CS 714

HELMHOLTZ DOMAIN DECOMPOSITION IMPLEMENTATION

Noah Cohen Kalafut
Computer Science Doctoral Student
University of Wisconsin-Madison
nkalafut@wisc.edu
https://github.com/0afish1/CSC-714

December 10, 2020

1 Abstract

A method outlined by Christiaan C. Stolk is outlined and implemented. The method is designed to be a rapid solver for the Helmholtz equation with potential for parallelization. This is achieved through domain decomposition with PML padding on internal boundaries. The technique allows for isolation of forward and backward-going waves. Starting from a rough estimate, the algorithm calculates a correction using preconditioned GMRES. This particular implementation does not involve multifrontal methods, but such methods would likely contribute to a significant decrease in computation time.

2 Background

2.1 The Helmholtz Equation

There are a variety of techniques that will be used in this implementation, but it may be useful to first examine the Helmholtz equation itself in order to better understand its purpose and how it should function.

Many publications have gone over the derivation of the Helmholtz equation in greater detail than I ever could [1]. So, I will make this brief.

We start with the Wave equation

$$u_{tt} = c^2 \Delta u$$

Let's consider the 2D case. If we assume that the spatial and time variables are separable, we can say $u(x, y, t) = v(x, y)w(t)$. Then,

$$\frac{w_{tt}}{c^2 w} = \frac{\Delta v}{v}$$

Notice that the left side is only a function of t, while the right is only a function of x, y. Therefore, they share no common variable and both sides must be equal to some constant. Typically, a number of the form $-k^2$ is used to represent this constant for reasons that will be discussed shortly. We can then say

$$w_{tt} = -k^2 c^2 w \qquad\qquad \Delta v = -k^2 v$$

We have essentially now separated the Wave equation into two distinct components. The Helmholtz equation is the latter.

Notice, in the first equation, $w_{tt} = (-i\omega)^2 w = -\omega^2 w$ in the frequency domain with ω as the angular frequency. Then, $k = \frac{\omega}{c}$.

In this application, k is known as the wave number.

Given that k is constant, note the relationship between ω and c. As one changes by some factor, if the other shrinks by that same factor, the solution is the same.

2.2 Perfectly Matched Layers

Perfectly matched layers, henceforth referred to as **PML**, is primarily used to create reflectionless boundaries. This is often used to approximate simulations on infinite or practically infinite domains.

Rather than a strict, one-point boundary, PML can be thought of as a buffer zone outside of the simulation area that mitigates the outgoing waves before reaching ultimate, normally homogeneous Dirichlet boundary conditions such that there is minimal reflection. In theory, the region of interest is not affected by this change and the PML region can be made arbitrarily small. In practice, discretization requires the waves to decay over a distance so that the solution in the area of interest is minimally affected.

PML is explained fantastically in [2]. Since this is mainly focused on implementation, I will summarize the major theoretical points within.

Assuming that points far from the region of interest are homogeneous and time-invariant, a 2D region of interest can be described as a superposition of plane waves

$$\sum_{k,\omega} W_{k,\omega}(y) e^{i(kx-\omega t)}$$

k, ω are the wave vector and frequency, respectively.

Then, for some complex $\tilde{x}(x) = x + if(x)$, $e^{i(k\tilde{x}(x)-\omega t)} \to 0$ as $f(x) \to \infty$. This is the primary idea behind PML.

To implement this equation in our method, we can simply substitute $\partial x \to (1 + if'(x))\,\partial x$ for some function $f(x)$. For reasons that will become apparent in the coming example, $f'(x)$ is normally of the form $\frac{\sigma_x}{\omega}$.

2.2.1 Example

Following [2], let's take the 1D wave equation

$$u_{tt} = u_{xx} \iff u_t = v_x, v_t = u_x$$

In the frequency domain, we can say

$$u_t = -i\omega u \qquad\qquad v_t = -i\omega v \tag{1}$$

Applying our substitution $\partial x \to \left(1 + i\frac{\sigma_x}{\omega}\right)\partial x$, we get

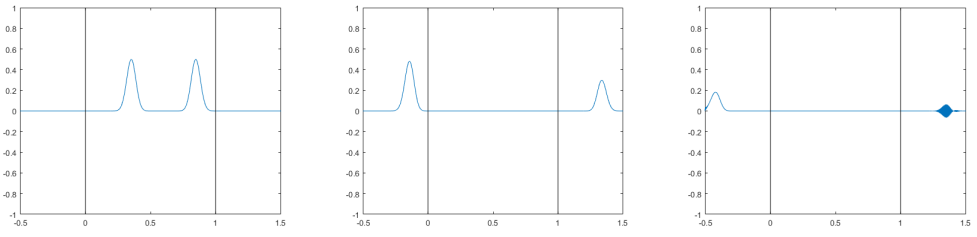
$$v_x = -i\omega u + \sigma_x u \qquad\qquad u_x = -i\omega v + \sigma_x v \tag{2}$$

We can then substitute and solve for the derivatives with respect to t

$$u_t = v_x - \sigma_x u \qquad\qquad v_t = u_x - \sigma_x v \tag{3}$$

This example can be seen in figure 1.

