THESIS

AN ANALYSIS OF DOMAIN DECOMPOSITION METHODS USING DEAL.II

Submitted by

Christina Rigsby

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2021

Master's Committee:

Advisor: Simon Tavener

Wolfgang Bangerth
Jiangguo Liu
Paul Heyliger

ABSTRACT

AN ANALYSIS OF DOMAIN DECOMPOSITION METHODS USING DEAL.II

Iterative solvers have attracted significant attention since the mid-20th century as the computational problems of interest have grown to a size beyond which direct methods are viable. Projection methods, and the two classical iterative schemes, Jacobi and Gauss-Seidel, provide a framework in which many other methods may be understood. Parallel methods, or Jacobi-like methods are particularly attractive as Moore's Law and computer architectures transition towards multiple cores on a chip. We implement and explore two such methods, the multiplicative and restricted additive Schwarz algorithms for overlapping domain decomposition. We implement these in deal.II software, which is written in C++ and uses the finite element method. Finally, we point out areas for potential improvement in the implementation and present a possible extension of this work to an agent-based modeling prototype currently being developed by the Air Force Research Laboratory's Autonomy Capability Team (ACT3).

ACKNOWLEDGEMENTS

I would like to acknowledge Wolfgang Bangerth as the author of the original step-6 code that I have modified as well as his regular guidance over the course of the Fall 2020 semester. I would also like to thank Simon Tavener for further weekly discussion on concepts relevant to the content of this program. Finally, I thank Jiangguo Liu and Paul Heyliger for their instruction in relevant courses, some of whose material is included herein.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Motivation

Strong incentive for the development of domain decomposition algorithms and its growth into an active research area has been the increasing availability of parallel computers [2]. Since the advent of scientific computing, the size of problems of interest has also grown substantially. Domain decomposition methods provide a means to divide and conquer such problems in a way that can leverage the computational capabilities of multiple processors. Increasing the efficiency of such methods while minimizing the incurred cost on the global solver has been explored [2]. In the following work, we continue this effort by eliminating the concept of a global solution entirely. Instead, we focus on solving problems of a reduced size defined on individual subdomains, only use these solutions through cycles of subdomain solves, but provide a method of global solution construction in case it is needed for the user's analysis and highlight the relationship among additive Schwarz, restricted additive Schwarz and multiplicative Schwarz algorithms.

1.2 Relevant Work

The Schwarz alternating algorithm is the first known domain decomposition method, originating in 1870 [3], and can be applied to partial differential equations after discretization. In the 1930s, the introduction and development of the finite difference method enabled such discretization. About 30 years later, the name of the finite element method made its first appearance in literature[1], but key ideas were introduced years earlier. For example, in 1943[2], Courant proposed the use of continuous piecewise linear approximating functions defined for a triangulation adapted

[1]Clough. The finite element method in plane stress analysis. Proceedings of Second ASCE Conference on Electronic Computation, Vol. 8, Pitssburg, Pensylvania pp. 345-378. 1960.

[2]Courant. Variational methods for the solution of problems of equilibrium vibrations. Bullitin of the American Mathematical Society, 49. pp. 1-23. 1943.

to the domain of the partial differential equation to be solved, a key concept in the finite element method [4]. Methods such as these, the finite difference method and the finite element method, can be recast as multigrid methods, which appear in the literature shortly afterward. In 1964, Fedorenko published his formulation of a multigrid algorithm for the standard five-point finite difference discretization of the Poisson equation, work that is now considered the first 'true' multigrid publication. It was later generalized by Bachvalov in 1966 [5]. Although multigrid methods have broad scope and applicability, they can be considered efficient in solving the linear systems resulting from this discretization of differential equations [5]. Another powerful technique is that of conjugate direction methods, of which the conjugate gradient method is one of the most well-known. For medium-sized linear problems, conjugate gradient methods are about as efficient as multigrid methods in accelerating basic iterative methods, with the added bonus that they are easier to program, but with the restriction that they are limited to linear problems [5]. Although these methods and others have proven useful, we focus on the theoretical and implementational aspects of the Schwarz algorithms. "[O]verlapping Schwarz methods have very good convergence properties" [6], which we explore through an analysis of their foundational iterative methods. "In many circumstances the resulting iterative method has a convergence rate independent of the problem size and number of processors" [6] and allow for concurrent multiscale coupling [3], making this method one worth implementing in robust software such as deal.II.

1.3 Contributions

In [7], Smith, Bjorstad and Gropp susinctly state a noteworthy difficulty in implementing overlapping domain decomposition methods is that "there is no simple data structure that can represent both the global vector and the elements of the subdomains". Their proposed solution, one that seems to traditionally be followed ([6], [8], [9], [10]) is to use direct representations of restriction matrices then multiplied to global matrices and vectors to retrieve smaller pieces defined on each subdomain. Contrastingly, we present the multiplicative, additive, and restricted additive Schwarz algorithms for overlapping subdomains but construct implementation strategies that forgo

the global solution construction step in the traditional formulation of the Schwarz algorithms required only to restrict back down to individual subdomains. We save memory space by preserving the reduced problem size that domain decomposition methods afford. Additionally, we minimize computational time required to complete a given number of solving cycles by implementing boundary conditions directly from subdomain solutions rather than using these subdomain solutions to construct a global solution from which we then impose the equivalent boundary conditions.

We discuss and provide code for implementation of these modified methods in deal.II while showing an example of the computational and memory costs avoided with our revised approach. Ideas for its optimization are included in case the user requires the construction of a global solution for analytical purposes. Lastly, we introduce an ongoing project supported by the Air Force Research Laboratory's Autonomy Capability Team (ACT3) that could potentially benefit from (1) domain decomposition methods' capability to reduce problem size and from (2) broadening its diffusion model to one represented by a partial differential equation to which the techniques included herein would then be applicable.

Chapter 2

Mathematical Background for Schwarz Domain Decomposition Methods

2.1 Iterative Methods

Consider a linear system $Ax = b$. Using coefficient matrix splitting techniques, we can convert this system to an iterative scheme of the form

$$x^{\{k+1\}} = Rx^{\{k\}} + c, \quad k = 0, 1, 2, ...$$

We consider the conditions necessary for such iterative schemes to converge and whether the limiting solution satisfies the original linear system. We then address the rate of convergence.

A basic iterative technique can be developed by splitting $A = M - K$, the linear system becomes

$$(M - K)x = b,$$

or equivalently

$$Mx = Kx + b. \tag{2.1}$$

Turning this into an iterative scheme, we have

$$Mx^{\{k+1\}} = Kx^{\{k\}} + b, \quad k = 0, 1, 2, ...$$

by left multiplying by M^{-1},

$$x^{\{k+1\}} = M^{-1}Kx^{\{k\}} + M^{-1}b, \quad k = 0, 1, 2, ... \tag{2.2}$$

or

$$x^{\{k+1\}} = Rx^{\{k\}} + c, \quad k = 0, 1, 2, ..., \text{ where } R = M^{-1}K, c = M^{-1}b. \tag{2.3}$$

If convergence is attained, then the limit x^* satisfies (2.1), so $Mx^* = Kx^* + b \implies (M - K)x^* = b \implies Ax^* = b$, meaning the limit of the iterative scheme also satisfies the original linear system.

We first introduce two methods which differ in coefficient matrix splitting and subsequent re-arranging of the original linear system, conduct general convergence analysis for these techniques and others that can be generalized to the form (2.3), and draw conclusions for these two specific schemes from the general results.

2.1.1 Jacobi

Take $Ax = b$ and split $A = D - \widetilde{L} - \widetilde{U} = D(I - L - U)$ where

$$D = \begin{cases} A, & i = j, \\ 0 & \text{otherwise,} \end{cases} \qquad \widetilde{L} = DL = \begin{cases} -A, & j < i, \\ 0 & \text{otherwise,} \end{cases}$$
$$\text{and} \quad \widetilde{U} = DU = \begin{cases} -A, & i < j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

Then $(D - \widetilde{L} - \widetilde{U})x = b$. Rearranging this equation in the following two ways and formulating each as an iterative process results in the Jacobi and Gauss-Seidel algorithms respectively. Let

$$Dx = (\widetilde{L} + \widetilde{U})x + b \tag{2.5}$$

$$(D - \widetilde{L})x = \widetilde{U}x + b. \tag{2.6}$$

Formulating (2.5) iteratively as

$$Dx^{\{k+1\}} = (\widetilde{L} + \widetilde{U})x^{\{k\}} + b, \quad k = 0, 1, ...$$

and left multiplying by D^{-1}, we have

$$x^{\{k+1\}} = D^{-1}(\widetilde{L} + \widetilde{U})x^{\{k\}} + D^{-1}b, \quad k = 0, 1, ..., \tag{2.7}$$

so the i^{th} component of the solution vector is

$$x_i^{\{k+1\}} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{\{k\}} - \sum_{j=i+1}^{n} a_{ij}x_j^{\{k\}}}{a_{ii}} \tag{2.8}$$

where a_{ij} is the element in the original coefficient matrix A's i^{th} row and j^{th} column [11].

In (2.8), we see that Jacobi only uses old solution values to calculate new ones, implying that this algorithm is easily parallelizable. Instead of solving sequentially as the for loop suggests, the system can be solved in one computational time step by being split up among different processors. Also to be noted, implementing (2.8) requires the storage of two copies of each approximate solution (the old solution and the updated one), which is something to bear in mind with large problems.

2.1.2 Gauss-Seidel

Formulating (2.6) iteratively as

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

and left multiplying by $(D - \widetilde{L})^{-1}$,

$$x^{\{k+1\}} = ((D - \widetilde{L})^{-1})\widetilde{U}x^{\{k\}} + (D - \widetilde{L})^{-1}b. \tag{2.9}$$

Therefore, the i^{th} component of the solution vector is [11]

$$x_i^{\{k+1\}} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{\{k+1\}} - \sum_{j=i+1}^{n} a_{ij} x_j^{\{k\}}}{a_{ii}}.$$

As alluded to previously, we now use $x_j^{\{k+1\}}$ for $j = 1, ..., i-1$ in calculating $x_i^{\{k+1\}}$. The most recently computed solutions are used whenever possible. Solutions $x_j^{\{k\}}$ from the previous cycle are only used for j not yet computed in the current cycle (i.e. for $j = i+1, ..., n$). This makes Gauss-Seidel more economical in terms of memory because the new solution can overwrite the old. Additionally, as proven below, this results in convergence in less than or equal to the amount of time Jacobi would take to converge. This does, however, imply the restriction that Gauss-Seidel is inherently a sequential scheme. Computational workload cannot be split among different processors to compute solutions independent of one another; calculating $x_i^{\{k+1\}}$ relies on having $x_j^{\{k+1\}}$ available for $j = 1, ..., i-1$.

2.1.3 Convergence

After introduction to these two schemes and the intuition that Gauss-Seidel performs at least as well as Jacobi with respect to convergence time, we proceed with proof of this claim. Comparison of convergence time is only sensible if convergence is attained at all, so one important question that remains unanswered is "what criteria must be met for a system to converge using either of these iterative processes?". Before beginning convergence analysis, we introduce some notation.

Both Jacobi and Gauss-Seidel can be generalized to the form

$$x^{\{k+1\}} = Rx^{\{k\}} + c \tag{2.10}$$

where R is an iteration matrix and c is the right hand side both specific to the particular scheme used. For the Jacobi scheme, we denote this particular R by R_J and this particular c by c_J. Likewise, for the Gauss-Seidel, we denote this particular R by R_{GS} and this particular c by c_{GS}. From

(2.7), we see that R_J and c_J are defined

$$\begin{aligned} R_J :=& D^{-1}(\widetilde{L} + \widetilde{U}) \\ c_J :=& D^{-1}b. \end{aligned} \tag{2.11}$$

Similarly, from (2.9), we see that R_{GS} and c_{GS} are defined

$$\begin{aligned} R_{GS} :=& (D - \widetilde{L})^{-1}\widetilde{U} \\ c_{GS} :=& (D - \widetilde{L})^{-1}b. \end{aligned} \tag{2.12}$$

Additionally, we recall that:

- if $||R|| < 1$ for a subordinate norm, then an iterative scheme (2.10) converges for any initial guess.
- an iterative scheme (2.10) converges if and only if the spectral radius of the iteration matrix is strictly less than 1; i.e. if and only if $\rho(R) < 1$,

so Jacobi and Gauss-Seidel converge if either of these conditions are met. We use these two facts to analyze the convergence of these algorithms when the coefficient matrix is strictly row diagonally dominant, when A is such that $|a_{i,i}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}|, 1 \leq i \leq n$.

