evaluating integrals of the basis functions requires the transfer of mesh data and data dependent on the mesh such as coordinate
Figure 5.1.2: Special function evaluations per second (all scaled by 1e9). This is computed similarly to figure 5.1.1, that is averaged over 400 runs. Each evaluation of the special functions sin, cos, exp are counted as a single special function evaluation. Many evaluations are required because of stabilized evaluations of the expression exp(z) – 1 for complex z. Study was performed on a mesh of 16976 triangular elements.
transformation factors, and possibly even quadrature rules for integration into shared or register memory before the calculation can take place. PWDG on the other hand only requires a minimal amount of mesh data and then the solution state itself. Therefore the total bandwidth, shown in figure 5.1.3, is quite low compared to other DG methods which might achieve over 200 GigaBytes per second [95]. Despite this, the method appears to run very well on the GPU compared to the CPU, which is the topic of the next computational study.

The last item of interest for performance of the PWDG method is to see how much faster the GPU code runs than the equivalent CPU code. Computing the speedup of a GPU code compared to a CPU code can sometimes yield misleading results, for example if the CPU code was very poorly written then a suboptimal GPU code may still exhibit significant speedup measurements and yet still have large room for improvement. Nevertheless I measure speedup by using the same OCCA code as in the GPU case, but targeting it to its OpenMP backend. To try and get a more meaningful measure I inlined all functions and passed aggressive optimization flags to the compiler, even those which can hurt accuracy such as “-fast-math.” Figure 5.1.4 demonstrates the results of this study. It suggests at least for the OCCA code available that the GPU is the significantly better option for a choice of backend.

The results in this chapter show reasonable scaling on the GPU architecture as the number of plane-waves per element is increased, and although the bandwidth is low compared to other DG type methods the speedup over its CPU equivalent was still substantial. It remains to verify some of the important convergence properties of this method, and this is the topic of sections 5.2 and 5.3.
Figure 5.1.3: Measurement of memory bandwidth of the PWDG operator evaluation on the GPU. Bandwidth is computed by measuring the total number of bytes that the PWDG kernel loads from GPU global memory. In some cases a byte contribution is scaled by the length of the OCCA inner-for loop in cases where accesses are uncoalesced. This count is then divided by the total time of execution. The bandwidth value here is low compared to other DG methods because very little data is necessary beyond mesh information and the solution vector, suggesting that the PWDG method has floating point arithmetic as a bottleneck, rather than a memory bandwidth bottleneck. All values are obtained by averaging over 400 runs, and on a mesh of 16976 triangular elements.
Figure 5.1.4: Measurement of the speedup of evaluating the PWDG operator on GPU by targeting CUDA and then OpenMP using the OCCA API. The OpenMP code was run first using 8 threads and then 16 threads to observe if hyperthreading improved the time, and then the fastest time was used to compute speedup. The compiler flags used for OpenMP code were the most aggressive vectorization and function inlining options available in gcc as well as the standard -O3 option. These values were obtained by averaging over 400 runs on a mesh of 16976 triangular elements.
5.2 Dependence of Error on Source Extraction Patch Size

In chapter 2 I gave the source extraction formulation for the first time for PWDG methods. It allows the transformation of certain Helmholtz equations with a nonzero source term (a point source in this case) into an equivalent system that is homogeneous, meaning its source term is zero. This makes the problem much more amenable to plane wave approximation, but it also creates additional questions.

One point of interest not yet explored in the literature is the impact on the size of the source extraction patch on the overall error, as compared to other parameters such as wavenumber and mesh size. To investigate this I consider three meshes of the box $[-5, 5] \times [-5, 5]$, these meshes are pictured in figure ???. Next I choose a radius from the point source such that all elements falling within that distance from the point source will reside in the source extraction patch, I then solve the PWDG system by running GMRES until it stagnates and compute the relative error. All relative errors are computed with the formula

$$\frac{||u - u_{exact}||}{||u_{exact}||} = \frac{\sqrt{\sum_{D \in T} \int_D |u - u_{exact}|^2}}{\sqrt{\sum_{D \in T} \int_D |u_{exact}|^2}}$$

I will separate the point source cases into the different wavenumbers studied, $\kappa = 11, 31, 51$. For all wavenumbers two radii will be investigated, $r = 1/2$ and $r = 3/2$. It will be seen however that the parameter $r$ has little impact on convergence, and that convergence is mostly guided by other parameters such as $N_w$ and the mesh size.
5.2.1 Point Source Case $\kappa = 11$

For the case of $\kappa = 11$ we see that all values of $N_w$ lead to convergence except for $N_w = 4$ which appears to be preasymptotic. The values of $r$ in this case do not appear to make any meaningful difference to this result, therefore for this frequency it appears that the size of the source patch is not an important consideration for approximating a point source. The results for the case of $r = 1/2$ may be found in figure 5.2.1, and the results for the case of $r = 3/2$ may be found in figure 5.2.2.