Figure 1: 1D wave equation with PML (*1D_Wave_PML.m*)



2.3 Domain Decomposition

Domain decomposition is the splitting of a domain into multiple sub-domains. This can be done in any number of dimensions. The crux of this method is boundary conditions between subdomains. Applications of this method vary widely, so most of the definition will be left to the following section.

The usefulness of this method might not be immediately apparent, as calculations are still being done across the whole of the original domain. However, this approach can allow for implementations of schemes that can run in parallel – greatly decreasing computing time.

Additionally, calculations on thin subdomains take a significantly shorter amount of time to converge than on the whole domain. This can also be true in the case of the total time required to calculate these solutions on each subdomain [3].

3 Methodology

All of the following is in service of an implementation of [4]. This is just a broad overview of the theories upon which the algorithm is built and serves mainly as a summary of the content within the paper.

The main motivation of this method lies in its fast convergence rate, especially (theoretically) with regard to multifrontal methods on subdomains of reduced dimensionality [3]. This implementation will not utilize multifrontal methods, and will instead focus on the concepts behind the rest of the algorithm.

3.1 Theoretical Formulation

We will be basing our method upon the Helmholtz equation as follows[1]

$$Au = -(\Delta + k^2)u = f$$

For implementing PML, we need to perform the substitution previously derived

$$\partial x \to \left(1 + i\frac{\sigma_x}{\omega}\right)\partial x$$

Let our domain be $[0, N] \times [0, 1]$. We will use PML boundary conditions.

We can define boundaries to split the domain into J subdomains

$$0 = b_0 < \cdots < b_J = N$$
$$b_1, \ldots, b_{J-1} \in \mathbb{Z} + .5$$

We denote $D^{(j)}$ as the region (b_{j-1}, b_j) and $\Omega^{(j)}$ as the same region with PML padding of length N_{pml} on either side. We denote $A^{(j)}$ as the Helmholtz operator on $\Omega^{(j)}$.

In [4], the value of k used on either end of the region is preserved throughout the PML layer. More formally,

$$k^{(j)}(x, y) = \begin{cases} k(x, y) & b_{j-1} < x < b_j \\ k(b_{j-1}, y) & x < b_{j-1} \\ k(b_j, y) & x > b_j \end{cases}$$

Similarly, our target function, f, is padded with zeros on the PML layers

$$f^{(j)}(x, y) = \begin{cases} f(x, y) & b_{j-1} < x < b_j \\ 0 & \text{otherwise} \end{cases}$$

We solve $v^{(j)}$ on $\Omega^{(j)}$ for the forward-propagating step (from 1 to J)

$$A^{(j)}v^{(j)} = f^{(j)} - 2\delta(x - b_{j-1})\partial_x v^{(j-1)}(b_{j-1}, \cdot) \tag{4}$$

The second term of (4) fulfills the role of a boundary condition in typical domain decomposition. The Dirac-delta function is used to insert the impulse $\partial_x v^{(j-1)}(b_{j-1}, \cdot)$ from the end of $D^{(j-1)}$, to the beginning of our current subdomain, $D^{(j)}$. Due to the PML padding in $\Omega^{(j)}$, this should only include forward-going waves with minimal reflections.

Taking the residual of our solution, we get

$$g = f - Av$$

[1] [4] suggests a formulation for non-constant k, which is easily extended from our argument.

We can then begin with our backward-propagation.

We define new boundaries

$$0 = \tilde{b}_0 < \cdots < \tilde{b}_J = N$$
$$\tilde{b}_1, \ldots, \tilde{b}_{J-1} \in \mathbb{Z} + .5$$
$$\forall (j, k \in [1, J-1]), \tilde{b}_j \neq b_k$$

On these new boundaries, we define $\tilde{D}^{(j)}, \tilde{\Omega}^{(j)}, \tilde{A}^{(j)}, g^{(j)}$.