Convergence of Jacobi

Strict row diagonal dominance of A ensures that Jacobi converges. Proof of this relies on the theorem stating: if $||I - M^{-1}A|| < 1$ for some subordinate matrix norm (M defined as in Section 2.1 where $A = M - K$), then the sequence produced by (2.2) converges to the solution of $Ax = b$ for any initial vector $x^{\{0\}}$.

First we explicitly formulate $M^{-1}A$ and then show that $||I - M^{-1}A|| < 1$ for some subordinate matrix norm. Comparing (2.2) and (2.7), we see that $M = D$ and $K = \widetilde{L} + \widetilde{U}$, which does indeed satisfy $A = M - K$. So

$$M^{-1}A = \frac{a_{ij}}{a_{ii}} = \begin{bmatrix} a_{11}/a_{11} & a_{12}/a_{11} & ... & a_{1n}/a_{11} \\ a_{21}/a_{22} & a_{22}/a_{22} & ... & a_{2n}/a_{22} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}/a_{nn} & a_{n2}/a_{nn} & ... & a_{nn}/a_{nn} \end{bmatrix}. \tag{2.13}$$

Combining the assumption of strict row diagonal dominance of A with (2.13), $M^{-1}A$ is a matrix whose diagonal entries are one and non-diagonal entries of each row sum up to some number less than one. This means that $||I - M^{-1}A|| < 1$, which concludes the proof [12].

Convergence of Gauss-Seidel

Strict row diagonal dominance of A also ensures that Gauss-Seidel converges. To prove this, we show that $\rho(R_{GS}) < 1$ (where R_{GS} is defined in (2.12)), for which it is sufficient to show that $|\lambda| < 1$ for some eigenvalue λ of R_{GS}. Take x to be the associated eigenvector of the eigenvalue λ and scale it such that $||x||_\infty \leq 1$. By definition of infinity norm, then for some i_0 less than or equal to the dimension of x, $|x_{i_0}| = 1$ and $|x_j| \leq 1$ for $j \neq i_0$. Since λ is an eigenvalue of R_{GS} with the associated eigen vector x,

$$\lambda x = R_{GS}x.$$

So by substitution using equality (2.12),

$$\lambda x = (D - \widetilde{L})^{-1})\widetilde{U}x.$$

Left multiplying by $(D - \widetilde{L})$,

$$(D - \widetilde{L})\lambda x = \widetilde{U}x,$$

which is a system of n equations with n unknowns. Extracting the i_0^{th} equation from this system,

$$\lambda a_{i_0,i_0} = -\lambda \sum_{j=1}^{i_0-1} a_{i_0,j}x_j - \sum_{j=i_0+1}^{n} a_{i_0,j}x_j.$$

Applying triangle inequalities,

$$|\lambda||a_{i_0,i_0}| \leq |\lambda| \sum_{j=1}^{i_0-1} |a_{i_0,j}||x_j| + \sum_{j=i_0+1}^{n} |a_{i_0,j}||x_j|. \tag{2.14}$$

Observe that the right hand side of (2.14) is less than or equal to $|\lambda| \sum\limits_{j=1}^{i_0-1} |a_{i_0,j}| + \sum\limits_{j=i_0+1}^{n} |a_{i_0,j}|$ since $|x_j| \leq 1$. Therefore,

$$|\lambda||a_{i_0,i_0}| \leq |\lambda| \sum_{j=1}^{i_0-1} |a_{i_0,j}| + \sum_{j=i_0+1}^{n} |a_{i_0,j}|.$$

Combining $|\lambda|$-terms,

$$|\lambda| \Big(|a_{i_0,i_0}| - \sum_{j=1}^{i_0-1} |a_{i_0,j}| \Big) \leq \sum_{j=i_0+1}^{n} |a_{i_0,j}|.$$

Dividing both sides of the inequality by $|a_{i_0,i_0}| - \sum\limits_{j=1}^{i_0-1} |a_{i_0,j}|$,

$$|\lambda| \leq \frac{\sum\limits_{j=i_0+1}^{n} |a_{i_0,j}|}{|a_{i_0,i_0}| - \sum\limits_{j=1}^{i_0-1} |a_{i_0,j}|} \tag{2.15}$$

where $\frac{\sum\limits_{j=i_0+1}^{n} |a_{i_0,j}|}{|a_{i_0,i_0}| - \sum\limits_{j=1}^{i_0-1} |a_{i_0,j}|} < 1$ by strict row diagonal dominance of A. Indeed, because A is strictly row diagonal dominant,

$$\sum_{j=1}^{i_0-1} |a_{i_0,j}| + \sum_{j=i_0+1}^{n} |a_{i_0,j}| < |a_{i_0,i_0}|,$$

so subtracting $\sum\limits_{j=1}^{i_0-1} |a_{i_0,j}|$ from both sides,

$$\sum_{j=i_0+1}^{n} |a_{i_0,j}| < |a_{i_0,i_0}| - \sum_{j=1}^{i_0-1} |a_{i_0,j}|.$$

Dividing both sides of the inequality by $|a_{i_0,i_0}| - \sum_{j=1}^{i_0-1} |a_{i_0,j}|$,

$$\frac{\sum_{j=i_0+1}^{n} |a_{i_0,j}|}{|a_{i_0,i_0}| - \sum_{j=1}^{i_0-1} |a_{i_0,j}|} < 1. \tag{2.16}$$

Combining (2.15) and (2.16), we conclude that $|\lambda| < 1$, showing that Gauss-Seidel converges for solving a linear system with a strictly row diagonally dominant coefficient matrix.

2.1.4 Comparison of convergence rates for Jacobi and Gauss-Seidel

Now we show that, assuming Jacobi converges, Gauss-Seidel does so at least as fast as Jacobi. We can do this by assuming that Jacobi converges (we assume that $||R_J||_\infty < 1$) and proving that

$$||R_{GS}||_\infty \leq ||R_J||_\infty < 1 \tag{2.17}$$

where, for an mxn matrix A, $||A||_\infty = max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{i,j}| = \left|\left||A|e\right|\right|_\infty$ with $e = [1, 1, ..., 1]^\intercal$, an $n-$dimensional vector of ones. Also, $|A|$ is an mxn matrix with the absolute values of the entries of A as its entries ($|A| = [|a_{i,j}|]_{m\times n}$). Instead of directly proving (2.17), we can prove a stronger component-wise inequality

$$|R_{GS}| \cdot e \leq |R_J| \cdot e. \tag{2.18}$$

We first simplify our expressions for R_J and R_{GS}. From (2.4), we have

$$\begin{aligned} \widetilde{L} &= DL, \\ \widetilde{U} &= DU. \end{aligned} \tag{2.19}$$

Substituting these into the expression of R_J in (2.11), we have

$$R_J = D^{-1}(DL + DU).$$

After distributing the D^{-1} term,

$$R_J = L + U.$$

Similarly, substituting (2.19) into (2.12), we have

$$R_{GS} = (D - DL)^{-1}(DU),$$

which we algebraically rearrange

$$\begin{aligned} R_{GS} &= (D - DL)^{-1}(DU) \\ &= \Big(D(I - L)\Big)^{-1}(DU) \\ &= (I - L)^{-1}D^{-1}DU \\ &= (I - L)^{-1}U. \end{aligned}$$

So, we have shown that we can equivalently define R_J and R_{GS}

$$\begin{aligned} R_J &:= L + U, \\ R_{GS} &:= (I - L)^{-1}U. \end{aligned} \tag{2.20}$$

Now we proceed to proving (2.18). Surely, if the sum of a given row's entries in matrix $|R_{GS}|$ is larger than the sum of that row's entries in matrix $|R_J|$, then the maximum of these sums for matrix $|R_{GS}|$ is larger than the maximum of these sums for matrix $|R_J|$. Substituting the two equalities in (2.20) into (2.18), we have

$$\Big|(I - L)^{-1}U\Big| \cdot e \leq |L + U| \cdot e. \tag{2.21}$$

Because

$$\begin{aligned}
\left|(I-L)^{-1}U\right| \cdot e &\leq \left|(I-L)^{-1}\right| \cdot |U| \cdot e && \text{by the triangle inequality} \\
&= \left|\sum_{k=0}^{n} L^k\right| \cdot |U| \cdot e \\
&= \left|\sum_{k=0}^{n-1} L^k\right| \cdot |U| \cdot e && \text{since } L^n = 0 \\
&\leq \sum_{k=0}^{n-1} |L|^k \cdot |U| \cdot e && \text{by the triangle inequality} \\
&= \big(I - |L|\big)^{-1} \cdot |U| \cdot e,
\end{aligned}$$

the following inequality is yet a stronger statement to prove than (2.21):

$$\big(I - |L|\big)^{-1} \cdot |U| \cdot e \leq \big(|L| + |U|\big) \cdot e. \tag{2.22}$$

To show that the above inequality holds, we start with algebraic manipulations and instead show that an equivalent inequality holds. We have

$$\begin{aligned}
|U| \cdot e &\leq \big(I - |L|\big) \cdot \big(|L| + |U|\big) \cdot e && \text{left multiplying (2.22) by } \big(I - |L|\big) \\
&= \big(|L| + |U| - |L|^2 - |L| \cdot |U|\big) \cdot e && \text{by multiplication} \\
0 &\leq \big(|L| + |U| - |L|^2 - |L| \cdot |U|\big) \cdot e - |U| \cdot e && \text{by subtraction} \\
&= \big(|L| - |L|^2 - |L| \cdot |U|\big) \cdot e \\
&= |L| \cdot \big(I - |L| - |U|\big) \cdot e && \text{by factoring out } |L|,
\end{aligned}$$

which is true if $|L|$, $(I - |L| - |U|)$, and e are all nonnegative. Trivially, $|L|$ and e are nonnegative, so it remains to show that $0 \leq (I - |L| - |U|)$. By right multiplying by e, we instead show that $0 \cdot e \leq (I - |L| - |U|) \cdot e$ or that $0 \leq (I - |L| - |U|) \cdot e$.

We have

$$\begin{aligned}
0 &\leq (I - |L| - |U|) \cdot e \\
&= I \cdot e - \big(|L| + |U|\big) \cdot e \\
\big(|L| + |U|\big) \cdot e &\leq I \cdot e,
\end{aligned}$$

where $|L| + |U| = |R_J|$. So by substitution, we want to show

$$|R_J| \cdot e \leq e,$$

which holds by our initial assumption that $||R_J||_\infty < 1$ [13].

2.2 Projection Methods

A large and increasing number of projection methods have been developed that encompass various fields including statistics, optimization, and computational mathematics [14]. The projectors which characterize this extensive list of methods are special matrices and operators. More specifically, a projection matrix P has the following properties:

- P is a symmetric idempotent matrix (a matrix is idempotent if $P^2 = P$),
- If P is an $n \times n$ matrix and $rank(P) = r$, then P has r eigenvalues equal to 1 and $n - r$ eigenvalues equal to 0,
- $trace(P) = rank(P)$,
- P is positive semidefinite.