Figure 5.2.1: Relative error using formula (5.2.1) for $r = 1/2$ and $\kappa = 11$. In this study all runs were reasonably successful except for the case of $N_w = 4$ which is in the preasymptotic regime.

5.2.2 Point Source Case $\kappa = 31$

For the case of $\kappa = 31$ we see that $N_w = 4$ is preasymptotic again, as in the case $\kappa = 11$, and also unlike the case $\kappa = 11$ the number of waves value $N_w = 8$ starts
Figure 5.2.2: Relative error for $r = 3/2$ and $\kappa = 11$. Despite the bigger element patch for the source function, the results are similar to the case $r = 1/2$ and the same wavenumber.

off preasymptotic but starts to converge as the meshes refine. Surprisingly the case $r = 3/2$ was actually slightly less accurate than the case of $r = 1/2$ as can be seen by comparing the figures for $r = 3/2$ in 5.2.4 versus the case $r = 1/2$ in figure 5.2.3.

I will next consider a much larger wavenumber $\kappa = 51$ and observe the point source behavior as a function of $r$ and $N_w$.

5.2.3 Point Source Case $\kappa = 51$

The case $\kappa = 51$ is the largest wavenumber considered for the point source problem. Since it is such a large wavenumber, the first two values of $N_w = 4, 8$ are preasymptotic for all three meshes and we only see convergence for $N_w = 16, 32$. This is the behavior for both the case $r = 1/2$ and $r = 3/2$, as can be seen in the figures 5.2.5 and 5.2.6.
Figure 5.2.3: Relative error (using formula (5.2.1)) for $r = 1/2$ and $\kappa = 31$. Only the last two cases observe convergence for mesh refinement, as the higher wavenumber increases the preasymptotic behavior.

Figure 5.2.4: Relative error for $r = 3/2$ and $\kappa = 31$. Despite the larger point source patch, the convergence behavior is no better (and actually, slightly worse) than the corresponding case with a smaller patch.
Except for the one case from $\kappa = 31$ where a bigger point source patch actually *hurt* accuracy.

It appears that little in the convergence profile of PWDG with source extraction when the source extraction patch is made larger. One possible reason for this is that the wavenumbers considered are moderate to large, and in fact the PWDG method is not able to handle lower wavenumbers because the poor conditioning of the basis would yield an operator that is numerically singular. It could be that in lower wavenumbers the singularity in the Green’s function becomes more challenging to resolve and is not sufficiently contained in a small source patch. This however is not really possible to study in the context of plane wave methods unless a very effective preconditioner is developed to handle the conditioning problem.

Figure 5.2.5: Relative error (computed using formula (5.2.1) for $r = 1/2$ and $\kappa = 51$). As in the previous study only the last two cases converge, but the case of $N_w = 32$ converges faster than the case of $N_w = 16$. 
Figure 5.2.6: Relative error for $r = 3/2$ and $\kappa = 51$. The results for this run are identical to the corresponding study for $r = 1/2$.

The conclusion to draw here therefore is that the patch size does not appear to have a significant impact on the resulting accuracy, at least in the presence of other approximations such as the PML. Once the number of waves is chosen appropriately compared to the wavenumber $\kappa$ and mesh size $h$, convergence is achieved independently of the point source patch size $r$. Therefore considerations of accuracy to the Helmholtz equation appear to be more important than the size of the source patch. More details for how to choose the parameter $N_w^D$ can be found in section 2.2.4.