We solve $w^{(j)}$ on $\Omega^{(j)}$ for the backward-propagating step (from J to 1)

$$\tilde{A}^{(j)} w^{(j)} = g^{(j)} + 2\delta(x - \tilde{b}_j)\partial_x w^{(j+1)}(\tilde{b}_j, \cdot) \tag{5}$$

The approximate solution to the Helmholtz equation, $v + w$, is then obtained. We define the operator P as $f \mapsto v + w$, which will be used as a preconditioner for GMRES.

3.2 Discretization

We can use a second-order accurate discretization

$$\Delta u \approx \alpha_x^2 \frac{U_{x+h} - 2U_x + U_{x-h}}{h^2} + \alpha_y^2 \frac{U_{y+h} - 2U_y + U_{y-h}}{h^2}$$

for $\alpha_x = \frac{1}{1 + i\frac{\sigma_x}{\omega}}$.[2]

Keep in mind that we will have homogeneous Dirichlet boundary conditions outside of the PML regions, meaning that no further significant adjustment needs to be made to our laplacian operator or A.

Also notice that, while $\alpha_y \neq 1 \iff y \notin [0, 1]$, α_x is used with much higher frequency, since we perform most calculations on $\Omega^{(j)}$.

Let's define n_x, where $n_x + 2w_{\text{PML}} + 1$ is the total number of points being evaluated on the x-axis, including PML layers on either end.

We can then define our boundaries

$$b_j = w_{\text{PML}} + \frac{j}{J} n_x$$

Also,

$$\tilde{b}_j = \begin{cases} b_0 & j = 0 \\ b_J & j = J \\ b_j + 1 & \text{otherwise} \end{cases}$$

For equations (4) and (5), we will use an upwind first-order scheme to approximate ∂_x

$$u_x \approx \frac{U_{x+h} - U_x}{h} \text{ or } \frac{U_x - U_{x-h}}{h}$$

Keep in mind that the 'upwind' direction will change depending on which equation (4 or 5) is being used.

Lastly, we can approximate the Dirac delta function as

$$\delta(x - b) = \begin{cases} \frac{1}{2h} & \lceil \frac{x-b}{h} \rceil + \lfloor \frac{x-b}{h} \rfloor = 1 \\ 0 & \text{otherwise} \end{cases}$$

In English, δ is non-zero when x borders the boundary b. Remember that the boundaries are defined at half (or approximately half) points.

[2] This is a simplification of the discretization from [4] which assumes $\alpha_x = \alpha^-_{x+\frac{h}{2}} = \alpha^+_{x-\frac{h}{2}}$.

3.3 General Algorithm

First, we can perform a backward pass, solving[3]

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

We can then calculate the residual, mainly for forward-going waves

$$\phi = f - A\tilde{u}$$

We will then solve

$$A\psi = \phi$$

by using GMRES on the right preconditioned system

$$APv = \phi \qquad\qquad \psi = Pv$$

This can be used to calculate our final u, observe:

$$u = \psi + \tilde{u}$$

The preconditioning here, combined with the proposed improved runtime of multifrontal methods on reduced-dimension subdomains [3], is the key idea behind this method's efficiency.

4 Numerical Results

As suggested by [4], $h\omega$ is being kept constant in this implementation.

The implementation of [4] used dynamically creates the matrix A for each use case. The time taken for the creation of these matrices is subtracted when timing the algorithm, as this process would hopefully be relegated to a one time cost in a practical application. This time is so pronounced, that it actually accounts for most of the running time of the implementation. In practice, a cache of previously created A can be added to speed up simulation time, as the algorithm itself does not take long. This also applies to the standard solver.

For conversion to real-world scaling, the grid spacing h_x was assumed to equal $h_y = \frac{1}{n_y}$, regardless of the ratio $n_x : n_y$.

Rather than a direct solver, solves of small u utilize a standard Jacobi method that continues until an iteration varies by less than 10^{-12} from the previous. This value was selected because it generally allows our method to converge without oscillation.

Even without the use of multifrontal methods, the algorithm shows an impressive convergence rate. Each additional GMRES iteration often corresponds to a reduction of the residual by 10^{-1} until convergence, which can be seen in table 4.

Memory, both RAM and virtual, turned out to be limiting factors in this implementation. Because some calculations need to be done on the whole matrix, it is difficult to avoid this limitation. If there is to be a future implementation, perhaps another technique can be used that avoids non-sparse matrix storage while still providing the speed benefits of a matrix.

The timing and convergence results for various examples $h\omega = 1, f(x, y) = sin(2\pi x)sin(2\pi y)$ can be seen in figures (4,2,3).