Take a linear system $Ax = b$ with residual $r = b - Ax$ ($0 = b - Ax^*$ for the exact solution, x^*). Using some method to provide a reasonable approximation C of A^{-1} and given an initial approximation x_0, a sequence of approximations can be formed recursively: $x_{k+1} = x_k + C_k r_k$, where $r_k = b - Ax_k$ and $s_k = x^* - x_k$. Indeed, take some approximate solution, x_k, where $Ax_k = b_k$. Then

$$A(x^* - x_k) = b - b_k \implies As_k = r_k \implies s_k = A^{-1}r_k \approx C_k r_k.$$

"A method of projection is one that assigns at any step a subspace defined by the $\rho_k \geq 1$ linearly independent columns of a matrix Y_k, and selects a u_k in such a way that if

$$C_k r_k = Y_k u_k$$

then

$$s_{k+1} = s_k - Y_k u_k$$

is reduced in some norm" [15]. Having this general formulation for projection methods, defining the Jacobi block technique will show it to be one of the many projection methods. First, we use blocks that allow for arbitrary groups of components to update and allow the blocks to overlap. So, in solving $Ax = b$, the problem is subdivided by splitting up the matrix A, the solution vector x, and the right hand side b in the following way

$$\begin{bmatrix} A_{11} & A_{12} & ... & A_{1p} \\ A_{21} & A_{22} & ... & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & ... & A_{pp} \end{bmatrix}, x = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_p \end{bmatrix}, b = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$

where A_{ij} are submatrices that can overlap (meaning two adjacent submatrices can share one or more elements of the original matrix A) and ξ_i and β_i are subvectors which can also overlap. So, in order to retrieve these subproblems, it is just a matter of restricting the original system down appropriately. After solving each subproblem, they can be enlarged to the size of the original problem using a prolongation operator and combined together. This is one form of domain decomposition, a method discussed more later. Consider a variable set $S = 1, 2, ..., n$ subdivided into subsets $S_1, S_2, ..., S_p$ such that

$$S_i \subseteq S, \bigcup_{i=1,...,p} S_i = S$$

where each S_i has cardinality n_i and is of the form

$$S_i = \{m_i(1), m_i(2), ..., m_i(n_i)\}.$$

To build the components needed to define a general Jacobi block iteration, let

$$V_i = [e_{m_i(1)}, e_{m_i(2)}, ..., e_{m_i(n_i)}], V_i \in \mathbb{R}_{n \times n_i}$$

and let

$$W_i = [\nu_{m_i(1)} e_{m_i(1)}, \nu_{m_i(2)} e_{m_i(2)}, ..., \nu_{m_i(n_i)} e_{m_i(n_i)}]$$

where $\nu_{m_i(j)}$ are weights such that

$$W_i^\intercal V_i = I.$$

These matrices respectively provide the restriction operator to shrink the original problem into a smaller block and the prolongation operator to blow a block's information up to the original size mentioned earlier. More specifically, the $n_i \times n_j$ submatrices A_{ij} are defined

$$A_{ij} = W_i^\intercal A V_j$$

and the subvectors ξ_i and β_i are defined

$$\xi_i = W_i^\intercal x, \beta = W_i^\intercal b.$$

$W_i^\intercal$ is the restriction operator, V_i is the prolongation operator, and $V_i W_i^\intercal$ is a projector from $\mathbb{R}$ to K_i, the subspace spanned by the columns $m_i(1), ..., m_i(n_i)$. Thus, since the component-wise form of the Jacobi iteration is

$$\xi_i^{\{k+1\}} = \frac{1}{a_{ii}} \left(\beta_i - \sum_{j=1, j \neq i}^{n} a_{ij} \xi_j^{\{k\}} \right), \quad i = 1, ..., n,$$

the block iteration is given by

$$\xi_i^{\{k+1\}} = \xi_i^{\{k\}} + A_{ii}^{-1} W_i^{\intercal}(b - Ax_k),$$

and is a projection method [16]. Block Gauss-Seidel can similarly be shown to be a projection method.

2.3 Domain Decomposition

Overlapping domain decomposition is an iterative algorithm in which the original domain is divided into smaller overlapping subdomains. The original problem is projected to each of the subdomains and solved together with appropriate boundary conditions. The global solution is obtained by combining the subdomain solution in a particular fashion which varies between different domain decomposition algorithms. The iteration proceeds until a desired level of convergence is attained, or until divergence is established. Splitting the problem in this way allows the original problem size to be reduced to smaller pieces that are easier to manage both in computational expense and memory capacity. The order in which these smaller problems are solved and how the global solution is constructed from these subdomain solutions are the means by which a variety of domain decomposition algorithms have been developed. It should be noted that the order of subdomain solves need not be sequential; certain strategies allow for parallelization. To ensure that a parallelized scheme results in the same solution as that attained by sequential subdomain solves, boundary conditions for any given subdomain need to be identical between the two methods. An understanding of how the global solutions are built and how the boundary conditions are applied is essential to understanding the differences and similarities between algorithms. Since this differs among algorithms, parallelization is discussed in further detail after the introduction of specific iterative algorithms: multiplicative, additive, and restricted additive Schwarz.

In addition to allowing for a reduction in problem size and parallelization, another advantage of domain decomposition is the flexibility with which the original problem can be defined. Higher

order partial differential equations can instead be defined as a collection of lower-order approximations on smaller regions. For example, for fluid flow problems with shocks, the fluid domain can be split into subdomains where subdomains containing parts of the fluid field without shocks can see a reduction in degrees of freedom with the use of complementary techniques like proper orthogonal decomposition while subdomains with shocks can maintain the full-order problem [17]. Because of the generality of domain decomposition, it is often paired in this way with additional mathematical methods to minimize computation time. Improvements can also result from more basic modifications such as simply changing the number of subdomains or their geometric positioning. There are ways to optimize the number of subdomains constructed by considering the number of processors available for computation. After devising a color scheme by which solving subdomains can be parallelized (such as those mentioned in later discussion), [7] suggests that, for the problems considered, about four to eight subdomains should be assigned to each available processor to maximize parallelization, which in turn would minimize computation time without surpassing the capabilities of any one processor. The layout of the subdomains can also be fine-tuned by taking advantage of the structure of the partial differential equation to be solved. For example, for a convection-dominated problem, "if it is possible to align the domains with the dominant flow direction locally, then significant improvements in performance can be achieved" [7]. Thus, domain decomposition is an approach in which a domain is spatially separated into subdomains and cyclical solves are performed on these, often performed with techniques that take advantage of the structure of the equation to be solved, the amount of processing power available, and advancement in solving procedures.

2.4 Discretization of Partial Differential Equations

2.4.1 Finite Element Method

After introducing concepts like block iterations and domain decomposition that include discretizing domains in order to solve problems in smaller chunks, it is now natural to need a way to do the same for the equation to be solved on the given domain. In the development of the men-

tioned concepts, it was assumed that the equation to be solved was linear. If provided a differential equation to be solved instead, how can it be expressed in terms of matrices and vectors so that restriction and prolongation operators can be applied to retrieve and solve smaller pieces of the original problem at a time? This is exactly the purpose that the finite element method serves.

We present the finite element method in the one-dimensional case for simplicity, but it can be extended to higher dimensions. Suppose we want to solve a boundary value problem that is in the form

$$-\frac{d}{dx}(a\frac{du}{dx}) = f \tag{2.23}$$

on a one-dimensional mesh whose elements are bounded by their respective nodes x_p and x_q, which are separated by a length h. The goal is to formulate this as the linear system

$$k_{ij}u_j = F_i \tag{2.24}$$

where k_{ij} is our coefficient matrix, F_i is our right hand side, and u_j is our vector of nodal values. From the nodal values, the finite element solution over each element can be approximated as algebraic polynomials [18]. For the following derivation, we seek to linearly approximate the finite element solution from the nodal values, meaning we assume solutions of the form $u(x) = \sum_{j=1}^{2} u_j\psi_j(x)$, where ψ_k are called the finite element approximation functions or shape functions. Their purpose is to interpolate nodal solution values on the edges connected to each node. To achieve this, we require that each node have its own shape function. A property of these functions is that they are only nonzero in the element containing their corresponding node. The shape functions have a value of one at their own node and, to ensure continuity, have a value of zero at the boundaries of the finite element on which they are defined. Take for example an h-long, 1-dimensional element oriented horizontally, bound by one node on the left and another on the right. Call the node on the left node 1 and say it is located at $x = x_p$. Call the node on the right node 2 and say it is located at $x = x_q$. Assuming that linear interpolation of the two nodal values

is desired, we define the shape function associated with node 1 as

$$\psi_1(x) = \frac{x_q - x}{h},$$

which does, indeed, have a value of one at $x = x_p$ and linearly decreases to a value of zero at the boundary of the finite element, at $x = x_q$. Similarly, the shape function for node 2 is defined

$$\psi_2(x) = \frac{x - x_p}{h},$$

having a value of one at node 2 and of zero at node 1.

Now we will formulate a linear system in the form of (2.24) for the boundary value problem (2.23) defined on a one-dimensional element. We start with the weak form of (2.23)

$$0 = \int_{x_p}^{x_q} v(x)\Big(-\frac{d}{dx}\Big(a\frac{du}{dx}\Big) - f\Big)\, dx.$$

Distributing the $v(x)$ term and adding $\int_{x_p}^{x_q} v(x) f\, dx$ to both sides,

$$-\int_{x_p}^{x_q} v(x)\Big(-\frac{d}{dx}\Big(a\frac{du}{dx}\Big)\Big)\, dx = \int_{x_p}^{x_q} v(x) f\, dx. \tag{2.25}$$

Integrating by parts,

$$-\int_{x_p}^{x_q} v(x)\Big(-\frac{d}{dx}\Big(a\frac{du}{dx}\Big)\Big)\, dx = \int_{x_p}^{x_q} a\frac{du}{dx}\frac{dv}{dx}\, dx - v(x)\Big(a\frac{du}{dx}\Big)\Big|_{x_p}^{x_p}. \tag{2.26}$$

Substituting (2.26) into (2.25) and rearranging,

$$\int_{x_p}^{x_q} a\frac{du}{dx}\frac{dv}{dx}\, dx = \int_{x_p}^{x_q} v(x) f\, dx + v(x)\Big(a\frac{du}{dx}\Big)\Big|_{x_p}^{x_p}. \tag{2.27}$$

Defining

$$P_1 = \Big(-a\frac{du}{dx}\Big)\Big|_{x_p}, \quad P_2 = \Big(a\frac{du}{dx}\Big)\Big|_{x_q},$$

then (2.27) becomes

$$\int_{x_p}^{x_q} a\frac{du}{dx}\frac{dv}{dx}\,dx = \int_{x_p}^{x_q} v(x)f\,dx + v(x_p)P_1 + v(x_q)P_2. \tag{2.28}$$

Defining our shape functions as above,

$$\psi_1(x) = \frac{x_q - x}{h} \quad \text{and} \quad \psi_2(x) = \frac{x - x_p}{h}.$$

Differentiating with respect to x,

$$\frac{d\psi_1}{dx} = -\frac{1}{h} \quad \text{and} \quad \frac{d\psi_2}{dx} = -\frac{1}{h}.$$

We assume solutions of the form $u(x) = \sum_{j=1}^{2} u_j\psi_j(x)$, whose derivative with respect to x is

$$\frac{du}{dx} = \sum_{j=1}^{2} u_j\frac{d\psi_j}{dx}. \tag{2.29}$$

Also, let

$$v_i(x) = \psi_i(x). \tag{2.30}$$

So, we have

$$\frac{dv}{dx} = \frac{d\psi_i}{dx}. \tag{2.31}$$

By substitution of (2.29), (2.30), and (2.31) into (2.28),

$$\int_{x_p}^{x_q} a\sum_{j=1}^{2} u_j\frac{d\psi_j}{dx}\frac{d\psi_i}{dx}\,dx = \int_{x_p}^{x_q} \psi_i(x)f\,dx + \psi_i(x_p)P_1 + \psi_i(x_q)P_2 \text{ for } i = 1, 2; j = 1, 2. \tag{2.32}$$