### 5.3 Error Behavior for Known Analytic Solutions

In this section I investigate how the error behaves as a function of three key parameters to DG discretizations of the Helmholtz problem: mesh spacing $h$, the number of waves per element $N_w$, and the wavenumber $\kappa$. 
The first test case is the Mie cylinder solution, the mesh sequence for convergence is given by figure 5.3.1

and the analytic solution used to compute error and impose Dirichlet boundary conditions is

\[
 u(x, y) = \exp(i\kappa x) + \sum_{m=-\infty}^{\infty} i^m H_0^m(\kappa r) - i H_0^m(\kappa r) \exp(i m \theta) \tag{5.3.1}
\]

where \( \theta = \arctan(y/x) \) and \( r = \sqrt{x^2 + y^2} \).

The next case shows how the method performs in the presence of more than one wavenumber. The mesh sequence is pictured in figure 5.3.2.

and following the notation from [89], The analytic solution for this test case is given by

\[
 u(x, y) = \begin{cases} 
 T \exp(i(K_1 x + K_2 y)) & : y > 0 \\
 \exp(i \kappa \eta_1 x d_x + y d_y) + R \exp(i \kappa \eta_1 (d_x x - d_y y)) & : y > 0 
\end{cases} \tag{5.3.2}
\]

where \( d_x = \cos(\theta), d_y = \sin(\theta) \) is the direction of the incident plane wave, and \( K_1 = \eta_2, K_2 = \kappa \sqrt{\eta_2^2 - \eta_1^2 d_x^2}, R = -(K_2 - \kappa \eta_1 d_y) / (K_2 + \kappa \eta_1 d_y) \). The final two parameters \( \eta_1, \eta_2 \) are what scale the wavenumber in the Helmholtz equation so that in the top half of the box we get \( \kappa \leftrightarrow \eta_1 \kappa \) and similarly for the bottom half of the box.

The two problems introduced in this section present two separate challenges to the PWDG method. The MIE scattering problem introduces a lot of local solution variation near the scattering object, and these can be challenging for the plane-wave basis to capture. Nevertheless convergence is observed in many cases for this problem.

The other challenge in the piecewise constant wavenumber case is to obtain the correct values near the material interfaces, and this depends heavily on the correct
Figure 5.3.1: Sequence of meshes for mie cylinder scattering test case.
Figure 5.3.2: Sequence of meshes for piecewise constant wavenumber test case.
interface condition, which PWDG should impose automatically. We will see however that reasonable convergence is achieved for both of these problems.

5.3.1 Convergence Through Mesh Refinement

This section investigates convergence through mesh refinement of the MIE cylinder scattering problem and the piecewise constant wavenumber box problem. Although in the context of traditional convergence studies these results may seem negative, it is more a reflection of the nature of the plane-wave basis. Unlike more traditional methods based on polynomials where the condition number of the system grows relatively smoothly as a function of mesh size and polynomial order, plane-wave based methods instead have a small window of usable values of the number of waves per element, which I will call $[N_{w}^{\min}, N_{w}^{\max}]$. If one goes below this window by choosing $N_{w} < N_{w}^{\min}$ then convergence is not observed (this is referred to as being “preasymptotic”), and if one goes above this window by taking $N_{w} > N_{w}^{\max}$ then conditioning of the system makes it numerically impossible to achieve even a single digit of accuracy. Therefore I present these results but they will primarily motivate the need for a good selection criteria for the number of plane-waves per element such as the one derived in section 2.2.4.

The first problem I investigate is the MIE cylinder scattering problem. The analytic solution is given by (5.3.1) and the mesh sequence used for convergence studies is illustrated in figure 5.3.1. Since there are many parameters at play here, the wavenumber, the number of plane-waves per element, and the mesh sequence, I break these results into multiple tests. Each test corresponds to a single wavenumber with the expectation that larger wavenumber leads to a more challenging test case. The first test is a moderate wavenumber of $\kappa = 11$
5.3.1.1 The MIE Cylinder Case for $\kappa = 11$

Here I investigate the case $\kappa = 11$ for the convergence problem given by the mesh sequence 5.3.1 and whose purely Dirichlet boundary conditions are given by the analytic solution 5.3.1. After running the study for the number of waves $N_w = 4, 8, 16, 32$ and computing relative error according to the formula 5.2.1 we observe that the case of $N_w = 8$ achieves the best convergence, but only on the first two meshes in the sequence. All other cases do not perform very well. The reason for this is that the wavenumber $\kappa$ is relatively small compared to the mesh, for example we have $11h = 0.125 \cdot 11$ for the final mesh in the sequence. By the linear dependence argument found in chapter 2 and summarized by the condition 2.2.33 this yields a very small window of choices for $N_w$ on this last mesh. The smaller choices of plane-waves per element however are not big enough for convergence, despite possibly satisfying 2.2.33 for a reasonable choice of $C$. Therefore what we see is only a single value of $N_w = 8$ leading to convergence (except on the last mesh, which again had too small edges), as anything above or below it will destroy accuracy either for being preasymptotic, or for ill-conditioning. The results of this study can be found in figure 5.3.3. Therefore the next step is to consider higher wavenumbers and see if managing conditioning becomes easier to do.