The algorithm tended to converge after 2 or 3 GMRES iterations. For most n, the algorithm appears to be significantly slower than basic iterative techniques. As n increases, this ratio declined, making our algorithm $\frac{1}{4}$ the speed of standard methods. Perhaps this effect would continue as n grows larger. This is likely a result of the lack of parallelization and multifrontal methods. Parallelization may seem impossible, but some suggestions are made in [4].

5 Conclusions

The method has potential. Without directly testing the effect of multifrontal methods, it is difficult to say whether or not this technique would overcome traditional solvers. The consistency in convergence and the ability to reliably find a solution equivalent to traditional solvers in precision with so few iterations is impressive. It would be great to see this technique implemented on a large scale simulation, so its time complexity could be reliably analyzed.

[3] Keep in mind that $\tilde{f} = f$. As previous, the tilde indicates the use of backward boundaries during the solve.

n_x	n_y	w_{PML}	J	Runtime	Residual	GMRES Iterations
200	50	3	50	1.774	$1.4 * 10^{-13}$	1
				1.912	$5.6 * 10^{-14}$	2
				2.167	$3.3 * 10^{-16}$	3
200	50	3	100	1.865	$1.5 * 10^{-13}$	1
				1.965	$3.6 * 10^{-14}$	2
				2.477	$2.2 * 10^{-16}$	3
200	50	3	150	1.946	$1.4 * 10^{-13}$	1
				2.046	$4.1 * 10^{-14}$	2
				2.526	$2.2 * 10^{-16}$	3
200	50	5	100	2.116	$1.6 * 10^{-13}$	1
				2.387	$2.9 * 10^{-14}$	2
				3.092	$2.2 * 10^{-16}$	3
300	50	3	100	3.391	$7.5 * 10^{-14}$	1
				3.578	$4.9 * 10^{-14}$	2
				4.777	$3.3 * 10^{-16}$	3
300	50	5	100	4.689	$8.5 * 10^{-14}$	1
				5.227	$6.6 * 10^{-14}$	2
				6.537	$2.2 * 10^{-16}$	3
200	100	3	100	5.297	$4.4 * 10^{-16}$	1
				5.975	$3.3 * 10^{-16}$	2

Figure 2: 200×50 grid with $J, w_{\text{PML}} = 50, 3$ run until convergence.

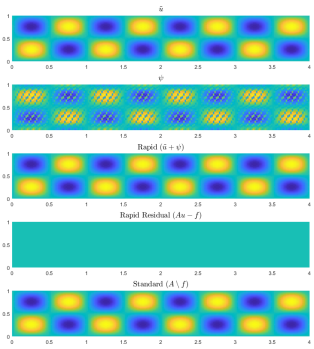
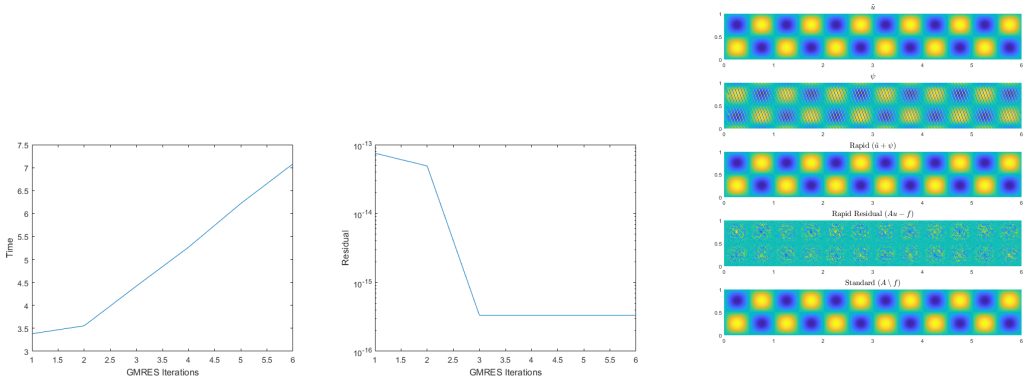


Figure 3: 300×50 grid with $J, w_{\text{PML}} = 100, 3$ run until convergence.



References

[1] Li Zhang. The finite difference method for the helmholtz equation with applications to cloaking. `https://math.missouristate.edu/Assets/Math/li.pdf`.

[2] Steven G. Johnson. Notes on perfectly matched layers. `https://math.mit.edu/~stevenj/18.369/pml.pdf`, 2010.

[3] Björn Engquist and Lexing Ying. Sweeping preconditioner for the helmholtz equation: Moving perfectly matched layers. *SIAM Journal on Multiscale Modeling and Simulation*, 9, 07 2010.

[4] Christiaan C. Stolk. A rapidly converging domain decomposition method for the helmholtz equation. Journal of Computational Physics 241 (2013) 240-252, 2012.