Because $\psi_1(x_q) = 0, \psi_1(x_p) = 1, \psi_2(x_q) = 1,$ and $\psi_2(x_p) = 0,$

$$\int_{x_p}^{x_q} a\sum_{j=1}^{2} u_j\frac{d\psi_j}{dx}\frac{d\psi_i}{dx}\,dx = \int_{x_p}^{x_q} \psi_i(x)f\,dx + P_i \text{ for } i = 1, 2; j = 1, 2$$

$$\implies k_{ij}u_j = F_i \text{ where } \begin{cases} k_{ij} = \int_{x_p}^{x_q} a \frac{d\psi_j}{dx} \frac{d\psi_i}{dx}\, dx, \\ F_i = \int_{x_p}^{x_q} \psi_i(x) f(x)\, dx + P_i \end{cases} .$$

[18]

Chapter 3

Schwarz Algorithms and Implementation

3.1 Schwarz Algorithms

3.1.1 Multiplicative Schwarz

Defining $H^1_{D_k}(\Omega_i) \equiv \{v \in H^1(\Omega_i) |\ v = u^{\{k+(i-1)/p\}} \text{ on } \partial\Omega_i\}$, letting $a_i(\cdot,\cdot)$ denote the restriction of $a(\cdot,\cdot)$ to Ω_i and $a_{ij}(\cdot,\cdot)$ the restriction of $a(\cdot,\cdot)$ to $\Omega_i \cap \Omega_j$, and letting $l_i(\cdot)$ be the restriction of $l(\cdot)$ to Ω_i we present the multiplicative Schwarz method in Algorithm 1:

Algorithm 1 Overlapping multiplicative Schwarz domain decomposition

Given $u^{\{0\}}$ defined on Ω

for $k = 0, 1, 2, \ldots, K-1$ **do**

for $i = 1, 2, \ldots, p$ **do**

Find $\widetilde{u}^{\{k+i/p\}} \in H^1_{D_k}(\Omega_i)$ such that

$$a_i\big(\widetilde{u}^{\{k+i/p\}}, v\big) = l_i(v), \quad \forall v \in H^1_0(\Omega_i). \tag{3.1}$$

Let

$$u^{\{k+i/p\}} = \begin{cases} \widetilde{u}^{\{k+i/p\}}, & \text{on } \overline{\Omega}_i, \\ u^{\{k+(i-1)/p\}}, & \text{on } \Omega \backslash \overline{\Omega}_i. \end{cases} \tag{3.2}$$

end for

end for

Here, k indicates the cycle number while i indicates the subdomain number [8].

Algorithm 1 provides a method to compute $u^{\{k+i/p\}}$ on Ω for each cycle k. This is the solution on the entire domain, $\bigcup_{i=1}^{p} \Omega_i$. We start the algorithm with $u^{\{0\}}$, which only need satisfy the given boundary conditions for Ω. From here, $u^{\{1/p\}}$ can be found after computing $\tilde{u}_i^{\{i/p\}}$on $\overline{\Omega}_i$

for $i = 1, ..., p$. So that each $u^{\{k+i/p\}}$ is continuous along $\partial\Omega_i$, we impose the already computed $u^{\{k+(i-1)/p\}}$ as a boundary condition along the interface between $\overline{\Omega}_i$ and $\Omega \backslash \overline{\Omega}_i$ to compute $\tilde{u}_i^{\{k+i/p\}}$. In other words, this algorithm boils down to solving on each of the p subdomains, computing the solution for the entire domain from these, imposing this global solution as boundary conditions on each subdomain for the next solve. We do this iteratively until $K - 1$ cycles have been complete.

As previously referenced, since the global solution is only needed in setting boundary conditions for solving on individual subdomains, its behavior along these bounding edges only need be known. In Algorithm 1, we see that the global solution is updated after every subdomain solve. This means that no matter how many subdomains overlap a given edge, the boundary condition imposed on that edge comes only from the solution of the subdomain solved most recently. To illustrate, consider a domain split into four subdomains in a 2x2-fashion:

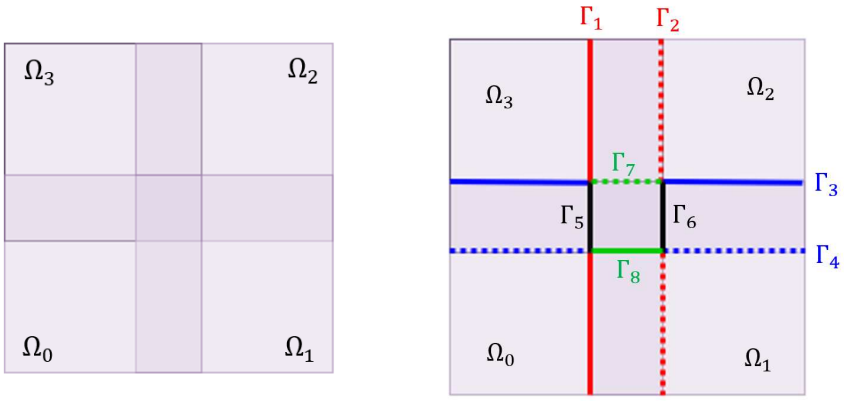


Figure 3.1: (a) Overlapping subdomain layout. (b) Domain with edge boundary_ids indicated for multiplicative Schwarz implementation.

Here, subdomains are solved in the same order of their numbering – counterclockwise starting with the subdomain in the bottom left. If the global solution were to be constructed, it would keep all of the most recently-computed subdomain solution (the solution from Ω_3), and a decreasing amount of older solutions. So, there exists a partition of the global domain that defines regions on which individual subdomain solutions from the current sweep are preserved, as shown in Figure 3.2.

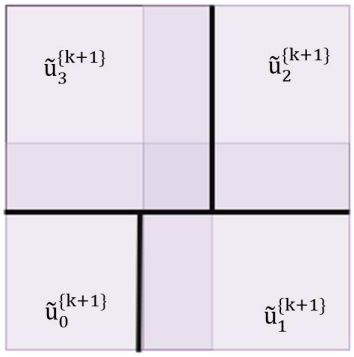


Figure 3.2: Partition preserving specific subdomain solutions in each region is overlaid on global domain. $\tilde{u}_i^{\{k+1\}}$ indicates that subdomain i's solution from cycle $k+1$ is preserved in the given region.

Therefore, in observing how the global solution is constructed along the edges of the subdomains, we can determine boundary conditions for all subdomain problems. We see that the boundary condition for the entire top edge of Ω_0 comes from Ω_3; the boundary condition on Γ_7 comes from Ω_3 rather than Ω_2 because Ω_3 was solved more recently than Ω_2. Similarly, the boundary condition for the bottom portion of Ω_0's right edge comes from Ω_1, and that for the remainder of Ω_0's right edge comes from Ω_3. In this way, we can impose boundary conditions for all subdomain solves, as summarized in Table **??**.

Therefore, to implement parallelization, we need only ensure that the order in which geometrically adjacent subdomains (subdomains whose solutions are boundary conditions for one another) are solved is preserved. Neighboring subdomains cannot be solved simultaneously, but ones whose intersections are empty can be. Therefore, an order-inducing color scheme can be devised such that only subdomains with empty intersections are of the same color, while using the minimum possible number of distinct colors to maximize parallelization. For example, assuming that the subdomains are arranged in a grid-like fashion (in the sense of having clear rows and columns, but still having non-zero overlap), they can be solved in parallel with a coloring system similar to the Red-Black scheme. The difference being that two Red-Black schemes would be spliced together, avoiding the overlap of subdomains of the same color in consecutive rows, as in Figure 3.3.

Table 3.1

Subdomain	Edge	Subdomain solution imposed as a BC
Ω_0	Γ_3	Ω_3
	Γ_7	Ω_3
	Γ_6	Ω_2
	Γ_2	Ω_1
Ω_1	Γ_1	Ω_0
	Γ_5	Ω_3
	Γ_7	Ω_3
	Γ_3	Ω_2
Ω_2	Γ_4	Ω_1
	Γ_8	Ω_1
	Γ_5	Ω_3
	Γ_1	Ω_3
Ω_3	Γ_2	Ω_2
	Γ_6	Ω_2
	Γ_8	Ω_1
	Γ_4	Ω_0

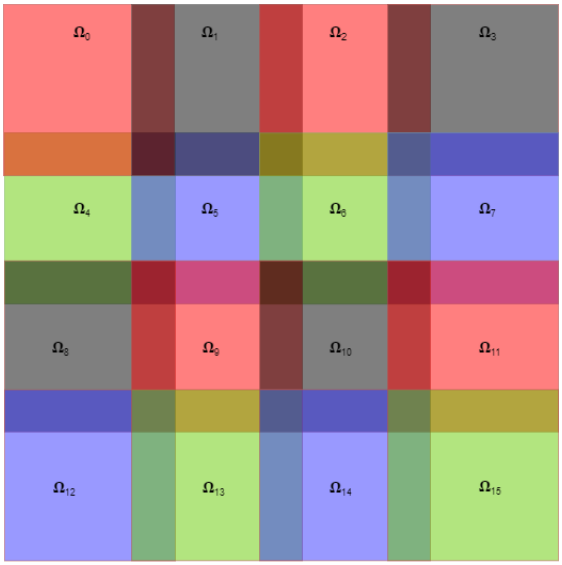


Figure 3.3: Example ordering scheme that would enable parallelization of multiplicative Schwarz algorithm.

For this parallelized scheme, there exists a sequential scheme with identical global solutions after each cycle of subdomain solves, should the global solution (unneccessarily) be constructed. Assuming that the order corresponding to Figure 3.3 is red, black, purple, green, this would be equiva-

lent to individually solving on the subdomains in the following sequence: $\Omega_0, \Omega_2, \Omega_9, \Omega_{11}, \Omega_1, \Omega_3, \Omega_8, \Omega_{10}, \Omega_5, \Omega_7, \Omega_{12}, \Omega_{14}, \Omega_4, \Omega_6, \Omega_{13}, \Omega_{15}$. In this particular case, there are several ways to reorder this sequence while preserving identical results (the order $\Omega_0, \Omega_2, \Omega_1, \Omega_3, \Omega_9, \Omega_{11}, \Omega_8, \Omega_{10}, \Omega_5, \Omega_7, \Omega_4, \Omega_6, \Omega_{12}, \Omega_{14}, \Omega_{13}, \Omega_{15}$ being another).

Lastly, note that multiplicative Schwarz with domain decomposition is closely related to the Gauss-Seidel iterative scheme, and therefore, is expected to converge faster than methods derived from the Jacobi scheme.

3.1.2 Additive Schwarz

We first present the traditional additive Schwarz algorithm:

Algorithm 2 Overlapping additive Schwarz domain decomposition [8]

Given $u^{\{0\}}$ defined on Ω

for $k = 0, 1, 2, \ldots, K-1$ **do**

for $i = 1, 2, \ldots, p$ **do**

Find $\widetilde{u}_i^{\{k\}} \in H^1_{D_k}(\Omega_i)$ such that

$$a_i\big(\widetilde{u}_i^{\{k+1\}}, v\big) = l_i(v), \quad \forall v \in H^1_0(\Omega_i). \tag{3.3}$$

Let

$$u^{\{k+1\}} = (1-\tau p)u^{\{k\}} + \tau\left(\sum_{i=1}^{p} \Pi_i \widetilde{u}_i^{\{k+1\}}\right) \text{ where } \quad \Pi_i \widetilde{u}_i^{\{k+1\}} = \begin{cases} \widetilde{u}_i^{\{k+1\}}, & \text{on } \overline{\Omega}_i, \\ u^{\{k\}}, & \text{on } \Omega \backslash \overline{\Omega}_i. \end{cases} \tag{3.4}$$

end for

end for

The Π_i operator provides a way to prolong the contributions of each subdomain to be of the same dimension as the entire domain so that these can actually be added together. Once $u^{\{k+1\}}$ is

computed, it can be evaluated along the boundaries of each subdomain and imposed as boundary conditions for the next solve. We see that the global solution at any point no longer comes from the solution from just one of the subdomains, but a linear combination of solutions. τ is the variable (determined by the user) used to determine the weight given to the subdomains' new solutions and the remaining weight goes to the equivalent of the old global solution present on the current subdomain, s. τ is a relaxation parameter required for convergence, controlling for potential volatility in our global solution between consecutive cycles.