5.3.1.2 The MIE Cylinder Case for $\kappa = 21$

Next I increase the wavenumber to $\kappa = 21$. In this case we see that unlike that case of $\kappa = 11$ the number of waves per element of $N_w = 8$ decreases its relative error for every single mesh, which is expected due to the increased wavenumber. One unexpected result however is that $N_w = 32$ achieves two digits of accuracy on the final mesh, and this appears to be in contradiction to the canonical behavior of plane-waves because
Relative Error for MIE Problem

Figure 5.3.3: Relative error for $\kappa = 11$ computed using the formula (5.2.1) taking $u_{exact}$ to be $u$ in the analytic formula (5.3.1). The mesh sequence for this study is given in figure 5.3.1. The error actually grows in most cases because of the small wavenumber compared to the mesh spacing, therefore the poor conditioning of these problems quickly overcomes any possible convergence.
the mesh at that level is very small in comparison to the wavenumber. This behavior however is not robust, it does not happen on any other test cases considered in this thesis. All other values of $N_w$ yield expected behavior, $N_w = 4$ is preasymptotic while $N_w = 8$ appears to be converging, and $N_w = 16$ fails due to conditioning. The results for this test may be found in figure 5.3.4. If the trend of the past two studies continues, then increasing the wavenumber further should make more values of $N_w$ preasymptotic while yielding convergence for larger $N_w$, this will be seen in the next section.

![Relative Error for MIE Problem](image)

Figure 5.3.4 : Relative error for $\kappa = 21$ computed using the formula (5.2.1) taking $u_{\text{exact}}$ to be $u$ in the analytic formula (5.3.1). The mesh sequence for this study is given in figure 5.3.1. Only the case $N_w = 32$ appears to exhibit convergence, possibly because the other cases are still preasymptotic.
5.3.1.3 The MIE Cylinder Case for $\kappa = 41$

In the final MIE cylinder test case I take $\kappa = 41$, which is the largest wavenumber considered for this test case. The convergence behavior in this case follows the expected trend. Since the wavenumber is large compared to the first two meshes, the number of waves $N_w = 4, 8$ are preasymptotic and do not yield meaningful accuracy. The case $N_w = 16$ however appears to fall within the window for convergence in this case except on the last mesh. Here the case $N_w = 32$ was too big and therefore bad conditioning prevented it from increasing accuracy as the mesh refined. The results for this test case can be found in figure 5.3.5.

![Relative Error for MIE Problem](image)

Figure 5.3.5: Relative error for $\kappa = 41$ computed using formula (5.2.1) taking $u_{exact}$ to be $u$ in the analytic formula (5.3.1). This uses the mesh sequence from figure 5.3.1.

The MIE cylinder results show that a balancing act must be played between the preasymptotic regime on the one hand, where not enough approximation is used to start seeing an accurate result, and conditioning on the other hand where too many
plane waves are used and therefore violating the condition (2.2.33). We will see essentially the same thing as well for the case of piecewise constant wavenumbers.

For the case of piecewise constant wavenumbers all studies use $\eta_1 = 1$ and $\eta_2 = 4$ from the analytic solution (5.3.2). Two incident wave directions are considered, $\theta_{inc} = \pi/4$ and $\theta_{inc} = \pi/3$. Again these will be broken down into separate cases according to the wavenumber, as this is the parameter that causes the most change in results.

### 5.3.1.4 Heterogeneous Box Case for $\kappa = 11$

For the case of $\kappa = 11$ we actually see error reduction for all values of $N_w$ on all meshes in the sequence. This does not contradict earlier comments, this is because on the bottom of the domain the actual wavenumber used is $\kappa \eta_2 = 44$ and therefore larger numbers of plane-waves can be used on the bottom of the domain, but still conditioning will impact those choices of the parameter $N_w$ on the top and therefore rather than seeing two digits of accuracy as in the MIE Cylinder case, we only see a single digit. In this case the choice of incident angle did not appear to make any meaningful difference in the convergence profile. The results can be seen for $\theta/4$ in figure 5.3.6 and for the case $\theta/3$ in figure 5.3.7. Since the value of the scaling factor $\eta_2$ is large, increasing the wavenumber should start to show significant effects with respect to what values of $N_w$ yield convergent behavior. I investigate next the larger value of $\kappa_2 1$.

### 5.3.1.5 Heterogeneous Box Case for $\kappa = 21$

As predicted in the previous section, increasing the wavenumber $\kappa$ had a very dramatic effect on the convergence behavior as a function of $N_w$. Since the wavenumber in the bottom half of the domain is $\kappa \eta_2 = 84$. All choices of $N_w$ except for $N_w = 16, 32$
Convergence for piecewise constant wavenumber test case

Figure 5.3.6: Relative error for $\kappa = 11$ and $\theta_{inc} = \pi/4$. The relative error is computed using formula (5.2.1) taking $u_{exact}$ to be $u$ in the analytic solution (5.3.2). The parameters are chosen as $\eta_1 = 1$, $\eta_2 = 4$.

are clearly preasymptotic in this case. One point of interest however is that the convergence behavior for the two incident angles are different now, possibly because the jump between the wavenumbers in the box are much larger, in the $\kappa = 11$ case the wavenumber difference was $44 - 11 = 33$, but in this case it is $84 - 21 = 63$. For the case $\theta = \pi/4$ illustrated in figure 5.3.8 we see convergence to one digit of accuracy for $N_w = 16$ but no convergence for $N_w = 32$, whereas for the case $\theta = \pi/3$ illustrated in figure 5.3.9 we get convergence for both $N_w = 16$ and $N_w = 32$. To investigate if this behavior continues by increasing wavenumber more, I increase the next study to $\kappa = 41$ to see if similar convergence behavior occurs.
Convergence for piecewise constant wavenumber test case

Figure 5.3.7: Relative error for $\kappa = 11$ and $\theta_{inc} = \pi/3$. The relative error is computed using formula (5.2.1) taking $u_{exact}$ to be $u$ in the analytic solution (5.3.2), the parameters are chosen as $\eta_1 = 1, \eta_2 = 4$

5.3.1.6 Heterogeneous Box Case for $\kappa = 41$

The case of $\kappa = 41$ is the largest for the heterogeneous box case. It is actually very large because of the parameter $\eta_2 = 4$ which scales the wavenumber in the bottom half of the box, making it effectively $\kappa \eta_2 = 164$. In this regime even small changes in data can have a large impact on the output, and all choices of $N_w$ are preasymptotic except for $N_w = 32$ for the case of $\theta = \pi/4$ (shown in figure 5.3.10). In the case of $\theta = \pi/3$ all values of $N_w$ are preasymptotic except for $N_w = 32$ and on the very last mesh the case $N_w = 16$ appears to also converge, this is shown in figure 5.3.11. In this case the wavenumber in the bottom half of the box is so large that this small change in input could account for the different convergence behavior, where in one case it leads to a solution which is more easily approximated by the $N_w = 16$ equally
Convergence for piecewise constant wavenumber test case

Figure 5.3.8: Relative error for $\kappa = 21$ and $\theta_{inc} = \pi/4$. The relative error is computed using formula (5.2.1) taking $u_{exact}$ to be $u$ in the analytic solution (5.3.2) the parameters are chosen as $\eta_1 = 1$, $\eta_2 = 4$.

The convergence studies in this section illustrate the need for careful choice of the parameter $N_w$. In the next section I will examine proper choices of this parameter based on the criterion derived in this thesis and summarized in equation (2.2.33).