Like Algorithm 1, Algorithm 2 provides a method to compute the global solution; however, here this is done after each cycle, k, rather than after each subdomain solve. In additive Schwarz, all subdomains can be solved simultaneously; a careful ordering scheme need not be developed. Several of the subdomains can be assigned to each available processor and solved in parallel. Instead of solving on each subdomain individually, they could be solved in batches: solve an optimal number of subdomains on available processors in parallel, store these solutions, solve another batch of subdomains divided among the processors, and repeat this process until all subdomains are solved, completing one cycle of computation. Then, as before, use these solutions as boundary conditions for the next cycle. In this way, the computation time could be cut from $t * n$ to $t * \texttt{ceiling}(\texttt{ceiling}(n/m)/p)$ where n is the total number of subdomains, p is the number of available processors, and m is the maximum number of subdomains that each processor can handle. To derive the expressions for computation time, we have assumed that computation time on all processors for up to m subdomains is the same; balancing problem size of all subdomains by constructing them in such a way that roughly equally divides the total number of degrees of freedom would help with this.

Unlike in multiplicative Schwarz, there does not exist a partition that covers the whole global domain where each region preserves one of the subdomain solutions. Every point in the overlapping regions of the domain is a linear combination of subdomain solution values, not just a copy of a particular one as was the case for multiplicative Schwarz. Therefore, constructing the global solution from subdomain solutions for additive Schwarz is more computationally expensive than

doing so for multiplicative Schwarz, but luckily, is equally unnecessary. Again, the global solution is only needed to impose boundary conditions for subdomain solves, so it only needs to be known along subdomain edges. Our boundary conditions now do not just come from one subdomain, but instead, a linear combination of solutions. So, the ability to implement additve Schwarz actually comes down to if we are able to compute a linear combination of the appropriate subdomain solutions at any given point. We devise a formula to compute the value of the global solution given any point of any edge from subdomain solutions. First we introduce some notation. Choose an edge $\Gamma_{s,R}$ on which we want to compute the global solution, where s is the subdomain on which we will impose the results as a boundary condition and R is the set of subdomains which also contain this edge, i.e. $R := \{j | \Gamma_{s,R} \in \overline{\Omega}_i \cap \overline{\Omega}_j\}$. Denote the global solution along this edge by $u_{s,R}$. Then

$$u_{s,R}^{\{k+1\}} = (1 - \tau p)\tilde{u}_s^{\{k\}} + \tau \left(\sum_{\substack{i \in R \\ i \neq s}} \tilde{u}_i^{\{k+1\}} + \left(p - |R| \right) \tilde{u}_s^{\{k\}} \right) \tag{3.5}$$

where the first term of (3.5) represents the contribution from the previous global solution and the remainder of (3.5) represents the contributions from all of the subdomains' most-recently computed solutions' global extensions, as in the original additive Schwarz algorithm presented earlier. After combining like-terms, (3.5) is equivalently

$$u_{s,R}^{\{k+1\}} = (1 - \tau |R|)\tilde{u}_s^{\{k\}} + \tau \left(\sum_{\substack{i \in R \\ i \neq s}} \tilde{u}_i^{\{k+1\}} \right). \tag{3.6}$$

In noting that additive Schwarz strictly uses old subdomain solutions, similarities between it and the Jacobi algorithm and its convergence behavior can be expected to be comparable. Therefore, one could expect that additive would not converge faster than multiplicative Schwarz.

3.1.3 Restricted Additive Schwarz

We first present the traditional restricted additive Schwarz algorithm:

Algorithm 3 Overlapping restricted additive Schwarz domain decomposition

Given $u^{\{0\}}$ defined on Ω

for $k = 0, 1, 2, \ldots, K-1$ **do**

for $i = 1, 2, \ldots, p$ **do**

Find $\widetilde{u}^{\{k\}} \in H^1_{D_k}(\Omega_i)$ such that

$$a_i\big(\widetilde{u}^{\{k+1\}}, v\big) = l_i(v), \quad \forall v \in H^1_0(\Omega_i). \tag{3.7}$$

Let

$$u^{\{k+1\}} = \begin{cases} \widetilde{u}^{\{k\}}, & \text{on } \overline{P}_i, \\ u^{\{k-1\}}, & \text{on } \Omega \backslash \overline{P}_i \end{cases} \text{ where } P_i \text{ is the partition of the global domain contained entirely in } \Omega_i. \tag{3.8}$$

end for

end for

Unlike Algorithms 1 and 2, Algorithm 3 does not directly use the geometry of the subdomains to construct the global solution. Instead, we construct a partition of the entire domain, identify which subdomain entirely contains the given partition and preserve that subdomain's solution on that region of the partition, thus tiling the global solution together. Taking the previous domain construction with four subdomains in a 2x2 configuration, we similarly present the global solution by labeling which subdomain solutions are preserved in particular regions of the global domain

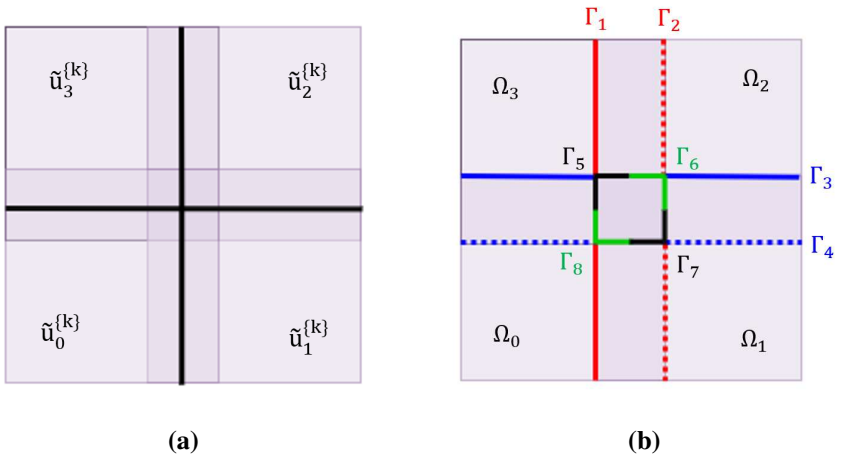


Figure 3.4: (a) Partition preserving specific subdomain solutions in each region is overlaid on global domain. $\tilde{u}_i^{\{k+1\}}$ indicates that subdomain i's solution from cycle $k+1$ is preserved in the given region. (b) Domain with edge boundary_ids indicated for multiplicative Schwarz implementation.

This scheme can also be stripped of the step in which the global solution is constructed and boundary conditions can be imposed directly from subdomain solutions from the previous sweep, as summarized in Table 3.2.

In comparing Tables **??** and 3.2, it should be noted that all sources of boundary conditions are the same between the multiplicative and restricted additive Schwarz algorithms except for those around the area of high overlap. Although the subdomains from which boundary conditions are imposed on particular edges are the same between algorithms, multiplicative Schwarz uses information from the current sweep rather than the previous one, ensuring convergence that is at least as fast as that of restricted additive. What then are the benefits of the restricted additive Schwarz algorithm? Firstly, it is restriced additive Schwarz' use of old solutions that makes it parallelizable, which can substantially cut elapsed time and can split computational time among processors. Additionally, the use of old solutions can significantly cut computational expense for symmetric problems (symmetric in terms of boundary conditions, domain geometry, and right hand side of the partial differential equation to be solved) whose lines of symmetry correspond to the lines defining the global partition for global solution construction (Figure 3.4 (a)). In this case, the problems solved on each subdomain are identical and the problem size can be reduced using

Table 3.2

Subdomain	Edge	Subdomain solution imposed as a BC
Ω_0	Γ_3	Ω_3
	Γ_5	Ω_3
	Γ_6	Ω_2
	Γ_7	Ω_1
	Γ_2	Ω_1
Ω_1	Γ_1	Ω_0
	Γ_8	Ω_0
	Γ_5	Ω_3
	Γ_6	Ω_2
	Γ_3	Ω_2
Ω_2	Γ_4	Ω_1
	Γ_7	Ω_1
	Γ_8	Ω_0
	Γ_5	Ω_3
	Γ_1	Ω_3
Ω_3	Γ_2	Ω_2
	Γ_6	Ω_2
	Γ_7	Ω_1
	Γ_8	Ω_0
	Γ_4	Ω_0

symmetry arguments. In theory, one could solve on only one subdomain, impose its own solution as boundary conditions on itself for the next solve, and then appropriately reflect the solution to the remainder of the domain.

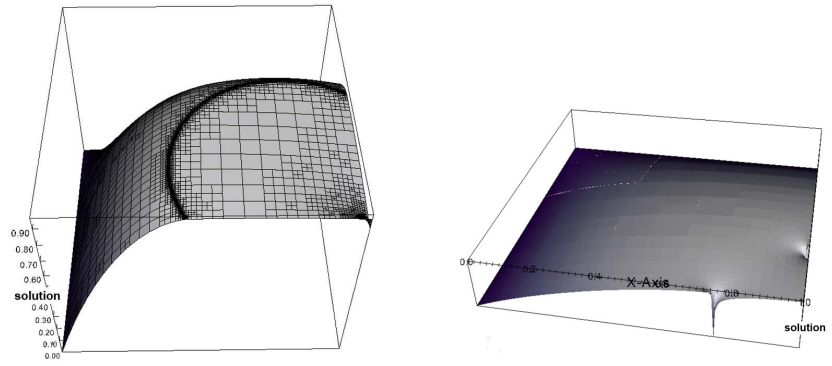
Lastly, we note that despite the similarities between the multiplicative and restricted additive Schwarz, the mathematical foundation of restricted additive algorithm comes from the Jacobi scheme which also uses old solutions rather than most-recently computed ones and therefore is expected to converge no faster than multiplicative Schwarz.

3.2 Implementation of Schwarz Algorithms in deal.II

We conceptually discuss how to implement domain decomposition algorithms in both the deal.II software (built with C++) before solving specific problems and analyzing performance in Section 4.

3.2.1 Multiplicative Schwarz

As discussed in Section 3.1.1, the solution on the entire domain Ω need not be computed; it is sufficient to compute solutions on each subdomain and impose their solution as a boundary condition to a neighbor where the two overlap. This fact is taken advantage of in my deal.II implementation and is done using the deal.II VectorTools::interpolate_boundary_values function. It is imperative that the appropriate solution (namely, the most recently computed one at a given point) be used in the imposing of boundary conditions so that continuity is enforced and singularities as in Figure 3.5, avoided. The proper solution to use as the `boundary_function` input of `interpolate_boundary_values` is given by the `get_fe_function` function.



(a) **(b)**

Figure 3.5: Singularities arise when inappropriate solutions are imposed as boundary conditions. Singularity in (a) is less severe than that depicted in (b), but can be detected by the high amount of mesh refinement in the bottom right corner of its domain.

Since only the most recent solution from each subdomain i is used, memory space could be saved by only storing one subdomain solution at a time and simply writing over them during each cycle. The drawback to the multiplicative Schwarz algorithm however is that no neighboring subdomains can be solved in parallel. For illustrative purposes, suppose that neighboring subdomains were to be solved in parallel. In this case, the only solutions available to impose as boundary conditions from one to the other is from the previous cycle k, not the current one. Therefore, if these neighboring subdomains are solved simultaneously, the solution $u^{\{k+i/p\}}$ that would essentially (but not in practice) be used in this case would be

$$u^{\{k+i/p\}} = \begin{cases} \tilde{u}^{\{(k-1)+i/p\}} & \text{on } \overline{\Omega}_i \\ u^{\{k+(i-1)/p\}} & \text{on } \Omega \backslash \overline{\Omega}_i \end{cases}$$

which is not in agreement with the multiplicative Schwarz algorithm 1.