5.3.2 Impact of Local Basis

In the previous section a more traditional convergence analysis was undertaken, and in all cases a range of number of plane-waves per element were studied. For a majority of the results it was found that one choice of $N_w$ would yield the best convergence, where all other choices gave suboptimal choices. The problem arises from the fact that if one chooses too few plane waves per element, then the approximation quality
Convergence for piecewise constant wavenumber test case

Figure 5.3.9: Relative error for \( \kappa = 21 \) and \( \theta_{inc} = \pi/3 \). The relative error is computed using formula (5.2.1) taking \( u_{exact} \) to be \( u \) in the analytic solution (5.3.2), the parameters are chosen as \( \eta_1 = 1, \eta_2 = 4 \)

Figure 5.3.10: Relative error for \( \kappa = 41 \) and \( \theta_{inc} = \pi/4 \). The relative error is computed using formula (5.2.1) taking \( u_{exact} \) to be \( u \) in the analytic solution (5.3.2), the parameters are chosen as \( \eta_1 = 1, \eta_2 = 4 \)
Convergence for piecewise constant wavenumber test case

![Graph showing relative error vs. wavenumber](image)

Figure 5.3.11: Relative error for $\kappa = 4$ and $\theta_{inc} = \pi/3$. The relative error is computed using formula (5.2.1) taking $u_{exact}$ to be $u$ in the analytic solution (5.3.2), the parameters are chosen as $\eta_1 = 1, \eta_2 = 4$

is too poor to yield a good solution, but if one tries to remedy this by picking many plane-waves they run the risk of increasing the condition number so dramatically to the point that there is no recoverable accuracy.

In chapter 2 I gave a mathematical derivation of a possible condition that can automatically choose a suitable value of $N_w^D$ for all mesh elements $D \in \mathcal{T}$, but it has a tunable parameter $C$. In this section I investigate the impact of this parameter $C$ on the MIE cylinder scattering problem with local mesh refinements, the mesh of which is pictured in figure 5.3.12.

what we should expect to see is that by increasing $C$ too much (allowing too many plane waves), accuracy deteriorates as the conditioning overcomes the available precision, and that by decreasing $C$ too much to improve conditioning will put the solution in the preasymptotic regime of convergence. The ideal situation is to find
Figure 5.3.12: MIE cylinder scatter problem with local refinements. The difference in mesh element sizes requires that full PWDG penalties are used and that a $C$ from the local basis criterion (2.2.33) is carefully selected to balance between both convergence and ill-conditioning.
a suitable value of $C$ that gives a certain level of accuracy. This is found in figure 5.3.13 where taking $C = 2.5$ appears to give the best result in terms of accuracy.

![Relative Error for MIE Problem](image)

Figure 5.3.13 : Relative errors for choice of local basis functions using the criteria (2.2.33) for different values of $C$. Analytic solution for error computation is given by equation (5.3.1). Small values of $C$ put the resulting solution in the preasymptotic, where large values result in a problem with no achievable accuracy due to conditioning.

Having now identified a possible value of $C$ for choosing the local basis I am now in a position to solve some common benchmark problems. This is the topic of the next section.

### 5.4 Solutions to Select Benchmark Problems

In this section I solve three benchmark problems for which there is no analytic solution. The first problem is the Marmousi problem, which is a heterogeneous velocity model. In terms of PWDG the Marmousi model is by far the most challenging, and
to my knowledge this is the first time the PWDG method has applied to the exact Marmousi model. The next problem is a scattering “U” shape, which creates internal reflections that can be challenging for the PWDG method to handle. The last problem is a scattering square, which is a classical test case for the Helmholtz equation.

5.4.1 Marmousi Model

The next problem considered is the Marmousi problem, which is a frequently considered benchmark for wave propagation solvers and for seismic applications (see e.g. [93], [139], [132]). The Marmousi model is pictured in figure 5.4.1. This test case is challenging because of the sharp contrasts in material properties, which translates to sharp contrasts in wavenumber. This fact makes it actually especially challenging for the PWDG method because it can only handle piecewise constant wavenumber, and therefore in order to be sure it is solving the correct model I use mesh adaptivity to ensure that the piecewise constant approximation to Marmousi is close to the original model.

Figures 5.4.2 and 5.4.3 show some approximation artifacts of using a regular mesh derived from the underlying regular grid that defines Marmousi. Two levels of automatic mesh refinement are done, with the error indicator being the mismatch between bilinear interpolation on the regular grid and piecewise constant approximation on a triangular mesh. The result of this is a large mesh (over 500000 elements) and so can not be visualized in its entirety, but a zoomed in portion can be seen in figure 5.4.4 along with the mesh that resulted from refinement in 5.4.5. It now remains to solve the model.

I present three visualizations of the field resulting from a high frequency solve involving the Marmousi model. Because of the locally refined mesh the penalties
Figure 5.4.1: Image of the Marmousi model, the color represents wavenumber for the case of 100Hz frequency. This image was constructed from the regular grid definition of Marmousi, but there is an extra layer around the model itself which is to serve as a PML in later computations but otherwise does not have a relationship to the model.

Figure 5.4.2: Marmousi model projected to piecewise constant velocity on regular mesh. There are some smearing artifacts near regions with sharp interfaces.
Figure 5.4.3: Marmousi model projected to piecewise constant velocity, showing the regular mesh. Note that although Marmousi itself is defined on a regular grid, producing a regular mesh from that grid and projecting the wavenumber to the space of piecewise constant functions introduces smearing artifacts.

Figure 5.4.4: Marmousi model projected to piecewise constant velocity on mesh with local refinement. The model appears much closer to the original Marmousi model.
Figure 5.4.5: Marmousi model projected to piecewise constant velocity. This shows how automatic mesh adaptivity has picked out the interfaces in order to capture the sharp changes in velocity.

are defined to be fully PWDG and so can be variable over the mesh, and because of the heterogeneous nature of this problem the number of waves per element also is allowed to vary and uses the formula (2.2.33) with $C = 2$. The wavenumber $\kappa_D$ is computed by the formula $\kappa_D = f/(2\pi v_D)$ where $f$ is the frequency in Hertz and $v_D$ is the local element velocity. A PML is also attached to the Marmousi model to minimize numerical reflections.

Figure 5.4.6 shows a zoomed in solution solved at the frequency $f = 120Hz$, and figures 5.4.7 and 5.4.8 show a field plot for the case of frequency $f = 150Hz$.

5.4.2 Scattering “U” Shape

Another problem of interest arises from the localized behavior that comes from a scattering object with a “U” shape. The internal reflections pose a challenge to the PWDG method, but it can be solved. A diagram of this problem is given by figure
Figure 5.4.6: Point source on Marmousi with frequency $f = 120\ Hz$. The point source location is $x = 1500\ m$ and $y = 53\ m$ and the source patch is defined to be all elements within $r = 50\ m$ of the point source location.

Figure 5.4.7: Point source on Marmousi with frequency $f = 150\ Hz$. The point source location is $x = 1500\ m$ and $y = 53\ m$ and the source patch is defined to be all elements within $r = 50\ m$ of the point source location. This is zoomed out to see global features of solution. This problem involved more than 20,000,000 degrees of freedom.
Figure 5.4.8: Point source on Marmousi with frequency $f = 150Hz$. The point source location is $x = 1500m$ and $y = 53m$ and the source patch is defined to be all elements within $r = 50m$ of the point source location. This image is zoomed into the point source to see finer scale features of solution. This problem involved more than 20,000,000 degrees of freedom.

5.4.9 as well as its associated mesh in figure 5.4.10.

The field plots in figures 5.4.11, 5.4.12, and 5.4.13, correspond to a homogeneous test cases of $\kappa_D = 25, 45, 65$ respectively. For all these runs the parameters were chosen as $N_{w}^{D} \approx h|\kappa_D$ with UWVF penalties: $s = 0$, and $C = 2$ in the local basis selection criteria 2.2.33.

5.4.3 Scattering Square

The last test case I consider is that of a scattering square, the mesh of which is pictured in 5.4.14.

The field plots in figures 5.4.15, 5.4.16, and 5.4.17 correspond to a homogeneous test cases of $\kappa_D = 25, 50, 75$ respectively, and as in the U-scattering problem the
Figure 5.4.9: Diagram of U shape model with point source. The cut out portion of the internal reflecting object causes incident waves to produce internal reflections with a lot of local variation, these are believed to be difficult to solve with PWDG method except on refined meshes.
Figure 5.4.10 : Mesh of U shape model.

Figure 5.4.11 : A U-shape scattering an incident wave originating from a point source. The point source location is $x = 3, y = 0$, the wavenumber is $\kappa = 25$ and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.
Figure 5.4.12: A U-shape scattering an incident wave originating from a point source. The point source location is $x = 3, y = 0$, the wavenumber is $\kappa = 45$ and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly $0$, a PML is added to reduce numerical reflections from the artificial boundary.
Figure 5.4.13: A U-shape scattering an incident wave originating from a point source. The point source location is $x = 3, y = 0$, the wavenumber is $\kappa = 65$ and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.
Figure 5.4.14: Mesh of square scatterer. The interior square boundary uses pure Dirichlet boundary conditions, and the outer boundary is a PML layer combined with Robin outgoing conditions.
The parameter $C$ from 2.2.33 is chosen to be $C = 2$. The boundary of the interior square uses Dirichlet boundary conditions of zero whereas the outer boundary conditions are prescribed by PML.