Once we solve on the subdomains, we can (but are not required to) construct the global solution. This is done in the current program merely to enable comparison in convergence time with the original deal.II global implementation. This is currently being done by constructing a uniform global mesh after every cycle of solves and interpolating the subdomain solutions onto this mesh

according to Figure 3.2. The refinement level of the global mesh is such that all of its cells are as small as the smallest cell present on any of the subdomains during the most recently-completed cycle. The global solution in the restricted additive code is constructed in the same way.

3.2.2 Additive Schwarz

The solutions $\tilde{u}_i^{\{k+1\}}$ for $i \in R$ are retrieved by the `get_overlapping_solution_functions` function and stored in the `overlapping_solution_functions` vector, which is used in the `MyOverlappingBoundaryValues<dim>` constructor. It is here that (3.6) is actually computed. An object of the `MyOverlappingBoundaryValues<dim>` type is passed to `interpolate_boundary_values` in `Step6::assemble_system`, which provides this object the `Point<dim>`s on which to compute.

3.2.3 Restricted Additive Schwarz

Although the name of the restricted additive Schwarz algorithm gives the impression that it is more similar to additive Schwarz than multiplicative, it can be thought of as a modified version of either algorithm. This flexibility allows the user the option to leverage either the convergence speed of multiplicative Schwarz or the parallelizability of additive Schwarz, and significantly helps with its implementation from preexisting software. To implement restricted additive Schwarz in deal.II, I adapted the multiplicative Schwarz code. The difference between the two implementations is in the construction of the subdomains' initial meshes and which solutions are imposed on particular edges around the center of our global domain. As is apparent when comparing Figure 3.2 and Figure 3.4, the domain requires more boundary ids to impose the boundary conditions for the restricted additive Schwarz algorithm. Therefore, subdomain mesh construction time during the first cycle of the program takes more time for restricted additive Schwarz than multiplicative.

Chapter 4

Results

4.1 Comparison of deal.II implementations with and without domain decomposition

Test Case 1

To compare the original deal.II code to that using domain decomposition, we solve the following partial differential equation modeling a force, like water flow for example, pushing on a membrane. The PDE is

$$\begin{aligned} -\nabla \cdot (a \nabla u) &= f \quad \text{on } \Omega \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

where we define $a(\mathbf{x})$ as the discontinuous function

$$a(\mathbf{x}) = \left\{ \begin{array}{ll} 20 & \text{if } 0.25 > \mathbf{x} \cdot \mathbf{x} = \sum_{i=1}^{n} x_i^2 \text{ where n is the dimension of } \Omega \\ 1 & \text{otherwise} \end{array} \right\}$$

describing the material of the membrane and define $f(\mathbf{x})$ as the constant function

$$f(\mathbf{x}) = 1$$

describing a constant forcing function. Further, we define Ω as the square centered about the origin.

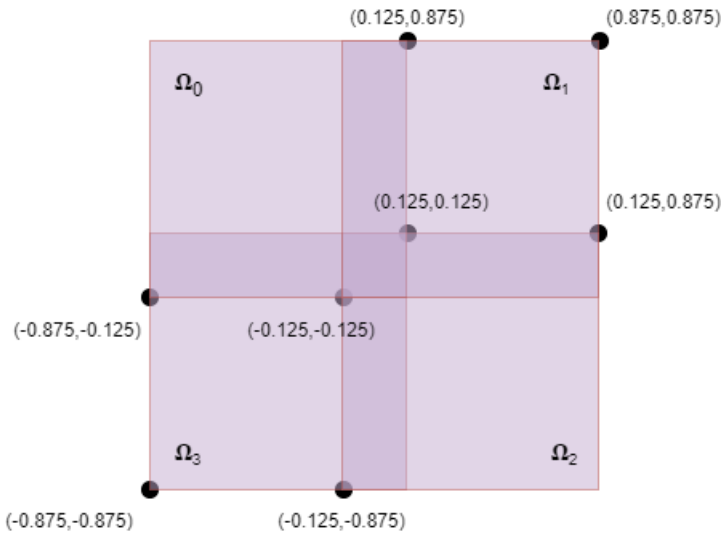
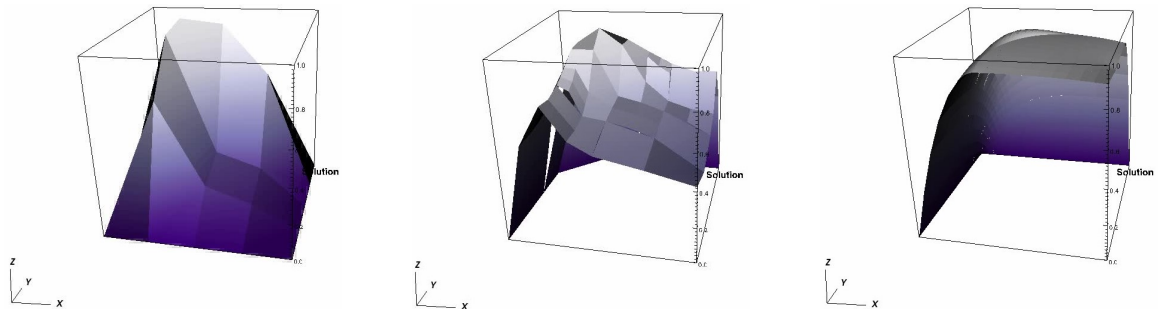


Figure 4.1: Test case 1 layout

In our test case, we solve the Laplace equation with a forcing term. Note that there is a region where all subdomains overlap each other, meaning that parallelization is not possible with the multiplicative Schwarz algorithm.



(a) **(b)** **(c)**

Figure 4.2: Solutions on Ω_0 from progressive cycles from left to right.

Problem Size

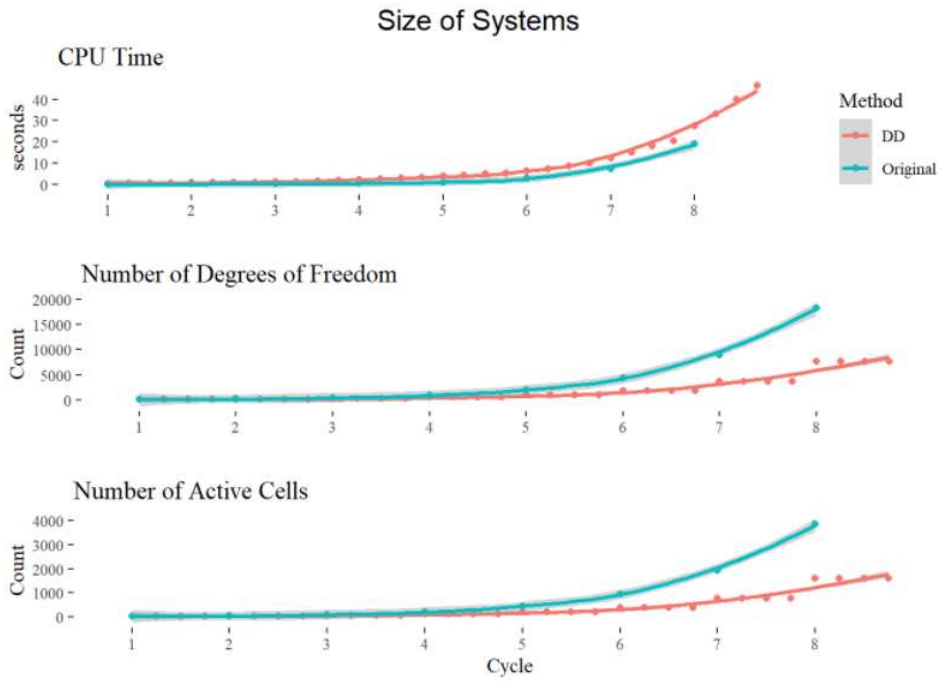


Figure 4.3: Comparison of the size of the systems solved using domain decomposition with sequential multiplicative Schwarz versus the original program

4.2 Comparison of deal.II implementations of two DD methods

Test Case 2

Now we solve:

$$\begin{cases} -\nabla \cdot (a\nabla u) = f & \text{on } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \quad \text{with } a(\mathbf{x}) = 1 \text{ and } f(\mathbf{x}) = 8\pi^2 \sin(2\pi x_1)\sin(2\pi x_2), \tag{4.1}$$

defining Ω as the unit square with the bottom left corner at the origin and the subdomains as having an overlap of 0.2.

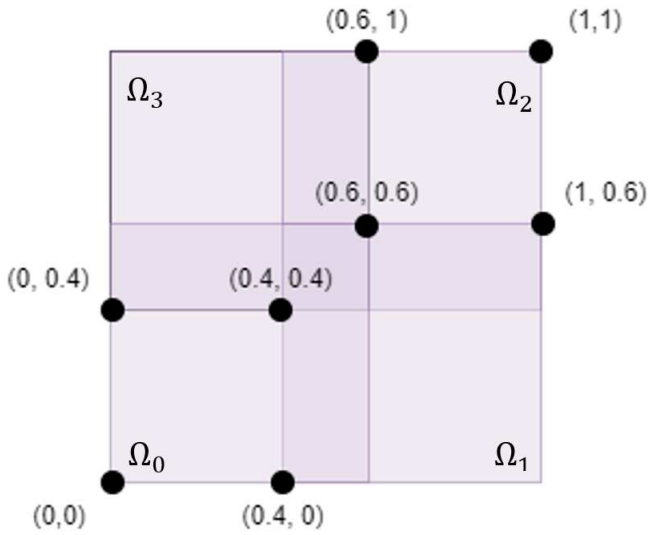
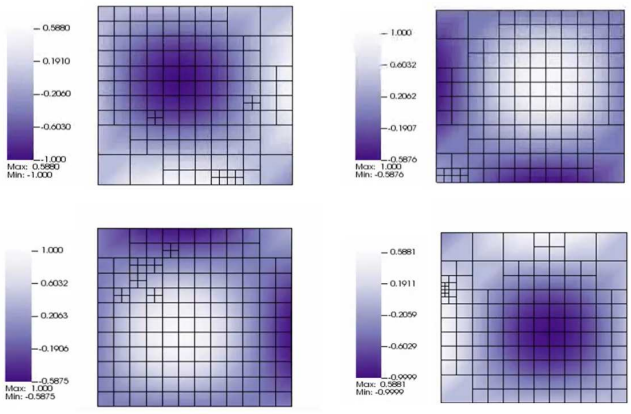


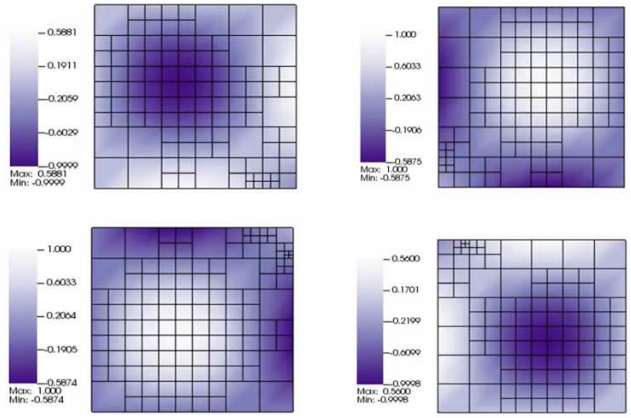
Figure 4.4: Test case 2 layout

4.2.1 Convergence

Experimentation results followed intuition provided by algebraic analysis. Multiplicative Schwarz did in fact converge toward the exact solution faster than restricted additive Schwarz. From numerically solving (4.1), we know that the largest absolute solution value on any of the subdomains should be 1.0. More specifically, subdomains 0 and 2 should have maximum solution values of 1.0 and subdomains 1 and 3 should have minimum solution values of -1.0. By the eighth cycle, Figure 4.5 shows that multiplicative Schwarz led to maximum solution values of 1.000 on subdomains 0 and 2, a minimum value of -0.9999 on subdomain 1, and a minimum value of -1.000 on subdomain 0. Comparatively, restricted additive slightly underperformed: subdomains 0 and 2 still had maximum values of 1.000, but subdomain 1 had a minimum solution value of -0.9998 and subdomain 3 had a minimum solution value of -0.9999. Although the difference in computational accuracy is negligable, it is important to remember that these two algorithms only differ in which edges retrieve their boundary conditions from particular subdomains in highly overlapping regions and whether the solutions imposed as boundary conditions are from the current cycle of solves or the previous one. Since multiplicative Schwarz uses most updated solutions, information travels



(a)



(b)

Figure 4.5: Meshes at the end of cycle 8 of solving (4.1) on each of the four subdomains using the (a)multiplicative Schwarz and (b)restricted additive Schwarz algorithms.

faster across the global domain using this algorithm over restricted additive, a fact that further exacerbates this difference in performance for larger domains split into more subdomains.

To accurately quantify and observe convergence of subdomain solutions to their respective exact solutions, it is important to consider the structure of the exact solution on

4.2.2 Problem Size

The initial meshes for the two algorithms are identical; the difference in problem size between the two is introduced through adaptive refinement. Refinement is being driven by error (computed using the Kelly error estimator) between adjacent cells where the 7% of cells with the highest error

are refined (by being split into four children cells geometrically similar to their parent cell) and the 3% of cells with the lowest error are coarsened. Since the only difference in the two algorithms is in highly overlapping regions, it is reasonable to expect that the solutions to be highly similar, the difference in solution values across cells to be highly similar, and therefore for adaptive refinement to produce similar meshes, resulting in a similar number of degrees of freedom from one cycle to the next. Indeed, Figure 4.6 shows that the total number of degrees of freedom among subdomains are comparable among subdomains with the only notable difference being in the number of degrees of freedom present in subdomain 0 between the two domain decomposition methods. Despite the similarity in total number of degrees of freedom among subdomains between algorithms, multiplicative Schwarz consistently performed better in terms of CPU time. Algebraic analysis shows that multiplicative Schwarz converges faster to the exact solution in terms of number of cycles and experimental results like those summarized in Figure 4.6 show evidence that it also converges faster in terms of CPU time. Therefore, parallelization is not only an option in restricted additive Schwarz implementation, but a requirement to make it advantageous over multiplicative Schwarz. It should be noted that for this numerical experiment, restricted additive Schwarz results in subdomains of roughly the same number of degrees of freedom whose respective problems require similar amounts of CPU time to solve. The current geometric construction of the subdomains seems appropriate for computational load balancing, but for less symmetric problems, redefining subdomains may result in more optimal parallelization schemes.

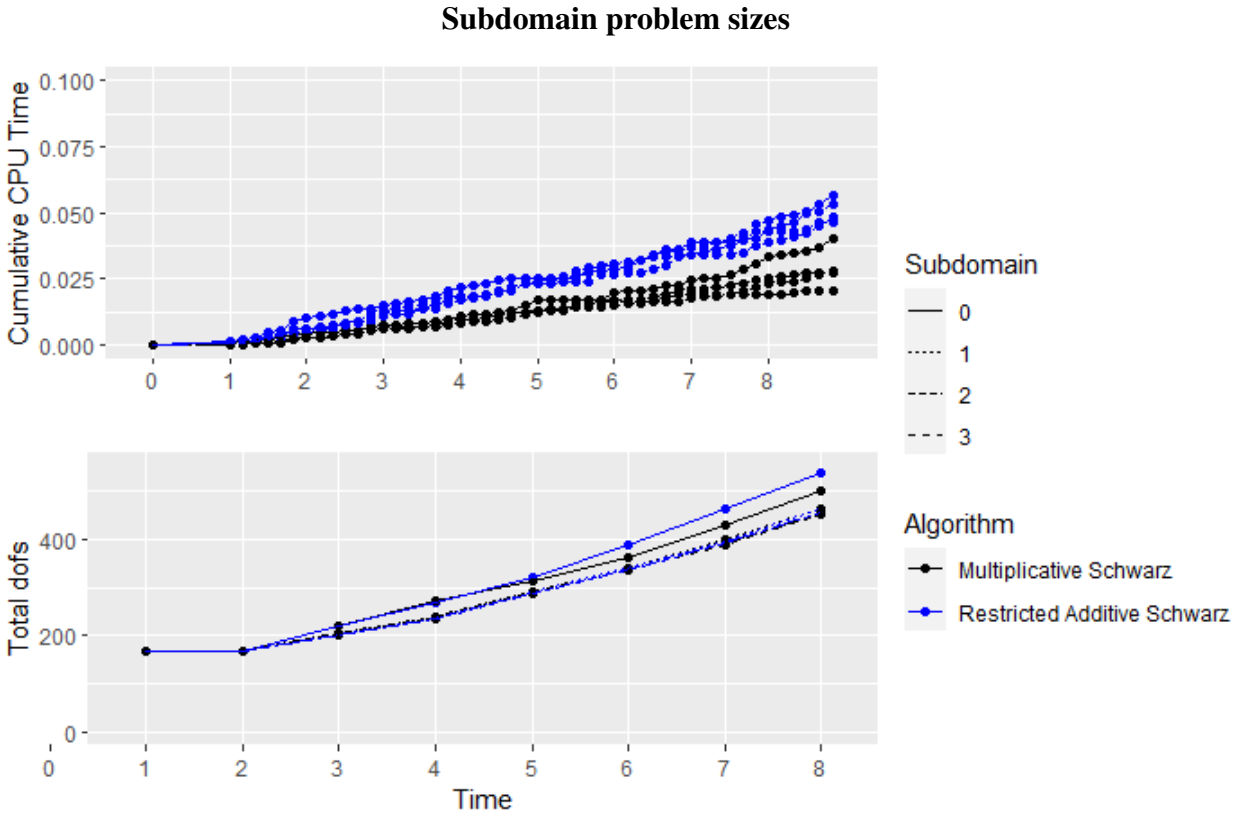


Figure 4.6: Cumulative CPU times and total degrees of freedom of each of the four subdomains for both the multiplicative and restricted additive Schwarz algorithms overlaid on each other.

Chapter 5

Conclusion

5.1 Improvements in Code Implementation

In my deal.II implementation, I am constructing the global solution by creating a new, uniform triangulation after every cycle of subdomain solves whose cell size is dependent on that of all four subdomains. Specifically, the global mesh size is that of the highest refinement level present on any of the subdomains so as to not throw away details of the solution in regions of higher refinement, details that we have already spent computational time and capacity finding. I preserved these details so that comparisons between the solution using domain decomposition and that of the original code could be compared accurately, without the construction of the global solution (which is only done for analysis) being a source of difference. This decision came at a high cost: the number of degrees of freedom present in the global mesh is at least as many as the number of degrees of freedom from each subdomain, which together describe the solution on same area ($number\_of\_dofs(u^{\{k\}}) \geq \sum_{i=1}^{n} number\_of\_dofs(\widetilde{u}_i^{\{k\}})$), the two only being equal
in the unlikely case that adaptive refinement leaves every subdomain's mesh as a uniform one. Otherwise, adaptive refinement introduces higher levels of refinement in areas of discontinuity (often introduced by way of sudden changes in forcing function over space or time, membranes made of materials with different properties of malleability, etc.) and significantly increases the number of degrees of freedom on our new global triangulation on which we need to interpolate subdomain solutions as shown in Figure 5.2. This can be a time consuming process as shown in Figure 5.1. Instead, we should take advantage of the fact that the subdomain triangulations are only highly refined in certain regions and can often remain relatively coarse in others. Three potential ways of doing this come to mind, all of which are different from the current implementation in how the global triangulation is constructed, but interpolate the appropriate subdomain solutions as before, allowing us to preserve the current code that achieves this (using `set_material_id` and

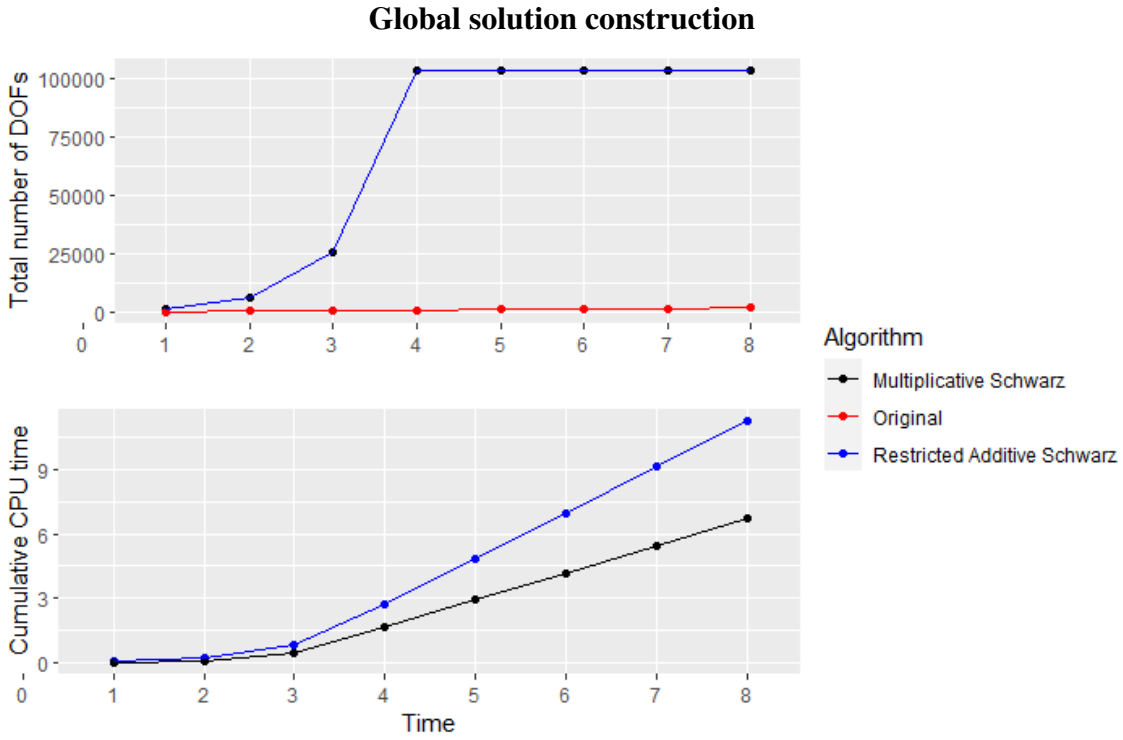


Figure 5.1: CPU time and total number of degrees of freedom needed to construct the global solution for each cycle using the original code, restricted additive Schwarz, and multiplicative Schwarz algorithms.

`interpolate_based_on_material_id`). (1) and (2) generate global meshes with cells of the same size while (3) would preserve the sizing that results from adaptive refinement during the subdomain solves:

1. Define a static global triangulation globally refined a specified number of times, independent of the refinement level of any of the subdomains. This would result in a loss of detail around discontinuities, but would substantially decrease the computational time involved in constructing the global solution and eliminate the need to construct a new global triangulation after every cycle of solves.