Figure 5.4.15: A square scattering an incident wave originating from a point source. The point source location is $x = 3.0, y = 0.1$, the wavenumber is $\kappa = 25$ and the internal box takes Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.
Figure 5.4.16: A square scattering an incident wave originating from a point source. The point source location is $x = 3.0, y = 0.1$, the wavenumber is $\kappa = 45$ and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.
Figure 5.4.17: A square scattering an incident wave originating from a point source. The point source location is $x = 3.0, y = 0.1$, the wavenumber is $\kappa = 65$ and the boundary of the internal box uses Dirichlet boundary conditions set to be uniformly 0, a PML is added to reduce numerical reflections from the artificial boundary.
Chapter 6

Conclusions and Future Work

This thesis explored the use of plane wave discontinuous Galerkin methods in several contexts not yet explored: applications to highly heterogeneous problems and problems with a point source, and high performance matrix-free implementations of these ideas on graphics processing units. Also examined were additive Schwarz domain decomposition based preconditioners, however achieving competitive speedup using these in a matrix-free fashion remains elusive. The work here builds on the PWDG method put forward by Hiptmair, Moiola, and Perugia in [71] for locally refined meshes, and also work by Howarth, Childs, and Moiola in [78] to extend this method to handle very heterogeneous wavenumbers as well as point sources. Furthermore this thesis heavily exploits the nature of plane-waves in a DG method to achieve high performance in a GPU implementation.

Despite the negative results of the domain decomposition preconditioners, this thesis shows that the low memory nature of the PWDG method can be exploited to solve problems of very high frequency on problems of potential practical interest, these have been solved entirely on a single GPU. One thing to note though that although the Schwarz preconditioners did not yield improvement in overall cost, there was a cutoff point at which the performance (in terms of total operator evaluation count - both coming from the preconditioner and from the FGMRES iterations) was roughly equivalent to that of the non preconditioned iteration. The significance of this is that the subdomains in the Schwarz method can be solved entirely independently of each
other, therefore this could be exploited to solve even larger problems by distributing
the Schwarz subdomains across multiple GPUs and relying on their local GMRES
solutions to drive the outer GMRES of the whole problem to convergence. In this
context it could significantly improve the communication overhead of simply solving
the unpreconditioned problem distributed over multiple GPUs, as this would require
repeated communication over the PCI bus for every iteration.

Additionally considering the very low memory requirement - but very high arith-
metic intensity - of the PWDG method presented in this thesis, a reasonable next
step could be to find ways to precondition the system which are often ignored because
of its additional memory overhead. Such methods could be for example aggressive
deflation based preconditioners, from which recent promising directions have been ex-
plored in the context of more classical discretizations to the Helmholtz equation see
e.g. [44] and [15] where “optimal” deflation is investigated in an attempt to circum-
vent the many problems associated with iterative methods applied to the Helmholtz
equation. Furthermore a common source of deflation vectors comes from a multigrid
coarsening hierarchy, and with recent multigrid work of forming coarse grids using
aggregate elements [11] which have a unique applicability to PWDG because it - un-
like polynomial elements - actually has no stability or numerical dependence on the
shape of underlying elements. This has been exploited also in recent work such as
e.g. the recent work of Weißer in the sequence of papers [136], [135], and the joint
work with Rjasanow in [114]. A possible direction of future work would be to take the
already successful solver framework built in this thesis and incorporate the new re-
search on deflation already mentioned with a systematic means of building deflation
vectors through the polygonal based Trefftz methods. Combining this with a dis-
tributed Schwarz method as earlier described could enable problems of significantly
larger scale than explored in this thesis, and may be an essential step in achieving scalable results in three dimensions.
Bibliography


[73] Ronald H. W. Hoppe, Xuejun Xu, and Huangxin Chen. Local multigrid on adaptively refined meshes and multilevel preconditioning with applications to problems in electromagnetism and acoustics. 2010.


[110] Reid Atcheson and David Medina. OCCA.jl.