2. We could instead have the global mesh consist of cells that are the mean or median size of cells in all of the subdomains. This could be done by looping through all coarse cells (the cells of the uniform mesh when the subdomain triangulation was created), storing in a vector the number of child cells that each of these coarse cells have, concatenating such vectors of all subdomains together, divide by the number of child cells created by each level

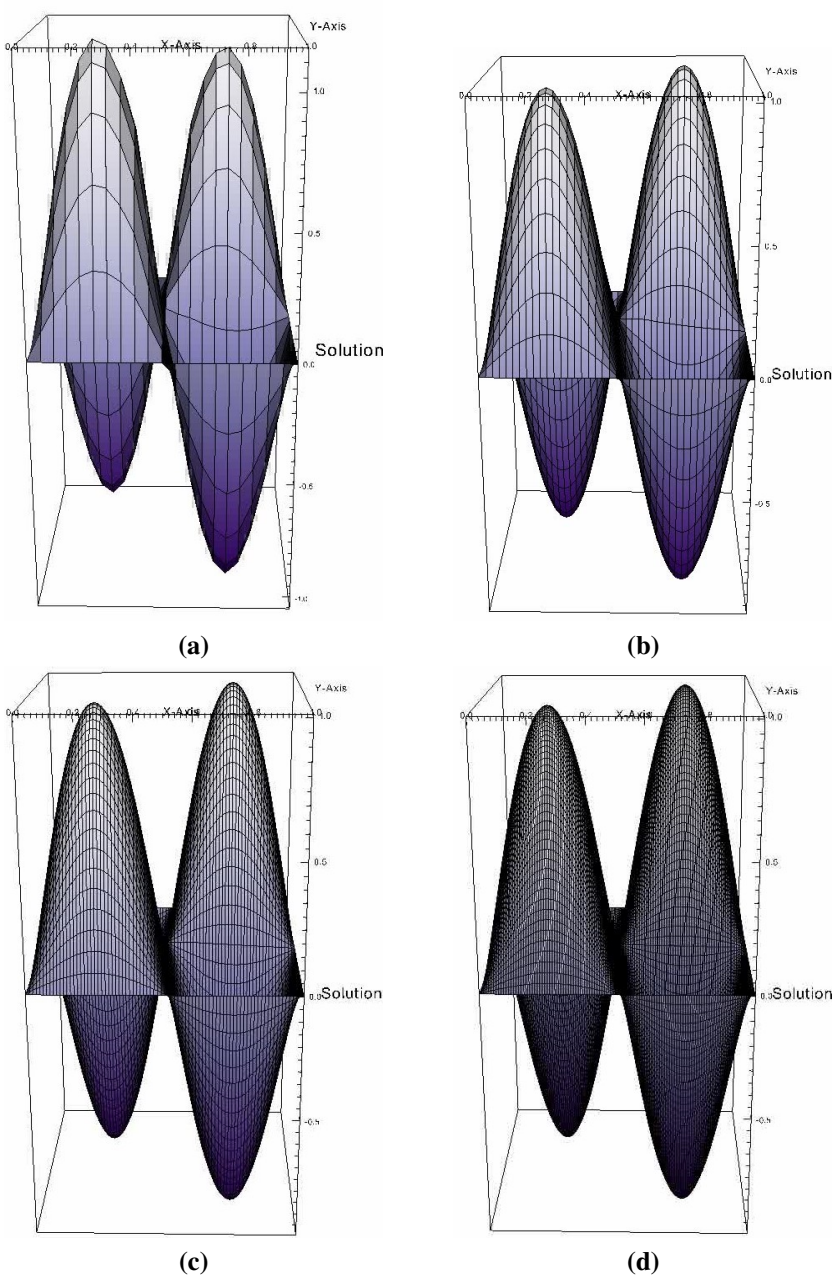


Figure 5.2: Global solution using the multiplicative Schwarz algorithm (that of restricted additive is similar) from cycles 1 through 4 showing the rapid increase in total number of degrees of freedom, especially from cycle 3 to cycle 4.

of refinement (4 in 2D), compute the mean/median of this vector, and globally refine the one-cell unit square global mesh this number of times. Similar to (1), this would also cut computation time, but would still require constructing a new global triangulation after every cycle of solves in the name of preserving a higher level of detail in the solution.

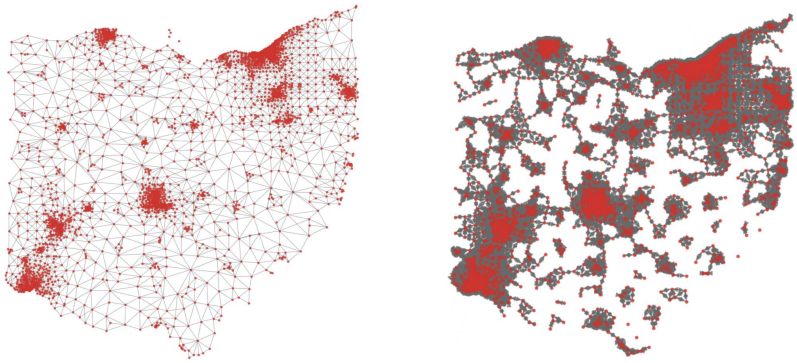
3. Copy all subdomain triangulations and only keep cells of these triangulation copies that tile together the global solution. For example, if solving on a square domain subdivided as in Figure 3.1, we would keep cells from Ω_i that live in the area of Ω where $\widetilde{u}_i^{\{k+1\}}$ is defined (shown in Figure 3.2) by removing all other cells from each subdomain's triangulation. This can be done using `GridGenerator::create_triangulation_with_removed_cells` which removes any cells in a `cells_to_remove` set. This function requires that the input triangulation consists of cells that are on the same refinement level. Because our subdomains are being solved with adaptive refinement, this requirement is not guaranteed to hold. The subdomain triangulations would need to be flattened to the same refinement level using `GridGenerator::flatten_triangulation` and then cells to remove from these new triangulations would be added to `cells_to_remove` with the use of a for loop and if statements (refer to https://dealii.org/developer/doxygen/deal.II/code_gallery_Quasi_static_Finite_strain_Quasi_incompressible_ViscoElasticity.html as an example), after which `GridGenerator::create_triangulation_with_removed_cells` could be called. Once all pieces of the global domain's partition defined by preserved subdomain solution have been created, they can be merged together with `GridGenerator::merge_triangulations`. Now we have a global triangulation that preserves the cell sizes of the subdomain triangulations on which we can interpolate the appropriate subdomain solutions as before, allowing us to preserve the current code that achieves this (using `set_material_id` and `interpolate_based_on_material_id`. It should be noted that this method generates multiple new triangulations after every cycle, so memory capacity and computation time of the mentioned functions should be weighed against the computational time saved by eliminating the need to interpolate solutions on extra degrees of freedom.

5.2 Future Work

As part of the Air Force Research Laboratory's ACT3 efforts to better understand the complex propagation of threats, Matrix Research has been collaborating on the development of agent-based modeling software and the analysis of the resulting synthetic data. A brief introduction to agent-based modeling is presented and enough details about the work currently being done in this area at Matrix Research to convey the improvements that domain decomposition techniques could help achieve by way of the algorithms and code previously discussed.

5.2.1 Background: Agent-based modeling

Recently, especially with the emergence and proliferation of COVID-19, there has been a surge in demand for mathematical modelling studies [19]. Such studies are vital to inform evidence-based decisions by health decision- and policy-makers [20], so their accuracy is of great importance. Classical epidemiological models are structured as differential equations which assume an unrealistic level of homogeneity among the ebbing and flowing population groups: the susceptible, the infected, and the recovered/removed. Among these groups everyone is identical– exhibiting the same behavior and consistent decision-making over time [21]. These models and the behaviors of the individuals they aim to describe simply do not capture the complexity of real social networks and interactions among individuals whose behaviors adapt over time, which can hugely effect disease progression [21]. Agent-based models are able to simulate such complexity in behavior and can therefore capture the spread of disease more accurately. These models require construction of a virtual world about which the agents move and interact. The prototype implemented by ACT3 researchers has a virtual world representing Ohio described by the directed property graph structure in Figure 5.3 about which agents move according to behavior present in Ohio-specific mobility data.



(a) (b)

Figure 5.3: (a)Ohio census tracts world with shared border edges and travel weights assigned by distance. (b)Ohio census tracts navigable small worlds with edges assigned by a distance probability function. [1]

5.2.2 Application

Every additional level of detail added to the way in which agents traverse the virtual world in these simulations lead to a more accurate representation of how people commute and interact in real life. Although progress has been made in making the agent behavior mimic that of our own, there are some inherent differences between analysis and verification of data from a simulation and data from the real world. In simulations, the user has the ability to extract data that is unknowable in the real world. For example, when analyzing the diffusion of disease, it is possible to know exactly which interaction led to a new infection of an agent. Further, it is possible to even construct exact social network graphs that describe which agents have come into contact with each other during a simulation. The use of personal data from smart devices allow for partial construction of such graphs, but such data is inaccessible for DOD efforts such as this. More concisely put, with real world diffusion, patterns are observable and strong channels of diffusion can be established, but to represent diffusion as a graph requires a level of precision that is not possible outside of simulations. Additionally, in these graphical simulations, travel, and therefore agent-facilitated diffusion, is restricted to edges and interactions restricted to nodes of the graph that construct the virtual world. Both our travel and location of potential contact with others are not restricted in the same manner, which are both differences worth considering. Lastly, agent-based models can be

computationally expensive, are typically stochastic requiring multiple realizations, can be difficult to analyze and to draw conclusions from such as sensitivity information without repeated solution.

In moving from a virtual world to the real world, shifting from modeling the spread of threats such as the COVID-19 pandemic as graphical diffusion to convection-diffusion, with highly traveled edges of the graph defining stronger convection channels, would be a choice worth exploring and is one that allows for the use of our iterative solvers for partial differential equations.

Bibliography

[1] William Aue, Matthew Barnes, Benjamin Bengfort, Dustin Dannenhauer, Matthew Molineaux, and Karleigh Pine. Targeted COVID-19 Interventions with an Ohio-Scale Agent Based Simulation, July 23, 2020.

[2] David Keyes. Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations (May 6-8, 1991). Society for Industrial and Applied Mathematics proceedings series. Philadelphia, 1992.

[3] Alejandro Mota, Irina Tezaur, and Coleman Alleman. The Schwarz Alternating Method in Solid Mechanics. Sandia National Laboratories, February 6, 2017.

[4] Vidar Thomee. From finite differences to finite elements: A short history of numerical analysis of partial differential equations. journal of computational and applied mathematics, vol. 128(1-2). pp. 1-54.

[5] Pieter Wesseling. An Introduction to Multigrid Methods. John Wiley and Sons Ltd., 1992.

[6] Barry Smith. Domain Decomposition Methods for Partial Differential Equations. In: Keyes D.E., Sameh A., Venkatakrishnan V. (eds) Parallel Numerical Algorithms. Interdisciplinary Series in Science and Engineering, Vol 4. Springer, 1997.

[7] Petter Bjorstad, William Gropp, and Barry Smith. Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, 1996.

[8] Jahanzeb Chaudhry, Donald Estep, and Simon Tavener. A posteriori error analysis for Schwarz overlapping domain decomposition methods.

[9] Tony Chan and Tarek Mathew. Domain decomposition algorithms. acta numerica. pp. 61-143. 1994.

[10] Xiao-Chuan Cai and Youcef Saad. Overlapping domain decomposition algorithms for general sparse matrices. https://www.colorado.edu/faculty/cai/sites/default/files/attached-files/ogd.pdf.

[11] James Demmel. Applied Numerical Linear Algebra. Society for Industrial and Applied Mathematics, 1997.

[12] David Kincaid and Ward Cheney. Numerical Analysis: Mathematics of Scientific Computing, Second Edition. Brooks/Cole, 1996.

[13] James W. Demmel. Applied Numerical Linear Algebra. Society for Industrial and Applied Mathematics, 1996.

[14] Aurel Galantai. Projectors and Projection Methods. Springer, 2004.

[15] Alston S. Householder. The Theory of Matrices in Numerical Analysis. Dover Publication, Inc., 1964.

[16] Yousef Saad. Iterative Methods for Sparse Linear Systems, 2nd Edition. Society for Industrial and Applied Mathematics, 2003.

[17] David Lucia, Paul King, and Philip Beran. Domain decomposition for reduced-order modeling of a flow with moving shocks. American Institute of Aeronautics and Astrocautics, 40, 11, November 2002.

[18] J. N. Reddy. An Introduction to the Finite Element Method, Second Edition. McGraw-Hill, Inc., 1993.

[19] M Egger, L Johnson, C Althaus, A Schöni, G Salanti, N Low, and S Norris. Developing WHO guidelines: Time to formally include evidence from mathematical modelling studies, August 29, 2017.

[20] B Tang, X Wang, Q Li, N Bragazzi, S Tang, Y Xiao, and J Wu. Estimation of the Transmission Risk of the 2019-nCoV and Its Implication for Public Health Interventions. Journal of Clinical Medicine, 9, Feb 2020.

[21] Josua Epstein. Modelling to Contain Pandemics. Nature, 460, 687, 2009.